Aura Microwave Limb Sounder (MLS)
IDL Callable Forward Model Interface Requirements Document

H. H. Nguyen

v0.0.1, April 20, 2011

Contents

1 Introduction and background

This document describes the external interface for, and other details of, the Aura Microwave Limb Sounder (MLS) ‘IDL Callable Forward Model’ (ICFM hereafter). More information on Aura MLS can be found in Waters et al. [2006]. ICFM is an IDL programming interface to the ’Callable Forward Model’ (CFM hereafter), of which more information can be found in Nguyen et al. [2010]. ICFM allows users to call CFM from IDL by using PVM to communicate between IDL and Fortran processes.

Section 2 reviews the overall structure of the CFM, including an outline of relevant underlying aspects of the MLS instrument, operations and data processing algorithms. This information can also be found in Nguyen et al. [2010]. Section 3 serves as a reference for the various data types and procedures that constitute the ICFM software. Appendix A gives examples of code that invokes the foward model that can serve as a starting point for assimilation studies etc. Finally, Appendix B provides tables and lists of relevant terms (units etc.).

2 Review of overall MLSCFM structure

This section introduces the various data entities and functionalities associated with the MLSCFM software. More details on specific derived type definitions and function / subroutine invocation are given in Section 2.

2.1 MLS radiance measurements and related parameters

The calibrated MLS radiance measurements are stored, along with radiance geolocation information, in the MLS Level 1B (L1 hereafter) files. The L1 files of relevance to MLSCFM applications are the radiance files and the orbit/attitude file. These files typically have a daily granularity running from midnight to midnight UT.

2.1.1 Major frames and minor frames

The principal divisions within the MLS L1 files are known as *major frames* (MAFs) and *minor frames* (MIFs). A major frame is one complete vertical scan of the MLS GHz and THz antennas (including limb scanning, calibration and retrace activities). The antennae move continuously during a MAF and radiances are integrated at 1/6 s intervals known as minor frames (MIFs). During a standard scan, limb views are taken for the first 120 MIFs (114 for the THz antenna). The remaining ∼26 MIFs in the MAF are spent performing calibration and antenna retraces. The on board software attempts to lock the MLS scans to the Aura orbit, such that scans are made at essentially the same latitudes each orbit. In order to accomplish this, occasional ‘leap MIFs’ must be added in selected MAFs.

2.1.2 Bands and channels

The other divisions of relevance for L1 data are band and channel. MLS makes measurements in 34 spectral bands, each of which has from 4 to 129 channels. The bands are described using the MLS ‘signal designation’ nomenclature

```
[Radiometer name].[Band].[Switch].[Spectrometer]
```

Examples are `R1A:118.B1F:PT.S0.FB25-1`, or `R3:240.B7F:O3.S0.FB25-7`. The complete list of MLS bands is given in the `MLS-Aura_L2Cal-Signals_...` file supplied with the software. For most bands, the number of channels is quoted in the `Spectrometer` clause (`FB25`, `MB11` or `WF4`) and are numbered 1...*n*. The exception are the Digital Autocorrelator Spectrometers (DACs) which have 129 channels numbered 0...128.

2.1.3 MLS L1 files

Three radiances files are produced for each day of MLS observations. The L1BRADG file contains the radiances for the majority of the GHz bands. The L1BRADD file contains the radiances for the four MLS digital autocorrelator spectrometers, also measuring GHz spectral regions. The L1BRADT file contains radiances measured by the MLS THz bands.

L1BRAD files are stored in HDF5 (not HDF-EOS) and are flat files containing many HDF arrays. The most important of these are radiances which are named according to the signal designation nomenclature described above. The arrays are dimensioned channel, MIF, MAF with channel the most rapidly changing index. Precisions are stored for each radiance, in a separate SD with the name ‘`<signal> precision`’ (note the intervening space). Radiances that the MLS Level 1 software deems should not be used are so indicated by having their precisions set negative. The additional quantity ‘`<signal> Baseline`’ values (dimensioned channel, MAF) should be added (for each MIF per channel/MAF) before radiances are used.

Similarly, the '`<signal> Baseline precision`' should be added in quadrature to the precisions. (The baseline 'AC' and 'DC' terms are diagnostic and may be ignored). All numbers refer to brightness temperature in Kelvins.

The L1BOA (orbit/attitude) file contains information on spacecraft location and the locations of the GHz and THz tangent points. All angles are degrees and all dimensions are meters. Time is the EOS standard TAI93 time (number of seconds, including leap seconds, since midnight UT on January 1st, 1993. Quantities are mostly dimensioned MIF, MAF, with vectors dimensioned 3, MIF, MAF.

2.1.4 A note on `GeodAngle`

An important quantity for MLS L1 (and L2) data is the so-called 'Orbit Geodetic Angle' (referred to in the software as `GeodAngle` or `Phi`). This is used as a horizontal coordinate for most quantities, and is defined as the geodetic great circle angle along the Aura orbit track, increasing with time from the first ascending equator crossing of the day (defined as 0°). Unusually for an angle, this number accumulates beyond 360°. So, the first complete orbit in the day is $0^\circ - 360^\circ$, the second is $360^\circ - 720^\circ$ etc.

2.2 Constructing the MLS state vector

The MLS state vector consists of a heterogeneous collection of quantities, some geophysical, some instrument related. As with the measurement vector (which mainly consists of MLS L1 radiances), these are contained in Fortran 90 derived types consisting of 'quantities' and 'vectors' as detailed in the following subsection.

The principal geophysical quantities in the state vector are consecutive vertical profiles of atmospheric temperature and composition on a regular grid in geodetic angle and pressure (actually $-\log_{10}(p/\text{hPa})$). Other important state vector quantities are the 'tangent pressures' for the MLS GHz and THz limb views, which are dimensioned MIF, MAF (only the GHz is likely to be of interest in most CFM applications).

2.3 Quantities and Vectors in MLSCFM

The heterogeneous nature of the MLS state and measurement vectors dictates that a flexible system be used for their description. In the MLSCFM software vectors consist of 'templates' and 'values'. Separating the vector template from its values allows for efficient storage of related vectors (e.g., the state, the initial guess, and the *a priori*, or the measurements and their forward model estimates). Separating the template from the values also aids more rigid error checking, such as only allowing to vectors to be added if they share a template. Vector templates are, in turn, composed of one or more 'quantity templates'. While quantities do not, strictly speaking, exist in isolation from vectors, the term 'quantity' is often loosely used in the software to refer to either a specific quantity within a vector, and/or its template.

Vector quantities typically describe a single- or multi-valued physical quantity (temperature profiles, abundances for one species, radiances for one band). They can have up to three dimensions, named 'channels', 'surfaces' and 'instances' (channels is the fastest changing index, instances the slowest). Instance is a time-like or horizontal dimension, and typically refers to either profiles (for geophysical quantities) or major frames (for instrument quantities such as measured radiances). Similarly, 'surfaces' can refer to pressure levels for geophysical quantities, or minor frames for instrument quantities. The channels dimension is typically only used for radiance quantities. The actual Fortran arrays storing vector quantity values are dimensioned `( channels * surfaces, instances )`. While increasing the complexity of access for multi-channel quantities, this simplifies the storage of matrices in the software, as discussed below.

'Coherent' quantities are those where the surfaces are the same for every instance. Geophysical quantities, being reported on a fixed pressure grid, are coherent, while radiance quantities where each MAF involves views of different altitudes, are incoherent.

'Stacked' quantities are those where the different surfaces in each instance share the same horizontal geolocation (latitude, longitude, time etc.). Again, geophysical quantities are typically stacked, with the quantity describing vertical profiles, while radiance and tangent pressure quantities are unstacked, as each minor frame within the scan has a slightly different geolocation.

As stated above, geophysical quantities are generally both coherent and stacked, while 'minor frame' (i.e., more instrument related) quantities are always incoherent and unstacked. Two 'minor frame' vector quantities describing parameters associated with the same MLS module (GHz, THz or spacecraft) can, by definition, be assumed to have the same geolocation. This important assertion is relied upon throughout the forward model software. Conversely, for most geophysical quantities, geolocation information does not need to be consistent from quantity to quantity. The vertical (or horizontal) grids for temperature need not relate to those for composition, nor do the grids for all molecules need to be identical.

In addition to geolocation information, quantity templates typically contain additional information to describe the quantity. This includes the molecule for an abundance-related quantity, the signal, radiometer and module for a radiance-related quantity etc. etc.

2.4 Matrices in MLSCFM

Matrices in the CFM software are defined by the vectors that describe their rows and columns. For example, the Jacobian matrix relates a state vector to a measurement vector, while a covariance matrix describes the covariance of a single vector. The matrices are stored in a blocked manner, with subblocks relating one instance of one quantity in the rows vector with an instance of another quantities in the columns vector. The tomographic nature of the MLS retrievals [Livesey et al., 2006] means that many of the blocks are all zero and can be omitted for efficiency of storage and computation. Similarly, many of the individual blocks are themselves sparse, and can be stored as such in the software.

The MLSCFM software contains two main modules for dealing with matrices. `MatrixModule_0` contains the software for storing and manipulating individual blocks, while `MatrixModule_1` contains the software that describes matrices as a collection of `Matrix_0` blocks, and the algorithms for operating upon these '`Matrix_1`' entities.

2.5 Defining coordinate systems

As discussed above, vector quantities describe geophysical, measurement, or other variables on up to three dimensions. These three dimensions are known as `vGrid` (for vertical coordinates), `hGrid` (for horizontal coordinates) and `fGrid` (for frequency / channel descriptions).

These grids can be specified prior to the construction of quantity templates. However, they are not essential to the creation of vector quantities. For example, minor frame quantities take their geolocation information (both horizontal and vertical) directly from the L1B information. In addition, radiance quantities take their frequency/channel information directly from the MLS signals database.

2.5.1 Vertical coordinates – `vGrid`

MLS vector quantities can be described on a variety of vertical coordinates, including pressure (and more commonly $-\log_{10}$[Pressure / hPa] (known as `zeta` in the software) and altitude (in meters). Typically `vGrids` are used only to define coherent vector quantities (see discussion above). Thus far, the only incoherent quantities in MLSL2 are minor frame quantities. These have their vertical coordinates defined by MLS L1B tangent point altitude information (in meters).

2.5.2 Horizontal coordinates – `hGrid`

The definition of horizontal coordinates differs from that of vertical and frequency coordinates, in that the coordinates change from scene to scene, and run to run, depending on the timespan of MLS data under consideration. There are several types of `hGrids` in the L2 software, but for the purposes of the CFM, only 'regular' grids are of interest.

These are created for a given 'chunk' (i.e., range of MAFs from the L1 file), and distribute profiles with a requested geodetic angle spacing along the orbit. The absolute offset of this even spacing can also be specified. Typically this is set to zero degrees meaning that this even spacing would place a profile exactly on the equator (were the MAFs in the supplied chunk to encompass the equator).

2.5.3 Frequency coordinates – `fGrid`

As discussed above, few quantities have more than one channel. The most significant group of multi-channeled quantities is radiances, these take their frequency information directly from MLS signals database. Some state vector quantities are used to describe spectrally flat, or slowly varying, offsets to observed radiance or atmospheric emission/absorption ('baseline' and 'extinction', respectively). If a (typically slow) frequency variation is required for these, then this is described as a linear interpolation between tie points specified in a `fGrid`.

2.6 Sources of information for MLS state vector components

The MLS forward models require a wide variety of vector quantities as input. Typically, two state vectors are supplied, one (`ForwardModelIn`) containing the quantities of interest (i.e., those quantities for which a solution is sought for some or all elements), and the other (`ForwardModelExtra`) containing other (e.g., calibration) quantities, also needed by the forward model. The majority of these lesser quantities will be defined and filled for the user by subroutines supplied as part of the MLSCFM software, and need little discussion here.

However, an important exception is the tangent pressure information, for which there are a variety of approaches. Firstly, the value retrieved as part of the MLS processing can be read from the L2 'DGM' file. This is useful for initial 'O–F' studies, but is not feasible when radiance assimilation is to be performed in real time. In addition the production processing's use of GMAO temperature as an *a priori* introduces potential biases in this approach.

An alternative approach is to estimate the tangent pressure given knowledge of atmospheric temperature and geopotential height, Aura orientation and MLS pointing. However, the pointing knowledge is imperfect and can also introduce biases.

The best approach is likely to solve for tangent pressure for each minor frame as part of any radiance assimilation activity. Whether this proves to be practical is the subject of further investigations. (Aside, one approach may be to take GMAO GPH as 'truth' and retrieve a – hopefully slowly varying – Aura attitude offset term).

2.7 Defining forward models

In addition to defining the state and measurement vectors (and possibly defining Jacobian matrices to connect them), the CFM software also requires individual forward models to be defined. There are several different types of forward models that can be defined. In the context of the CFM, we anticipate that the 'Full', 'Linearized', 'Baseline' and 'Scan2D' models will be of interest.

2.7.1 The ‘full’ forward model

The ‘full’ forward model is the cornerstone of the MLS forward models. This model is described in Read et al. [2006] and Schwartz et al. [2006] and is a full 2-dimensional ray tracing line-by-line forward model. One full forward model call computes radiances for one or more channels within a given radiometer. As the forward model computes for the full spectral range covered by all the channels requested, in cases where radiances from distinct spectral regions (e.g., groups of channels above and below, but not at, the line center) are to be computed, more efficiency is gained if separate full forward models are defined, each covering only adjacent channels. Full forward models may be ‘scalar’ or ‘polarized’, the latter being needed for accurate modeling of the 118 GHz O_2 line in the mesosphere.

In addition to computing radiances, full forward models can also be requested to compute Jacobians with respect some state vector quantities (down to the element by element level). Various approximations (e.g., the degree to which full Gauss-Legendre integration is performed, and the so called ‘pre frequency averaging’ assumption) can be requested, depending on the desired efficiency and accuracy.

2.7.2 The ‘linearized’ forward model

The ‘linearized’ forward model simply computes radiances based on a linearized form of the full forward model, to whit:

$$\mathbf{y} = \mathbf{y}_0 + \mathbf{K}(\mathbf{x} - \mathbf{x}_0) \tag{1}$$

Where $\mathbf{x}$ is the state vector, and $\mathbf{y}$ is a vector of radiances (on a fixed tangent pressure grid). $\mathbf{y}_0$ and $\mathbf{K}_0$ are pre-computed radiances and jacobians, respectively, corresponding to a pre-defined linearization point $\mathbf{x}_0$ in state space. Following this Taylor series expansion, the radiances are interpolated to the actual tangent pressures defined in the supplied state vector. The precomputed radiances and Jacobians for pre-selected states are stored in ‘Level 2 Processing Coefficient’ (L2PC) files, and are generated by off-line runs of the MLSL2 software invoking the full forward models described above. The linearization points are typically defined as a function of latitude (in $\sim$15°-wide bins) and calendar month.

The linearized model produces results much faster than all but the simplest full forward model runs. However, being linear, its accuracy degrades rapidly as atmospheric optical depth increases (i.e., close to line centers and/or lower in the atmosphere).

2.7.3 The ‘baseline’ forward model

Most MLS geophysical products derive from observations of spectral contrast (channel to channel changes in observed radiances). Typically the ‘baseline’ upon which these spectra sit is of less interest (and is typically more sensitive to errors in spectroscopy or instrument calibration). The MLS data processing algorithms can retrieve a ‘baseline’ offset term along with composition and temperature profiles etc. This term is simply a spectrally-flat (or slowly varying with frequency) offset added to each forward model calculation. This can be invoked by adding the `do_baseline` flag to a forward model, or, if desired, a separate ‘baseline’ forward model can be manually invoked to add these terms.

2.7.4 The ‘scan’ forward model

Given knowledge, from the MLS state vector, of the atmospheric temperature profile on pressure surfaces, the geopotential height at one pressure surface, and MLS tangent point pressures for each minor frame, it is possible, through hydrostatic balance (and refraction calculations, for which a water vapor profile is also needed for accuracy), to compute the expected tangent point altitudes for the MLS radiance observations. Comparing these to the tangent point altitudes derived from the MLS antenna position encoder and the

Table 1: State vector quantities needed for different forward model types. Cells with '?' indicate that the quantity may be required for some configurations.

Quantity	Full	Linearized	Scan
Temperature	Y	Y	Y
Reference GPH	Y	Y	Y
Tangent pressure	Y	Y	Y
Composition	Y	Y	Y
Space radiance	Y		
Earth reflectivity	Y		
Spacecraft altitude	Y		
Orbital inclination	Y		
Line of sight velocity	Y		
Spacecraft altitude	Y		
Field of view offsets	Y		
Sideband fractions	Y	?	
Magnetic field	?		

Aura attitude control system provides an additional constraint on the atmospheric temperature profiles, and enables retrieval of geopotential height (subject to inaccuracies in MLS/Aura pointing).

The 'scan residual' model invokes these calculations as described in Read et al. [2006]. For a variety of reasons, this model is formulated to compute the 'scan residual', being the difference between the tangent point altitudes inferred from hydrostatic balance and those inferred from Aura/MLS pointing. The 'measurement' of this quantity is assumed to be zero with a precision reflective of the precision of the MLS scan encoder ($\sim$30 m).

2.8 Invoking forward models

Once forward models are defined, and state and measurement vectors are defined and filled, forward models can be invoked. As described above, two state vectors can be supplied – `ForwardModelIn` defining the quantities for which solutions are sought, and `ForwardModelExtra` defining other needed quantities. From the forward model's perspective, the only difference between these two is that Jacobians, if requested, are only computed for quantities in the first vector.

The forward models, particularly the full forward model, require a diverse range of quantities to be supplied through the state vector(s). These are detailed in Table 1.

2.9 A note on the 'mockup'

The mockup code supplied at the end of this document is simply meant to serve as an example of the kind of code that can use the forward model. It is *not* intended to be turned into some kind of subroutine to be called by external software. To do so would necessitate rereading all the calibration files, recreating the vector templates etc. This is largely unnecessary work (though we note that the vector templates do need to be recreated each time a new 'scene' is considered). Once the calibration files are read and vectors defined and filled, forward models can be called as many times, with as many different configurations as are desired.

3 Reference

The ICFM software is made of two parts: the server, which is an executable, and the client, which is an IDL library. The two communicate with each other via PVM. The users of ICFM have to start PVM on the machine where the server and the client run, then start the server, then use procedures provided in the IDL library to invoke the forward model.

How to run the server will be shown in Appendix A. This section will focus on the IDL library pertaining to ICFM. Although the example will involve the use of a few AtGod procedures, it is only an illustration of ICFM's compatibility to AtGod library, and those AtGod procedures won't be discuss here.

3.1 Data types

3.1.1 `Vector`

Vector is an anonymous data type that contains a list of quantities. To access a quantity inside the vector, the users use the dot notation: `[vector].[quantity name]`. Vector should be created and manipulated via the procedures provided in Section 3.2 only.

3.1.2 `Quantity`

This is another anonymous data type. It lays out a collection of profiles ('instances') for a given quantity (temperature, composition, tangent pressure, radiance etc.). It is similar to CFM's QuantityTemplate_T type. For more information, please refer to Section 2.3 in Nguyen et al. [2010]. However, unlike QuantityTemplate_T, Quantity can contains both the quantity's value and its mask. Not every Quantity object is the same, because there are fields that some quantities have and others don't. Consequently, users should only create quantities via CreateQuantity procedure described below.

3.2 Procedures and Functions

This subsection describes the procedures/functions the users will need to invoke in order to create the input data for and to invoke the forward model. They are presented in alphabetical order. The example given in Appendix A illustrates their invocation and logical flow.

3.2.1 `AddQuantity2Vector`

```
pro AddQuantity2Vector, vector, qty, overwrite=overwrite
    ; vector: the vector to which the quantity is added
    ; qty: the quantity to add, the quantity's name must not be 'name'
    ; overwrite (optional): if false, then the procedure will generate
    ; an error if there is already a quantity of the same name
    ; as qty's in vector. If true, then the procedure will replace
    ; the quantity in vector with qty.
end
```

3.2.2 `Core2Qty`

This procedure is to help convert AtGod relaxed data type into the stricter Quantity data type.

```
pro core2qty, core, qty, name, type, signal=signal, molecule=molecule, module=module
    ; core: AtGod quantity, usually created by AtGod_CreateCore function
    ; qty: output, ICFM Quantity data type that is equivalent to core
    ; name: unlike core, qty needs a name
    ; type: unlike core, qty also needs a type. Please refer to Appendix
    ; for applicable types.
    ; signal (optional): some quantities have an associated signal. Please
    ; refer to Appendix for applicable signals.
    ; molecule (optional): some quantities have an associated molecule.
    ; Please refer to Appendix for applicable molecules.
    ; module (optional): some quantities are associated with a module.
    ; Module is one of "GHz", "THz", and "sc".
end
```

3.2.3 `CreateQuantity`

```
pro CreateQuantity, quantity, name, type, instanceOffset, value=value, $
logbasis=logbasis, startValue=startValue, badValue=badValue, $
molecule=molecule, module=instrumentModule, signal=signal, $
radiometer=radiometer, frequencies=frequencies, $
frequencyCoordinate=frequencyCoordinate, noChans=noChans, surfs=surfaces, $
nosurfs=noSurfs, verticalCoordinate=verticalCoordinate, noprofs=noprofs, $
phi=phi, geodlat=geodlat, longitude=lon, losAngle=losAngle, $
solarZenith=solarZenith, solarTime=solarTime, time=time, mask=mask, $
coherence=coherence
    ; quantity: output, the quantity to be returned
    ; name: name of the quantity
    ; type: a string. Please refer to Appendix for applicable types.
    ; instanceOffset: index of the first non-overlapped instance
    ; value (optional): the value for this quantity
    ; logbasis (optional): whether the value is on a log scale. Default is false.
    ; startValue (optional): only valid when logbasis is true.
    ; badValue (optional): the constant representing an invalid value point
    ; in the value array
    ; molecule (optional): the molecule associated with this quantity if any.
    ; Please refer to Appendix for a list of applicable molecules.
    ; module (optional): the instrument module associated to this quantity
    ; if any. Module is a string, one of "GHz", "THz", "sc".
    ; signal (optional): the signal associated with this quantity if any.
    ; Please refer to Appendix for a list of applicable signals.
    ; radiometer (optional): if signal is not needed, there can still a
    ; a radiometer associated with this quantity. Please refer to the
    ; signal list in the Appendix for applicable radiometers.
    ; frequencies (optional):
    ; frequencyCoordinate (optional): valid only when frequencies are present
    ; noChans (optional): needed only when frequencies are not present. When
    ; frequencies are present, noChans is set to the size of frequencies.
    ; Otherwise, this value is 1.
    ; surfs (optional): associated pressure surfaces
    ; nosurfs (optional): number of pressure surfaces
    ; verticalCoordinate (optional): only valid when surfs is present.
    ; Please refer to Appendix for a list of valid vertical coordinate.
    ; noProfs (optional): a.k.a. noInstances
    ; phi (optional)
    ; geodlat (optional):
    ; lon (optional):
    ; losAngle (optional):
    ; solarZenith (optional):
    ; solarTime (optional):
    ; time (optional):
```

```
    ; coherence (optional): default is true
end
```

3.2.4 CreateVector

```
pro CreateVector, vector, name
    ; vector: output
    ; name: name of the vector
end
```

3.2.5 ForwardModel

```
pro ForwardModel, tid, info, mafNo, state, stateExtra, measurement, output
    ; tid: server’s tid
    ; info: output, status message of the call to foward model
    ; mafNo: the maf number of the data to compute
    ; state: input vector
    ; stateExtra: input vector
    ; measurement: input vector, but no values needed for quantities of this
    ; output: variable to hold output data. Output will have as many
    ; quantities as measurement. However, those quantities are AtGod core
    ; data type.
end
```

3.2.6 ICFM_Cleanup

```
pro ICFM_Cleanup, tid, info
    ; This procedures cleanup data on the server, after the user
    ; finishes with the forward model. This procedure only need
    ; to be call once while forwardmodel procedure can be called
    ; multiple times.
    ; tid: server’s tid
    ; info: output, status of the call
end
```

3.2.7 ICFM_SendQuantity

```
pro ICFM_SendQuantity, qty, tid=tid, info=info, msgtag=msgtag, packonly=packonly
    ; qty: quantity to send
    ; tid (optional): server’s tid, only needed if packonly is not set
    ; info (optional): output status info
    ; msgtag(optional): should be 200L, but only needed if packonly is not set
    ; packonly (optional): if set, this procedure will only pack the quantity
    ; into the send buffer without sending it
end
```

3.2.8 `ICFM_SendVector`

```
pro icfm_sendvector, vector, tid=tid, info=info, msgtag=msgtag, packonly=packonly
    ; vector: vector to send or pack
    ; tid (optional): server's tid, needed only if packonly is not set
    ; info (optional): output, status report
    ; msgtag (optional): should be 200L, only needed if packonly is not set
    ; packonly (optional): if set, this procedure will only pack the vector
    ; into the send buffer without sending it
end
```

3.2.9 `ICFM_Setup`

```
pro ICFM_Setup, tid, info, signalfile, configfile, spectroscopy, $
antennaPatterns, filterShapes, dacsFilterShapes, pointingGrids, $
pfa, hdf5l2pc
    ; This procedures allocate common databases of different forward
    ; model calls on the server.
    ; tid: server's tid
    ; info: output, status report info
    ; signalfile: absolute file name of the signal file
    ; configfile: absolute file name
    ; spectroscopy: absolute file name of spectroscopy file
    ; antennapatterns: absolute file name of the antenna file
    ; filterShapes: absolute file name
    ; dacsFilterShapes: absolute file name
    ; pointingGrids: absolute file name
    ; pfa: an array of absolute file names
    ; hdf5l2pc: an array of absolute file names
end
```

A Examples

A.1 Starting the server

```
/path/to/executable/server
```

A.2 Invoking a forward model

The following code gives an example of a simple client that invokes the MLS forward model.

```
l2aux = '/data/emls/l2aux/v03.30/2009/040/MLS-Aura_L2AUX-DGM_v03-30-c01_2009d040.h5'
l1boa = '/data/emls/l1b/v03.30/2009/040/MLS-Aura_L1BOA_v03-30-c01_2009d040.h5'
l1brad = '/data/emls/l1b/v03.30/2009/040/MLS-Aura_L1BRADG_v03-30-c01_2009d040.h5'
filenames = [l1boa, l1brad, l2aux]
maf = 214
band = 9
ptan = reademlsmifq(filenames=filenames, 'GHz.ptan-Core', vertical='A', nomafs=1, $
firstmaf=maf)
createvector, stateExtra, 'stateExtra'
core2qty, ptan, temp, 'ptan', "ptan", module="GHz"
addquantity2vector, stateExtra, temp
geodalt = reademlsmifq(filenames=filenames, 'GHz/GeodAlt', firstmaf=maf, nomafs=1, $
vertical='N')
```

```
core2qty, geodalt, temp, 'geodaltitude', 'tngtgeodalt', module='GHz'
addquantity2vector, stateExtra, temp
losvelghz = reademlsmifq(filenames=filenames, 'GHz/LosVel', firstmaf=maf, nomafs=1, $
vertical='N')
core2qty, losvelghz, temp, 'losVelGHz', 'LOSVel', module='GHz'
addquantity2vector, stateExtra, temp
orbincl = reademlsmifq(filenames=filenames, 'sc/OrbIncl', firstmaf=maf, nomafs=1, $
vertical='N', module='sc', /limitedgeolocation)
core2qty, orbincl, temp, 'orbincl', 'orbitinclination', module='sc'
addquantity2vector, stateextra, temp
phitan = ptan
phitan.val = ptan.geodangle
core2qty, phitan, temp, 'phitan', 'phitan', module='GHz'
addquantity2vector, stateextra, temp
spaceRadiance = AtGod_CreateCore ( vertical='N', noProfs=1 )
spaceRadiance.val = 2.735
core2qty, spaceradiance, temp, 'spacerad', 'spaceradiance'
addquantity2vector, stateextra, temp
earthReflectivity = AtGod_CreateCore ( vertical='N', noProfs=1 )
earthReflectivity.val = 0.05
core2qty, earthreflectivity, temp, 'earthrefl', 'earthRefl'
addquantity2vector, stateextra, temp
scgeocalt = reademlsmifq (filenames=filenames, 'sc/GeocAlt', vertical='N', $
firstmaf=maf, nomafs=1, module='sc', /limitedgeolocation)
core2qty, scgeocalt, temp, 'scGeocAlt', 'scgeocalt', module='sc'
addquantity2vector, stateextra, temp
ghzgeocalt = reademlsmifq (filenames=filenames, 'GHz/GeocAlt', vertical='N', $
firstmaf=maf, nomafs=1, module='GHz')
core2qty, ghzgeocalt, temp, 'geocAlt', 'tngtgeocalt', module='GHz'
addquantity2vector, stateextra, temp
sbfFile = '/data/emls/l2cal/MLS-Aura_L2Cal-SBFraction_v3-0-2_0000d000.l2cf'
band='B9F'
band9 = reademlsmifq(filenames=filenames, signal=band, vertical='A', nomafs=1, $
firstmaf=maf)
createvector, measurement, 'measurement'
core2qty, band9, temp, 'band9', 'radiance', signal='R3:240.B9F:CO'
addquantity2vector, measurement, temp
sbf9l = atgod_createcore(vertical='N', noProfs=1, auxinfo=band9.auxinfo)
sbf9l.val = readl2cfarray(sbffile, $
prefix='!define(limbradsbfraction9L,{', suffix='})!dnl', extrasuffix=',/spread')
core2qty, sbf9l, temp, 'lsf9l', 'limbsidebandfraction', $
signal='R3:240.B9LF:CO.S0.FB25-9'
addquantity2vector, stateextra, temp
sbf9u = atgod_createcore(vertical='N', noProfs=1, auxinfo=band9.auxinfo)
sbf9u.val = readl2cfarray(sbffile, $
prefix='!define(limbradsbfraction9U,{', suffix='})!dnl', extrasuffix=',/spread')
core2qty, sbf9u, temp, 'lsf9u', 'limbsidebandfraction', signal='R3:240.B9UF:CO'
addquantity2vector, stateextra, temp
elevoffset9l = atgod_createcore(vertical='N', noProfs=1, auxinfo=band9.auxinfo)
elevoffset9l.val = 0
core2qty, elevoffset9l, temp, 'eo9l', 'elevoffset', signal='R3:240.B9LF:CO'
addquantity2vector, stateextra, temp
elevoffset9u = atgod_createcore(vertical='N', noprofs=1, auxinfo=band9.auxinfo)
elevoffset9u.val = 0
core2qty, elevoffset9u, temp, 'eo9u', 'elevoffset', signal='R3:240.B9UF:CO'
addquantity2vector, stateextra, temp
date = '2009d040'
version = 'v03.30'
CO = ReadEMLSDaySeries ( date, date, version=version, order='cycle desc', $
    product='CO', /quiet )
;; Select only the first 'noProfs' profiles
noProfs = maf + 6
CO = ExtractProfiles ( CO,  noProfs)
core2qty, co, temp, 'CO', 'vmr', molecule='CO'
createvector, state, 'state'
addquantity2vector, state, temp
temperature = reademlsdayseries(date, date, version=version, order='cycle desc', $
product='Temperature', /quiet)
temperature = extractprofiles(temperature, noprofs)
core2qty, temperature, temp, 'Temperature', 'temperature'
addquantity2vector, stateextra, temp
gph = reademlsdayseries(date, date, version=version, order='cycle desc', $
product='GPH', /quiet)
gph = extractprofiles(gph, noprofs)
core2qty, gph, temp, 'GPH', 'gph'
addquantity2vector, stateextra, temp
refgph = atgod_createcore(vertical='P', nosurfs=1, noprofs=temperature.noprofs)
refgph.surfs = 100.0D
refgph.val = 16000.0D
refgph.lat = temperature.lat
refgph.lon = temperature.lon
refgph.sza = temperature.sza
refgph.day = temperature.day
refgph.time = temperature.time
refgph.lst = temperature.lst
refgph = create_struct(['losangle', 'geodangle'], temperature.losangle, $
temperature.geodangle, refgph)
core2qty, refgph, temp, 'refGPH', 'refGPH'
addquantity2vector, stateextra, temp
common pvmlib, config
if n_elements(config) eq 0 then Initpvmlib
msgtag=200L
signalFile='/users/honghanh/mlspgs/cfm/signals.l2cf'
configFile='/users/honghanh/mlspgs/idlcfm/config.l2cf'
mafNo = 0L
spectroscopy='/data/emls/l2cal/MLS-Aura_L2Cal-Spectroscopy-PFA_v3-0-4_0000d000.h5'
antennaPatterns='/data/emls/l2cal/MLS-Aura_L2Cal-AAAP_v2-0-0_0000d000.txt'
filterShapes='/data/emls/l2cal/MLS-Aura_L2Cal-Filters_v3-0-2_0000d000.txt'
DACSFilterShapes='/data/emls/l2cal/MLS-Aura_L2Cal-DACSFilters_v1-5-1_0000d000.txt'
pointingGrids='/data/emls/l2cal/MLS-Aura_L2Cal-PFG_v3-0-0_0000d000.txt'
pfa=['/data/emls/l2cal/PFA_R5V_FS-04.h5', $
'/data/emls/l2cal/PFA_R5H_FS-04.h5', $
'/data/emls/l2cal/PFA_R4_FS-04.h5', $
'/data/emls/l2cal/PFA_R3_FS-04.h5', $
'/data/emls/l2cal/PFA_R2_FS-04.h5', $
'/data/emls/l2cal/PFA_R1B_FS-04.h5', $
'/data/emls/l2cal/PFA_R1A_FS-04.h5', $
'/data/emls/l2cal/PFA_DACS_FS-04.h5']
hdf5l2pc= $
['/data/emls/l2cal/l2pc_30H6/MLS-Aura_L2Cal-L2PC-band9-LATSCALARHIRESO3HR_v3-00-HO-06_m02.h5']

tid = 1 ; replace this with the tid that server prints when it starts
icfm_setup, tid,info, signalfile, configfile, spectroscopy, $
antennaPatterns, filterShapes, dacsFilterShapes, pointingGrids, $
pfa, hdf5l2pc
forwardmodel, tid, info, mafNo, state, stateExtra, measurement, $
output
icfm_cleanup, tid, info
```

```
end
```

B Lists and tables

B.1 Types

```
cloud                          IceDensity
cloudExtinction                iwc
cloudIce                       IWCfromExtinction
cloudInducedRadiance           iwc_high_height
cloudTemperature               iwc_low_height
cloudWater                     iwp
cloudy_110RH_below_top         ptan
cloudy_110RH_in_cloud          radiance
elevOffset                     refGPH
fieldAzimuth                   rhi
fieldElevation                 RHIFromH2O
fieldStrength                  RHIPrecisionFromH2O
geocAltitude                   systemTemperature
geodAltitude                   temperature
geodAngle                      temperature_precision
gph                            tngtgeocalt
gph_precision                  tngtgeodalt
H2OFromRHI                     vmr
H2OPrecisionFromRHI
```

B.2 Units

Quantity	Units
`invalid`	Default value for uninitialized variables
`dimensionless`	Value for unitless quantities
`length`	Meters
`time`	Seconds
`pressure`	Millibars
`temperature`	Kelvins
`vmr`	Number of ‘parts per one’
`angle`	Degrees
`mafs`	Number of mafs
`mifs`	Number of mifs
`frequency`	MHz
`zeta`	$\log_{10}$ (Pressure / hPa)
`velocity`	meters / second
`extinction`	1 / meters
`icedensity`	g / meters^3
`colmabundance`	$\log_{10}$ (g /meters)^3
`pctrhi`	Relative humidity wrt. ice (%)
`gauss`	Gauss
`profiles`	Number of profiles

B.3 Molecules

```
BR_79_O              CH3COCH3             CO
BR_79_O_V1           CH3D                 CO_17
BR_81_O              CH3O                 CO_18
BR_81_O_V1           CH3OH                CO2
BRO                  CH3OOH               COF2
C_13_H2CHCN          CH4                  D2O
C_13_H3CN            CL_35_NO2            DCL_35
C_13_O               CL_35_O              DCL_37
CH2C_13_HCN          CL_35_ONO2           DF
CH2CDCN              CL_35_OOCL_35        DNO3
CH2CHC_13_N          CL_35_OOCL_37        EXTINCTION
CH2CHCN              CL_35_O_V1           EXTINCTIONV2
CH2OHCOCH2OH         CL_37_NO2            H2C_13_O
CH3C_13_N            CL_37_O              H2CO
CH3CH2CH3            CL_37_ONO2           H2CO_18
CH3CH2CN             CL_37_O_V1           H2O
CH3CH2OH             CLNO2                H2O_17
CH3CL                CLO                  H2O_17_V2
CH3CL_35             CLONO2               H2O_18
CH3CL_37             CLOOCL               H2O_18_V2
CH3CN                CLOUD_A              H2O2
CH3CN_15             CLOUD_S              H2O_V2
```

H2S	HOBR_81	O3
H2S_32	HOCL	O3_2V2
H2S_32_O4	HOCL_35	O3_ASYM_O_17
H2SO4	HOCL_37	O3_ASYM_O_18
HBR	HONO	O3_ASYM_O_18_V2
HBR_79	HONO_18_O	O3_R
HBR_81	HONOO_18	O3_SYM_O_17
HC_13_N	HOONO	O3_SYM_O_18
HCL	HOONO2	O3_SYM_O_18_V2
HCL_35	N_15_H3	O3_V1_3
HCL_35_O	N_15_NO	O3_V1_3_V2
HCL_37	N2	O3_V2
HCL_37_O	N2O	OBR_79_O
HCLO	N2O_17	OBR_81_O
HCN	N2O_18	OBRO
HCN_15	N2O_2V2	OC_13_S_32
HCN_V2	N2O_V1	OCL_35_O
HCOCH2OH	N2O_V2	OCL_37_O
HCOOH	NF3	OCLO
HDO	NH2CH2CH2OH	OCO_18
HDO_17	NH3	OCS
HDO_18	NH3_V2	OCS_32
HDO_V2	NN_15_O	OCS_34
HF	NO	OD
HN_15_O3	NO2	OH
HNO3	NO_PLUS	OH_V1_2
HNO3_V5	O	S_32_O_18_O
HNO3_V6	O_17	S_32_O2
HNO3_V7	O_17_O	S_32_O2_V2
HNO3_V8	O_18	S_33_O2
HNO3_V9	O_18_H	S_34_O2
HO_18_NO2	O_18_O	SNGLT_DLTA_O_18_O
HO2	O_18_O_V1	SNGLT_DLTA_O2
HOBR	O2	SO2
HOBR_79	O2_V1	

B.4 Bands

R1A:118.B1LF:PT	R4:640.B10LF:ClO	R5V:2T5.B19LF:OH
R2:190.B2LF:H2O	R4:640.B11LF:BrO	R5V:2T5.B20LF:PT
R2:190.B3LF:N2O	R4:640.B12LF:N2O	R1B:118.B21LF:PT
R2:190.B4LF:HNO3	R4:640.B13LF:HCl	R1A:118.B22LD:PT
R2:190.B5LF:ClO	R4:640.B14LF:O3	R2:190.B23LD:H2O
R2:190.B6LF:O3	R5H:2T5.B15LF:OH	R3:240.B24LD:O3
R3:240.B7LF:O3	R5H:2T5.B16LF:OH	R3:240.B25LD:CO
R3:240.B8LF:PT	R5H:2T5.B17LF:PT	R1B:118.B26LD:PT
R3:240.B9LF:CO	R5V:2T5.B18LF:OH	R2:190.B27LM:HCN

```
R4:640.B28LM:HO2        R3:240.B7UF:O3          R5V:2T5.B19UF:OH
R4:640.B29LM:HOCl       R3:240.B8UF:PT          R5V:2T5.B20UF:PT
R4:640.B30LM:HO2        R3:240.B9UF:CO          R2:190.B23UD:H2O
R4:640.B31LM:BrO        R4:640.B10UF:ClO        R3:240.B24UD:O3
R1A:118.B32LW:PT        R4:640.B11UF:BrO        R3:240.B25UD:CO
R3:240.B33LW:O3         R4:640.B12UF:N2O        R2:190.B27UM:HCN
R1B:118.B34LW:PT        R4:640.B13UF:HCl        R4:640.B28UM:HO2
R2:190.B2UF:H2O         R4:640.B14UF:O3         R4:640.B29UM:HOCl
R2:190.B3UF:N2O         R5H:2T5.B15UF:OH        R4:640.B30UM:HO2
R2:190.B4UF:HNO3        R5H:2T5.B16UF:OH        R4:640.B31UM:BrO
R2:190.B5UF:ClO         R5H:2T5.B17UF:PT        R3:240.B33UW:O3
R2:190.B6UF:O3          R5V:2T5.B18UF:OH
```

References

N. J. Livesey, W. V. Snyder, W. G. Read, and P. A. Wagner. Retrieval algorithms for the eos microwave limb sounder (mls) instrument. *IEEE Trans. Geosci. Remote Sens.*, 44(5):1144–1155, 2006.

H. H. Nguyen, P. A. Wagner, and N. J. Livesey. Aura microwave limb sounder (mls) callable forward model interface requirements document. *Jet Propulsion Laboratory*, 2010.

W. G. Read, Z. Shippony, M. J. Schwartz, N. J. Livesey, and W. V. Snyder. The clear-sky unpolarized forward model for the EOS Microwave Limb Sounder (MLS). *IEEE Trans. Geosci. Remote Sens.*, 44(5): 1367–1379, 2006.

M. J. Schwartz, W. G. Read, and W. V. Snyder. Polarized radiative transfer for Zeeman-split oxygen lines in the EOS MLS forward model. *IEEE Trans. Geosci. Remote Sens.*, 44(5):1182–1190, 2006.

J. W. Waters, L. Froidevaux, R. S. Harwood, R. F. Jarnot, H. M. Pickett, W. G. Read, P. H. Siegel, R. E. Cofield, M. J. Filipiak, D. A. Flower, J. R. Holden, G. K. Lau, N. J. Livesey, G. L. Manney, H. C. Pumphrey, M. L. Santee, D. L. Wu, D. T. Cuddy, R. R. Lay, M. S. Loo, V. S. Perun., M. J. Schwartz, P. C. Stek, R. P. Thurstans, K. M. Chandra, M. C. Chavez, G. Chen, M.. A. Boyles, B. V. Chudasama, R. Dodge, R. A. Fuller, M. A. Girard, J. H. Jiang, Y. Jiang, B. W. Knosp, R. C. LaBelle, J. C. Lam, K.. A. Lee, D. Miller, J. E. Oswald, N. C. Patel, D. M. Pukala, O. Quintero, D. M. Scaff, W. V. Snyder, M. C. Tope, P. A. Wagner, and M. J. Walch. The Earth Observing System Microwave Limb Sounder (EOS MLS) on the Aura satellite. *IEEE Trans. Geosci. Remote Sens.*, 44(5):1075–1092, 2006.