IRAM Memo 2014-1

CLASS Data Fillers

S. Bardeau[1], J. Pety[1,2]

1. IRAM (Grenoble)
2. LERMA, Observatoire de Paris

December, 2^{nd} 2016
Version 1.2

Abstract

CLASS is the program of the **GILDAS** suite dedicated to the data reduction and analysis of heterodyne instruments at cm and (sub-)mm single-dish telescopes. It is used by the IRAM-30m, APEX, SOFIA/GREAT, Herschel/HIFI, YEBES, Effelsberg. Because of tremendous improvement of the heterodyne spectrometer, the **CLASS** data format is evolving again after years of stability. To help non-IRAM telescope to cope with these changes, the **CLASS** team is now providing fillers to the Class Data Format writers. Three different possibilities are given: 1) A procedure using the **SIC** intepreter; 2) A Fortran program linked to the **CLASS** library; And 3) a Python script using a dedicated Python module. This document describes how those 3 APIs can be used and examples are provided in the appendices.

Contents

1 The Sic API

The historical way (and may be easiest for a **CLASS** user) to produce spectra is to do it directly in the **CLASS** program itself. The appendix A.1 reproduces a demonstration procedure provided with **CLASS**. It can be called at the **CLASS** prompt by typing:

```
LAS90> @ gag_demo:demo-telwrite.class
```

This procedure performs 4 main tasks:

1. it opens a new output **CLASS** file. The `SINGLE` mode is intended to be used when writing independent observations (*e.g.* at the telescope),
2. it disables all sections in the R buffer header, except the ones written in this example. The sections General, Position and Spectroscopy (resp. Drift) are the 3 minimum to be filled when writing a spectroscopic observation (resp. a continuum drift).
3. if fills the data and desired sections. The command `MODEL` is used to transfer the channel intensities to the R buffer, and the command `LET` is used to set up the sections.
4. finally it writes the content of the R buffer from the memory to the output file thanks to the command `WRITE`.

A loop is also performed in order to write a hundred of observations to the file. Useful help can be found in the the online HELP of the following commands:

- `FILE`: open a **CLASS** file for writing.
- `SET VARIABLE`: turn the requested section read-write. See `HELP SET VARIABLE` for a detailed description of each section.
- `MODEL`: store a spectrum in the R buffer,
- `WRITE`: write the R spectrum to the output file.

2 The Fortran API

Using the **CLASS** program is convenient in a development phase for testing, or for small projects, but in a production phase it may be better to use a program linked to the **CLASS** library with the dedicated Fortran API. The main advantages of this API is 1) that it does not use the **SIC** interpreted language (compiled code is faster) and 2) that it does not start the whole **CLASS** program mechanisms (*e.g.* command logging, graphic support, etc).

A demonstration program is presented in the Appendix B.1. This program does the same as the `demo-telwrite.class` procedure. Note that all subroutines using the Fortran API *must* use the module `class_api`. It provides **CLASS** types, parameters, and procedure interfaces. Here is a summary of the subroutines provided by the API:

- `class_write_init`: initialize the libclass for writing process,
- `class_write_clean`: clean the libclass after writing process,
- `class_fileout_open`: initialize/reopen a Class output file,
- `class_fileout_close`: close the Class output file,
- `class_obs_init`: initialize an `observation` variable,
- `class_obs_reset`: zero-ify and resize an observation,

- `class_obs_write`: write an observation to the output file,
- `class_obs_clean`: clean/free an `observation` variable.

The detailed API is available in the appendix B.2. It is the responsiblity of the calling program to instantiate a `type(observation)` variable and to pass them to the related subroutines. There is no global buffers or variables involved here, on purpose.

2.1 Compiling your program

Your Fortran program must be compiled and linked to the **CLASS** library in the **GILDAS** *compilation* environment, so that it can find the modules and entry points described above. Then it should be run in the **GILDAS** *execution* environment. Two options are available:

1. you have write access to the Gildas compilation and installation directories, and it is acceptable that your program is installed together with the other Gildas programs[1]. If yes, you have to load the usual compilation environment:

   ```
   $ cd gildas/gildas-src-XXX/
   $ source admin/gildas-env.sh
   ```

   Then go where your program can be found and copy the Makefile from the Class main directory:

   ```
   $ cd /path/to/program/
   $ cp $gagsrcdir/packages/class/main/Makefile .
   ```

   Edit this Makefile and replace the targets by your own program on the line `EXECUTABLES =` . Then type

   ```
   $ make
   $ make install
   ```

   Your program is now visible and executable like any other programs from the execution environment.

2. you have no write access to the Gildas compilation and installation directories, or you don't want your program to be visible by other users. If yes, you have to load usual compilation environment with the special option "-u" (use directory):

   ```
   $ cd gildas/gildas-src-XXX/
   $ source admin/gildas-env.sh -u $PWD/integ
   ```

   Then go where your program can be found and copy the Makefile from the Class main directory:

   ```
   $ cd /path/to/program/
   $ cp $gagsrcdir/packages/class/main/Makefile .
   ```

   Edit this Makefile and replace the targets by your own program on the line `EXECUTABLES =` . Then type

   ```
   $ export gagintdir=/path/to/program/exe  ! Define here a custom installation directory
   $ make
   ```

   The environment variable `$gagintdir` has be set to the desired installation directory (create the directory if needed). The Makefile creates a local installation tree under this target. Finally, don't forget to put the path to the executable in your `$PATH`:

   ```
   $ export PATH=/path/to/program/exe/$GAG_EXEC_SYSTEM/bin:$PATH
   ```

[1] If this is a global installation for several users, this means all users will be able to run your program.

3 The Python API

If Python is installed on the system the **CLASS** library is compiled on, the module `pyclassfiller` will be created and installed in the **GILDAS** `$PYTHONPATH`[2]; it is available when using the **GILDAS** execution environment. This module provides a Python overlay to the Fortran API, including access to the appropriate **CLASS** parameters and subroutines. A Python script equivalent to the programs described in the previous sections can be found in the appendix C.1.

3.1 File support

Opening and closing an output file is done thanks to the `ClassFileOut` class, e.g.

```
fileout = pyclassfiller.ClassFileOut()
fileout.open(file='foo.30m',new=True,over=True,size=10000,single=True)
fileout.close()
del fileout
```

The above example creates a new and empty Class file named *foo.30m*. The `open()` and `close()` methods are described in details in Appendix C.2. As in the **CLASS** program (`LAS\FILE OUT`), there can be only 1 output file opened at the same time.

3.2 Observation support

In order to fill the output file, observation have to be defined and written. This is done with the `ClassObservation` class:

```
obs = pyclassfiller.ClassObservation()
```

The sections present should first be declared, *e.g.*:

```
obs.head.presec[:]            = False  # Disable all sections except...
obs.head.presec[code.sec.gen] = True   # General
```

CLASS will accept to write observations which provide *at least* the General, Position, and Spectroscopy or Continuum Drift sections.

Each section has then to be filled, e.g. the Spectroscopic section:

```
obs.head.spe.line = "MYLINE"
obs.head.spe.restf = 123456.
obs.head.spe.nchan = nchan
obs.head.spe.rchan = 1.
obs.head.spe.fres = 1.
obs.head.spe.vres = -1.
obs.head.spe.voff = 0.
obs.head.spe.bad = -1000.
obs.head.spe.image = 98765.
obs.head.spe.vtype = code.velo.obs
obs.head.spe.vconv = code.conv.rad
obs.head.spe.doppler = 0.
```

All elements should be filled, Class does not support partially filled sections. At the time this document is written, only the sections General, Position, Spectroscopy, Frequency Switch, Calibration, and continuum Drift are supported. The content of these sections, as seen from Python, is strictly equivalent to the Sic variables in `R%HEAD%` (*i.e.* see `HELP SET VARIABLE` in the **CLASS** program for the sections details).

Some elements in the Class sections are arrays. They must be defined as `numpy` arrays, e.g.:

[2]Although there are many similarities, this module is not linked to the Gildas-Python binding.

```
import numpy
obs.head.cal.count = numpy.array(range(3),dtype=numpy.float32)
```

The `numpy` array must be defined with the correct size and type: for efficiency purpose (which is often needed when writing **CLASS** files *e.g.* at the telescope), the module `pyclassfiller` does not provide implicit conversion to the appropriate type. You must foresee it when writing your scripts.

Also, some other elements are internal codes, *e.g.*

```
from pyclassfiller import code
obs.head.gen.kind = code.kind.spec
```

Again, these codes are equivalent to the same codes in the **SIC** structure `SIC%CODE%`. They should be used when relevant according to the `SET VARIABLE` help.

The data array (most likely the spectrum intensities) should be also provided as a single precision numpy array, e.g.:

```
obs.datay = numpy.array(range(nchan),dtype=numpy.float32)
```

Finally, once the sections and data are filled, the observation has to be written to the output file:

```
obs.write()
```

If `obs.head.gen.num` is 0, **CLASS** will automatically give the next available (unused) number in the file.

Once you are done with a `ClassObservation` object, you should delete it to release its memory.

4 Acknowledgement

The authors thank S. Guilloteau as a major contributor of the **CLASS** program (including the **SIC** API).

A The Sic API

A.1 Demonstration procedure

```
! Open a new CLASS file
file out classdemo.30m single /overwrite

! Create several CLASS observations
define integer mynchan
let mynchan 128
define real data[mynchan]

set variable sections write  ! Allow to change the section status
let r%head%presec no  ! Disable all sections except...
set variable general write
set variable position write
set variable spectro write

for ispec 1 to 100
  ! Data
  let data[ichan] sin(2.*pi*ichan/mynchan)**2
  model data /xaxis 1.0 0.0 -1.0 V

  ! General
  let r%head%gen%num 0  ! Automatic numbering at WRITE time
  let r%head%gen%ver 0  ! Automatically increased at WRITE time
  let r%head%gen%teles "MYTELES"
  let r%head%gen%dobs 0
  let r%head%gen%dred 0
  let r%head%gen%kind sic%code%kind%spec
  let r%head%gen%qual 0
  let r%head%gen%scan 1
  let r%head%gen%subscan 1
  let r%head%gen%ut 0.d0
  let r%head%gen%st 0.d0
  let r%head%gen%az 0.
  let r%head%gen%el 0.
  let r%head%gen%tau 0.
  let r%head%gen%tsys 100.
  let r%head%gen%time 100.
  let r%head%gen%parang 0.d0
  let r%head%gen%xunit 0

  ! Position
  let r%head%pos%sourc "MYSOURCE"
  let r%head%pos%system sic%code%coord%equ
  let r%head%pos%equinox 2000.0
  let r%head%pos%proj  sic%code%proj%none
  let r%head%pos%lam   pi/2.d0
  let r%head%pos%bet   pi/2.d0
  let r%head%pos%projang 0.d0
  let r%head%pos%lamof 0.
```

```
  let r%head%pos%betof 0.

  ! Spectro
  let r%head%spe%line "MYLINE"
  let r%head%spe%restf 123456.d0
  let r%head%spe%nchan mynchan
  let r%head%spe%rchan 1.
  let r%head%spe%fres 1.
  let r%head%spe%vres -1.
  let r%head%spe%voff 0.
  let r%head%spe%bad -1000.
  let r%head%spe%image 98765.d0
  let r%head%spe%vtype sic%code%velo%obs
  let r%head%spe%vconv sic%code%conv%rad
  let r%head%spe%doppler 0.d0

  write

next ispec
```

B The Fortran API

B.1 Fortran demonstration program

```
program classdemo
  use class_api
  ! Local
  logical :: error
  !
  ! Init
  error = .false.
  call class_write_init(error)
  if (error)  stop
  !
  ! Open (or overwrite) a Class file
  call classdemo_openfile(error)
  if (error)  stop
  !
  ! Write all the observations
  call classdemo_writeall(error)
  if (error)  continue  ! Continue to ensure correct cleaning
  !
  ! Close the Class file
  call classdemo_closefile(error)
  if (error)  continue  ! Continue to ensure correct cleaning
  !
  ! Quit Class cleanly
  call class_write_clean(error)
  if (error)  stop
  !
end program classdemo
!
subroutine classdemo_openfile(error)
  use gildas_def
  use class_api
  logical, intent(inout) :: error
  ! Local
  character(len=filename_length) :: file
  integer(kind=entry_length) :: size
  logical :: new,over,single
  !
  file = 'classdemo.30m'  ! Output file name
  new = .true.            ! Create new file or append?
  over = .true.           ! Overwrite file if it already exists?
  size = 1000             ! Maximum number of observations per (V1) file
  single = .true.         ! Single/multiple file kind
  call class_fileout_open(file,new,over,size,single,error)
  if (error)  return
  !
end subroutine classdemo_openfile
!
subroutine classdemo_closefile(error)
```

```
  use class_api
  logical, intent(inout) :: error
  !
  call class_fileout_close(error)
  if (error)  return
end subroutine classdemo_closefile
!
subroutine classdemo_writeall(error)
  use class_api
  logical, intent(inout) :: error
  ! Local
  type(observation) :: obs  ! Use a custom observation, not the R buffer
  integer(kind=4) :: iobs
  !
  call class_obs_init(obs,error)
  if (error)  return
  !
  do iobs=1,100
    call classdemo_fillobs(obs,error)
    if (error)  return
    !
    call class_obs_write(obs,error)
    if (error)  return
  enddo
  !
  call class_obs_clean(obs,error)
  if (error)  return
  !
end subroutine classdemo_writeall
!
subroutine classdemo_fillobs(obs,error)
  use gbl_constant
  use phys_const
  use class_api
  type(observation), intent(inout) :: obs    !
  logical,           intent(inout) :: error  !
  ! Local
  integer(kind=4) :: nchan,ichan
  !
  nchan = 128
  !
  call class_obs_reset(obs,nchan,error)
  if (error)  return
  !
  obs%head%presec(:) = .false.  ! Disable all sections (except next ones)
  !
  ! General
  obs%head%presec(class_sec_gen_id) = .true.
  obs%head%gen%num = 0  ! 0 = Automatic numbering
  obs%head%gen%ver = 0  ! 0 = Automatic increment at write time
  obs%head%gen%teles = 'MYTELES'
  obs%head%gen%dobs = 0
```

```
  obs%head%gen%dred = 0
  obs%head%gen%kind = kind_spec
  obs%head%gen%qual = 0
  obs%head%gen%scan = 1
  obs%head%gen%subscan = 1
  obs%head%gen%ut = 0.d0
  obs%head%gen%st = 0.d0
  obs%head%gen%az = 0.
  obs%head%gen%el = 0.
  obs%head%gen%tau = 0.
  obs%head%gen%tsys = 100.
  obs%head%gen%time = 100.
  obs%head%gen%parang = 0.
  obs%head%gen%xunit = 0
  !
  ! Position
  obs%head%presec(class_sec_pos_id) = .true.
  obs%head%pos%sourc = 'MYSOURCE'
  obs%head%pos%system = type_eq
  obs%head%pos%equinox = 2000.0
  obs%head%pos%proj  = p_none
  obs%head%pos%lam   = pi/2.d0
  obs%head%pos%bet   = pi/2.d0
  obs%head%pos%projang = 0.d0
  obs%head%pos%lamof = 0.
  obs%head%pos%betof = 0.
  !
  ! Spectro
  obs%head%presec(class_sec_spe_id) = .true.
  obs%head%spe%line = 'MYLINE'
  obs%head%spe%restf = 123456.d0
  obs%head%spe%nchan = nchan
  obs%head%spe%rchan = 1.
  obs%head%spe%fres = 1.
  obs%head%spe%vres = -1.
  obs%head%spe%voff = 0.
  obs%head%spe%bad = -1000.
  obs%head%spe%image = 98765.d0
  obs%head%spe%vtype = vel_obs
  obs%head%spe%vconv = vconv_rad
  obs%head%spe%doppler = 0.d0
  !
  ! Data
  do ichan=1,nchan
    ! Fill with dummy values
    obs%data1(ichan) = sin(2.*pi*ichan/nchan)**2
  enddo
  !
end subroutine classdemo_fillobs
```

B.2 API description

```
subroutine class_write_init(error)
  !---------------------------------------------------------------------
  ! @ public
  !   Public procedure suited for programs linked basically to the
  ! libclass, i.e. with no Gildas specific features like interpreter or
  ! initializations or so. See demo program classdemo-telwrite.f90
  !   Perform the basic initializations of the libclass needed for
  ! the writing process (e.g. stand-alone program at the telescope).
  !   Should be done once at program startup.
  !---------------------------------------------------------------------
  logical, intent(inout) :: error  ! Logical error flag
end subroutine class_write_init
subroutine class_write_clean(error)
  !---------------------------------------------------------------------
  ! @ public
  !   Public procedure suited for programs linked basically to the
  ! libclass, i.e. with no Gildas specific features like interpreter or
  ! initializations or so. See demo program classdemo-telwrite.f90
  !   Perform the basic cleaning of the libclass needed after the
  ! writing process (e.g. stand-alone program at the telescope).
  !  Should be done once before leaving the program
  !---------------------------------------------------------------------
  logical, intent(inout) :: error  ! Logical error flag
end subroutine class_write_clean
subroutine class_fileout_open(spec,lnew,lover,lsize,lsingle,error)
  use classic_api
  !---------------------------------------------------------------------
  ! @ public
  !  Public entry point to open a new or reopen an old output file.
  !---------------------------------------------------------------------
  character(len=*),           intent(in)    :: spec     ! File name including extension
  logical,                    intent(in)    :: lnew     ! Should it be a new file or an old?
  logical,                    intent(in)    :: lover    ! If new, overwrite existing file or raise an
  integer(kind=entry_length), intent(in)    :: lsize    ! If new, maximum number of observations allow
  logical,                    intent(in)    :: lsingle  ! If new, should spectra be unique?
  logical,                    intent(inout) :: error    ! Logical error flag
end subroutine class_fileout_open
subroutine class_fileout_close(error)
  !---------------------------------------------------------------------
  ! @ public
  ! CLASS External routine (available to other programs)
  ! Close the output file.
  !---------------------------------------------------------------------
  logical, intent(inout) :: error  ! Error status
end subroutine class_fileout_close
subroutine class_obs_init(obs,error)
  use class_types
  !---------------------------------------------------------------------
  ! @ public
  !  Initialize the input observation for later use
```

```
  !  See demo program classdemo-telwrite.f90
  !-----------------------------------------------------------------------
  type(observation), intent(inout) :: obs    !
  logical,           intent(inout) :: error  !
end subroutine class_obs_init
subroutine class_obs_reset(obs,ndata,error)
  use class_types
  !-----------------------------------------------------------------------
  ! @ public
  !  Zero-ify and resize the input observation for later use
  !  See demo program classdemo-telwrite.f90
  !-----------------------------------------------------------------------
  type(observation), intent(inout) :: obs    !
  integer(kind=4),   intent(in)    :: ndata  ! Number of values the observation can accept in return
  logical,           intent(inout) :: error  ! Logical error flag
end subroutine class_obs_reset
subroutine class_obs_write(obs,error)
  use class_types
  !-----------------------------------------------------------------------
  ! @ public
  !  Write the observation to the currently opened output file
  !  See demo program classdemo-telwrite.f90
  !-----------------------------------------------------------------------
  type(observation), intent(inout) :: obs    !
  logical,           intent(inout) :: error  ! Logical error flag
end subroutine class_obs_write
subroutine class_obs_clean(obs,error)
  use class_types
  !-----------------------------------------------------------------------
  ! @ public
  !  Clean/free the input observation before deleting it
  !  See demo program classdemo-telwrite.f90
  !-----------------------------------------------------------------------
  type(observation), intent(inout) :: obs    !
  logical,           intent(inout) :: error  !
end subroutine class_obs_clean
```

C The Python API

C.1 Python demonstration script

```
import pyclassfiller
from pyclassfiller import code  # The various codes needed by the Class Data Format
import numpy   # The Y data (and CAL%COUNT) must be numpy arrays

fileout = pyclassfiller.ClassFileOut()
fileout.open(file='classdemo.30m',new=True,over=True,size=999999,single=True)

obs = pyclassfiller.ClassObservation()
obs.head.presec[:]            = False  # Disable all sections except...
obs.head.presec[code.sec.gen] = True   # General
obs.head.presec[code.sec.pos] = True   # Position
obs.head.presec[code.sec.spe] = True   # Spectroscopy
nchan = 128

for i in xrange(100):
    # Fill the observation with dummy values
    #
    obs.head.gen.num = 0
    obs.head.gen.ver = 0
    obs.head.gen.teles = "MYTELES%s" % (i+1)
    obs.head.gen.dobs = 0
    obs.head.gen.dred = 0
    obs.head.gen.kind = code.kind.spec
    obs.head.gen.qual = code.qual.unknown
    obs.head.gen.scan = 1
    obs.head.gen.subscan = 1
    obs.head.gen.ut = 0.
    obs.head.gen.st = 0.
    obs.head.gen.az = 0.
    obs.head.gen.el = 0.
    obs.head.gen.tau = 0.
    obs.head.gen.tsys = 100.
    obs.head.gen.time = 100.
    obs.head.gen.parang = 0.
    obs.head.gen.xunit = code.xunit.velo  # Unused here
    #
    obs.head.pos.sourc = "MYSOURCE%s" % (i+1)
    obs.head.pos.system = code.coord.equ
    obs.head.pos.equinox = 2000.0
    obs.head.pos.proj = code.proj.none
    obs.head.pos.lam = numpy.pi/2.
    obs.head.pos.bet = numpy.pi/2.
    obs.head.pos.projang = 0.d0
    obs.head.pos.lamof = 0.
    obs.head.pos.betof = 0.
    #
    obs.head.spe.line  = "MYLINE%s" % (i+1)
    obs.head.spe.restf = 123456.7
```

```
    obs.head.spe.nchan = nchan
    obs.head.spe.rchan = 1.
    obs.head.spe.fres = 1.
    obs.head.spe.vres = -1.
    obs.head.spe.voff = 0.
    obs.head.spe.bad = -1000.
    obs.head.spe.image = 98765.
    obs.head.spe.vtype = code.velo.obs
    obs.head.spe.vconv = code.conv.rad
    obs.head.spe.doppler = 0.
    #
    obs.datay = numpy.sin(2.*numpy.pi / nchan *
                  numpy.array(range(nchan),dtype=numpy.float32))
    #
    obs.write()

fileout.close()

del obs,fileout
```

C.2 API description

```
ClassFileOut.open(file,new,over,size,single)

    Open a Class output file. There must be only one output file opened
    at the same time.

    Parameters
    ----------
    file: the file name
    new: can the file already exist, i.e. reopen it for appending (False),
          or should it be new (True)?
    over: overwrite (True) the previous version of the file, if 'new' is
          requested?
    size: the maximum number of observations the file will store (V1 Class
          files, obsolescent parameter)
    single: if True, there can be only one version of each observation
          in the file.

ClassFileOut.close()

    Close a Class output file.

    Parameters
    ----------
    None

ClassObservation.write()

    Write the observation to the output file currently opened (see
    ClassFileOut.open). Sections present must have been declared and
    and filled before, they must be at least the General, Position,
```

```
and Spectroscopy or Continuum Drift sections.

If obs.head.gen.num is 0, Class will automatically give the next
available (unused) number in the file.
```