Schematic view of *GasCell* software

A. Díaz Pulido

Informe Técnico IT-OAN/CDT 2016-26

Contenido

1 Introducción

En este manual se pretende dar cuenta de las funciones a ejecutar por el usuario del progama *gascell*, así como de una vista esquemática del funcionamiento interno de éste. Además, se pretende dar cuenta de información complementaria, como el Monitor de espectros, información referente a los logs y la manipulación a nivel de usuario de la base de datos mediante Python.

2 Utilización de $gasCell$

- ***operator****(id='NN')*

  Establece el nombre de la persona que realiza pruebas en *gascell*.

- ***source****(name='')*

  El nombre de la fuente o reactivo que se esté observando.

- ***project****(pid='GC-NNN-YYYY', createDir=True)*

  El proyecto (carpeta) en el que se guardarán los diferentes logs y el archivo tipo class. Ejemplo: 'GC-042-2016'

- ***receiver****('hetName', freqMHz = 43.122e3, lineName='default')*

  Aquí se define la frecuencia a observar: nombre del receptor a usar, valor de la frecuencia en MHz, nombre de la línea. Ejemplo: receiver('het45', 43122, 'SiO(v=1,1-0)')

- ***backend****(sectionList=[3,4], back_end='FFTS')*

  Con este comando se establece el espectrómetro a usar:

  Combinaciones sectionList / back_end: [1,2],[3,4], [5,6] ó [7,8] para back_end= 'FFTS' & [1,2] para back_end= 'FFTS2'

Es imprescindible llamar a las últimas dos órdenes al iniciar *gascell*, mientras que la tercera es sólo recomendable, y las dos primeras opcionales.

A continuación se detallan los modos de observación que permite el software:

2.1 Calibración

Los dos posibles modos de calibrar nuestras medidas se detallan a continuación:

- ***cal****(intTime, pauseNeeded=True, coldSkyOrN2 = "sky")*

  Toma dos espectros de calibración a la frecuencia anteriormente definida (con receiver()): primero cielo (o muestra de interés) y, tras una pausa, la carga fría empleada ("sky" o "N2").

- ***calFSW****(intTime, pauseNeeded=True, coldSkyOrN2 = "sky",fswAmplitude=48)*

  Toma dos espectros en cada una de las etapas mencionadas en cal() (muestra / carga fría), separadas una frecuencia fswAmplitude [MHz], con la frecuencia definida por receiver() en el centro. Es decir, toma 4 espectros en total.

2.2 Conmutación en posición

- ***offon****(intTime, reps=1, dec=1, pauseNeeded=True, coldSkyOrN2 = "sky", processData=True, rfOnOffSwitch = 'False')*
- ***offonoff****(intTime, reps=1, dec=1, pauseNeeded=True, coldSkyOrN2 = "sky", processData=True)*

2.3 Conmutación en frecuencia

- ***fsw****(intTime, fswAmplitude, reps=1, coldSkyOrN2 = "sky", cal2FSW=False, processData=True)*

  "Frequency Switching" scan con resultado 1 sólo espectro por polarización en Class, donde:

  intTime: Si el tiempo de integración es mayor a 5 segundos se tomarán intTime/5 subscans en cada frecuencia.

  fswAmplitude: separación en frecuencia de las 2 fases

  reps: número de subscans que se realizan, aunque el resultado final sea un sólo scan.

  cal2FSW: False si se desea calibrar con una sóla frecuencia (central) y tras haber realizado una calibración previa con cal(). True si se desean calibrar ambas fases con su respectiva calibración en frecuencia, imprescindible haber ejecutado calFSW() anteriormente.

- ***fswcal****(intTime, fswAmplitude=48, reps=1, coldSkyOrN2 = "sky", processData=True)*

  Función de uso idéntico a fsw(con cal2FSW=True), pero con una diferencia en su funcionamiento interno: No utiliza el canal de notificaciones para establecer la conmutación en frecuencia, lo que disminuye considerablemente algunos "tiempos muertos". Este método necesita ser "testeado" mejor.

2.4 Repetición de un scan

- ***cycles****(ncycles=1)*

  Si se quiere llevar a cabo un mismo tipo de scan repetidas veces(ncycles), y que cada vez se genere un espectro, se puede llevar a cabo mediante este método. Cuando se utiliza pedirá que se introduzca el comando de observación (cualquiera de los anteriormente listados en esta sección).

3 gascellMonitor

/home/almamgr/scanMonitorgascellMonitor.py

4 Logs

/home/almamgr/nanocosmos/gascellLogs

5 Obteniendo informacion de la base de datos

Todas las variables relevantes se guardan (además de en el archivo de Class) en la base de datos. Se puede acceder a ésta por ejemplo mediante Python, con los siguientes comandos:

» import _mysql

» db = _mysql.connect('172.16.10.157','cellgasuser','ga5c311','gascelldb')

» pregunta = 'SELECT * FROM dataScan'

En lugar de * se puede preguntar por cualquiera de los campos que se detallan más adelante. Al final de la pregunta, se puede añadir un condicional como 'WHERE sourcename = GC_ORION'

» db.query(pregunta)

» filas = db.store_result().fetch_row(0)

Cada una de las filas contendrá la información de los campos que se listan a continuación para cada subscan y polarización:

id pID idScan scantype offID onstartTime nsections centralSection nchannels centralChannel bwSection freqGHz fswAmpMHz tint tempEnv tempGas pressGasCap pressGasPir RFON azim elev taus h2o tcold tsys linename sourcename operator

6 Esquema

- initLogger ()

  Inicia el Logger, para que queden guardados los diferentes logs en la ruta: /home/almamgr/nanocosmos/gascellLogs

- gascell ()

  Nos da información de su funcionamiento

- *class* globalVar → gv

  Clase que permite almacenar, modificar y hacer llamadas a los valores de ciertas variables usadas a lo largo del programa.

  - *__init__* ()
    El lugar donde están almacenados los valores de las variables que gestiona esta clase

- resetCounter ()

  Pone a cero el contador de scans referido al projecto. Utiliza el objeto gv

- stop ()

  Detiene la adquisición de datos. Utiliza el objeto gv

- settings (writeRawData=False)

  Indica si se generarán espectros en formato CLASS. Utiliza el objeto gv

- silent (mode='off') !No se usa

  No devuelve información en el prompt de GasCell. Utiliza el objeto gv

- warnings (mode='on') !No se usa

  Cambia el estado de alertas. Utiliza el objeto gv

- weatherDataHandler (weatherStr)

  Mediante la correspondiente subscripción al canal de notificaciones, hace lectura de los datos de temperatura, presión y humedad relativa ambientes. Utiliza el objeto gv

- timeNow (printIt=True)

  Devuelve la hora actual en formato HH:MM:SS

- logCell (logline, target=0, color=False, timeTag = True, msresolution = True)

  Muestra el log por pantalla y/o lo escribe en el correspondiente archivo.log con opciones de línea: color y añadir marca temporal (mostrando o no hasta milisegundos)

- errorLog (logLine, errorLevel, target=0)

  Define el color de la línea según el errorLevel establecido, target indica si queremos escribir una línea por pantalla y/o guardarla en el log. Utiliza logCell()

- debug (level = 1)

  Modifica el nivel de depuración del log, lo que implica que no se escribirán/guardarán aquellas líneas con un nivel de depuración menor al de fullDebug (variable global)

- printDebug (message, debugLevel = 1, color=1,logwrite = True)

  Permite asignar un nivel de depuración a una cierta línea de código. Utiliza errorLog()

- project (pid='GC-NNN-YYYY', createDir=True)

  Define el nombre del proyecto y crea el correspondiente directorio si éste no existe.

- __checkProjectIDDirectory (dataPID)

  Verifica la existencia del proyecto.

- __createPIDDirectory (dataPID)

  Crea el proyecto

- operator (id='NN')

  Establece las iniciales del usuario.

- bye ()

  Libera el uso de ciertos componentes, se desconecta de los canales de notificaciones y corta los hilos creados que sigan en uso.

- subscribeToNotificationChannels

  Se subscribe a los canales de notificaciones del FFTS y de la estación meteorológica, declarando los objetos creados como globales.

- gettimestamp (string=True,nowDT='',join=False)

  Devuelve la fecha en formato string (YYYY-mm-dd, HH:MM:SS) o no (YYYmmdd, HHMMSS.mcs). Esta fecha puede ser la actual o una que introduzcamos. La salida puede ser un solo string con toda la información, o 2 salidas con la fecha y la hora por separados.

- getFileName (pID, idScan, scantype, onstartTime, section)

  Devuelve el nombre del archivo con su ruta correspondiente, según los identificadores que pasemos como argumentos. Todas las alusiones a escritura y lectura de archivos deben utilizar este método, para garantizar la homogeneidad en el criterio de creación de ficheros.

- getPreScanInfo()

  Meditante llamadas a diferentes objetos, obtiene información de interés para poder devolver parámetros como la frecuencia actual, el vapor de agua, temperaturas de cielo y del sistema, etc, que serán introducidos en la base de datos. Este método sólo es llamado por setPreScan() en la clase scan().

- *class* dataBase $\rightarrow$ datab

  Esta clase tiene por objetivo gestionar el uso de la base de datos, ya sea introduciendo la información referida a un scan o extrayendo de ésta información de interés.

  - *__init__* (ipaddress, user, passwd, db)
    Establece la conexión a la base de datos, utilizando la dirección ip, nombre de usuario, contraseña y el nombre de la base de datos.
  - reconnect()
    Cierra y vuelve a conectar.
  - disconnect()
    Cierra la conexión
  - getInfoDB ()
    Devuelve dos arrays, uno con los nombres de los campos que figuran en la base de datos, y otro con el tipo de variables que se guardan.
  - updateScan (scanInst)
    Introduce una nueva línea en la base de datos, una vez que estos han sido tomados. El objeto de entrada (scanInst = currentScan) hace referencia a la instancia de la clase scan(), en cuyos parámetros de inicialización se encuentran los que aquí se emplean.
  - show1000lastScans ()
    Muestra las últimas 1000 líneas de la base de datos.
  - getLastXFile (scantypeS, nchannelsS, bwSectionS, freqGHzS, fswAmpMHzS)
    Busca y encuentra, en los últimos 1000 scans de la base de datos, la observación que concuerde con los parámetros de entrada. Devuelve la información necesaria para identificar el archivo que contiene los datos de la observación, y su ubicación.

  - getLastScan () !No se usa
  - getMaxScan (pID)
    Devuelve el número del último scan realizado en el proyecto que se indica. Si no hay, éste será cero.
  - getnsections (pID, startTime, idScan)
    Identifica el número de secciones usadas en un scan.
  - getId (pID, startTime, idScan, section)
    Obtiene el número identificador (siempre acumulativo) de un scan según los parámetros de entrada.
  - getAll (id, pID)
    Devuelve una lista con los valores de todos los campos de la base de dados asociados a un determinado scan, que queda unívocamente definido por su identificador y proyecto.
  - getTempEnv (id)
    Obtiene el valor de la temperatura en la sala de receptores para el scan correspondiente.
  - getTcold (id)
    Obtiene la temperatura de la carga fría empleada en el scan indicado.

- getFSWSign (currentPhase, fswAmplitude)

  Devuelve el signo del incremento en frecuencia para un scan de FSW, cero si no es ese tipo de scan.

- fftsDataHandler (fftsStr)

  Gestiona el uso de ciertas variables del FFTS, tales como el cambio en frecuencia, y se encarga de gestionar la estructura que éste devuelve. En esta estructura podemos encontrar, entre otros, los datos de los números de cuentas por canal. El array generado es guardado en un fichero ASCII

- *class* scan → currentScan

  - *__init__*()
    Variables de inicialización, correspondientes a las 27 variables que se guardan en la base de datos y que lleva asociadas el objeto currentScan que instancia a esta clase.
  - fillScan () !No se utiliza
  - setIntTime (inttimesecs)
    Modifica el valor del t_int en el objeto currentScan.
  - setProject (projectId)
    Modifica el proyecto en el objeto currentScan.
  - getProject ()
    Devuelve el nombre del proyecto en el objeto currentScan.

- setOperator (operator)

  Modifica el nombre del operador/usuario en el objeto currentScan.

- setSourceName (sourceName)

  Modifica el nombre de la fuente/compuesto químico que esté siendo observado, en el objeto currentScan.

- setScan (number, dtimepy)

  Modifica el número de scan (dentro del proyecto) en el objeto currentScan.

- setGasPressure (pressPir, pressCap)

  Modifica los valores de presión medidas por los sensores Pirani y Capacitron en el objeto currentScan.

- setRFState (rfon)

  Modifica el valor del generador de RF (RFGen) en el objeto currentScan. 1: Encendido, 0: apagado.

- setFrequency (lineName, freqGHz)

  Modifica el valor de la frecuencia sintonizada y el nombre asignado en el objeto currentScan.

- setFSW (fswAmp)

  Modifica el valor de la conmutación en frecuencia, en MHz, que esté siendo utilizada (0 si no hay tal conmutación); modifica también el booleano que indica si se trata de un scan de FSW o no, en el objeto currentScan.

- setfswFlag (boolean=True)

  Modifica el booleano que indica si se trata de un scan de FSW o no, en el objeto currentScan. No es redundante con la función anterior, ambas se emplean de manera independiente.

- setPreScan (stype, intTime, fullinfo=True)

  Modifica, en el objeto currentScan, los valores de: scantype, intTimeSecs, bwSection, nchannels, centralChannel, nsections, tempEnv, tempGas, presGasPir, presGasCap, rfon, az, el, taus, h2o, tcold y tsys. 11 de esos 17 parámetros se consiguen llamando al método getPreScanInfo(), que demora 1.3 segundos en su ejecución, por lo que puede obviarse su utilización en algunos casos (con fullinfo=False) y los valores que se emplean son los guardados con anterioridad en la variable de la clase glovalVariables, scanInfoArray.

- timestamp () !No se emplea

- onStartMjd ()

  Devuelve el día juliano modificado.

- onStartUTC ()

  Retorna el tiempo universal coordinado.

- onStartLST ()

  Devuelve el tiempo sidereo local.

  - setTsys (tsys)
    Modifica el valor de la temperatura de sistema en el objeto currentScan.
  - setSection (backendNumber)
    Modifica el valor de la sección usada en el objeto currentScan.

- *class* spectralBackend → ffts

  Esta clase se encarga de manipular y configurar el backend elegido. Las opciones pueden ser: 'FFTS' y 'FFTS2'

  - *__init__* (instanceName = 'FFTS')
    En caso de que haga falta, configura el FFT.
  - *__configure* ()
  - release ()
    Libera el componente.
  - stopIntegration ()
    Detiene la integración
  - startIntegration ()
    Arranca la integración.
  - setNumPhases (numphases = 1)
    Si hay un cambio en los tiempos 'muertos' y de integración, configura el FFT.
  - setIntegrationTime (syncTimeMs = 5000, blankTimeMs = 5, typeOfObs = 'OBS')
    En caso de que hayan cambiado los tiempos blankTime y syncTimeMs configura el FFT, se tolera hasta un 0.2 % de error relativo.
  - setPatch (sectionList = [7,8])
    Modifica las secciones del FFT que están siendo utilizadas, sus respectivos anchos de banda y canales que utilizan.
  - getBandwidth (section)
    Devuelve la sección del FFT en uso.
  - getNumChannels (section)
    Devuelve el número de canales asignado a esa sección.
  - getLowestSection ()
    Devuelve la primera sección en uso.
  - getSectionList ()
    Retorna la lista de secciones en uso por el FFT en cuestión.
  - getFFTSname ()
    Nos da el nombre del FFT que estamos utilizando.
  - __del__ ()
    Libera el componente.

- *class* frontend → Rx

  Clase que gestiona el uso del receptor en cuestión.

  - *__init__* (hetName = 'het45', freqMHz = 43.122e3, line = "default")
  - setFreq_and_Attenuation (line = 'default', freqMHz = 43e3)
    Nos redirecciona a setFrequency() y setIFAttenuation()
  - setFrequency (line = 'default', freqMHz = 43e3)
    Configura la frecuencia de cielo del receptor y el nombre con el que designemos a esa línea.
  - setIFAttenuation ()
    Atenúa el receptor según el valor de la frecuencia y del receptor usado.
  - setIFCenter (ifCenter)
    Nos da el valor de la diferencia en frecuencia entre la frecuencia de cielo y la IF del oscilador local.
  - connectRx ()
  - getFrequency ()
    Modifica la frecuencia en el objeto Rx.
  - __del__ ()
    Libera el componente.

- receiver (freqMHz = 43.122e3, lineName='default')

  Crea el objeto Rx (instancia a frontend)

- backend

  Crea el objeto ffts (instancia a spectralBackend)

- source (name = '')

  Define el nombre de la fuente/compuesto siendo observado.

- operator (id='NN')

  Define el nombre del usuario/operador que maneja el programa *gasCell*

- __checkNTP ()

  Comprueba la sincronización con *ARIES*

- __fsCommand__ (commLine)

  Permite insertar una instrucción en el Field System.

- funcCommandLine (func)

- cycles (ncycles=1, preScanInfo = True)

  Permite realizar multiples observaciones del mismo tipo de scan, cada uno de los ciclos generará un espectro en *CLASS*. preScanInfo = False para evitar la demora de 1.3 segundos en el primero de los subscans que conforman cada uno de los scans.

- run (file, verbose=True)

  Intenta ejecutar una macro programada en Python. No se garantiza su funcionamiento.

- activatePirani ()

  Intenta establecer conexión con el componente del sensor de presión Pirani.

- activateCapacitron ()

  Intenta establecer conexión con el componente del sensor de presión Capacitrón.

- activaterfSynth ()

  Intenta establecer conexión con el componente del generador de RF RFGen.

- integration (onReps = 1)

  Se encarga de gestionar el uso del fftsDataHandler() a través del condicional dataValid (booleano), que depende de si se ha alcanzado o no el número de scans previsto según las especificaciones de tiempos del usuario.

- setSyncTime (onReps = 1, intTimeSecs = 1, onDec = 1)

  Comprueba la viabilidad de tiempo de integración y número de repeticiones especificados por el usuario, y los cambia en caso necesario.

- obs () ! No se utiliza

- cal (intTime, pauseNeeded=True, coldSkyOrN2 = "sky")

  Calibra a una sola frecuencia un máximo de 5 segundos en cada carga, si se requiere pausa el programa esperará hasta que pulsemos INTRO para integrar sobre la carga caliente. La carga fría puede ser Sky (cielo) o N2 (N_2 líquido).

- calFSW (intTime, pauseNeeded=True, coldSkyOrN2 = "sky",fswAmplitude=48)

  Similar a la anterior calibración, pero en doble frecuencia, la separación entre ellas viene dada por fswAmplitude. Es decir, a las frecuencias: la definida en receiver() más o menos fswAmplitude/2.

- otf (intTime, reps=1, calibrated = True ,coldSkyOrN2 = "sky", processData=True, dec=1)

  Modo de observación que sólo integra a una determinada frecuencia. Podemos elegir generar el espectro en CLASS con los datos calibrados o no.

- onoff (intTime, reps=1, dec=1, pauseNeeded=True, coldSkyOrN2 = "sky", processData=True, rfOnOffSwitch = 'False')

  Conmutación en posición.

- offon(intTime, reps=1, dec=1, pauseNeeded=True, coldSkyOrN2 = "sky", processData=True, rfOnOffSwitch = 'False')

  Conmutación en posición integrando primero "fuera" de la fuente de interés, o con la cámara de gas vacía, o sin el plasma encendido.

- offonoff(intTime, reps=1, dec=1, pauseNeeded=True, coldSkyOrN2 = "sky", processData=True)

  Conmutación en posición, que integra con la cámara vacía tanto antes como después de llenar ésta.

- fsw (intTime, fswAmplitude, reps=1, coldSkyOrN2 = "sky", cal2FSW=False, smooth_reference = 1, processData=True, reference = [])

  Conmutación en frecuencia. Podemos elegir calibrar cada fase o a una sola frecuencia (cal2FSW), antes se ha de haber realizado el scan de calibración pertinente. Si se calibra cada frecuencia, las opciones posibles son: "suavizar" (smooth_reference) la referencia, y elegir la referencia indicando en reference[] un array con los scans de inicio y fin de un FSW anterior, por defecto calibra con carga fría.

- calBand () ! No se usa aún

  Calibra la banda completa, fruto de la unión de diferentes secciones del FFTS.

- *class* temperatures → tObject

  - *__init__* ()
    tsysPolArray nos indica las temperaturas del sistema en cada sección del FFT, puestas a cero al inicializar.
  - tsys (scanHot, scanCold, section=7,freq_index='')
    Realiza las cuentas pertinentes para devolvernos la temperatura del sistema, la temperatura estimada de receptor y el rms de la T_sys; todo ello en función de los parámetros de entrada: scans de carga caliente y fría, sección y calibración doble o no.

- createSpectrum (*args)

  Función que gestiona los métodos getSpectrum() / getFSWCalibratedSpectrum() y writeGildasSpectrum() a través del uso de *hilos*, para permitir la escritura y análisis de datos en paralelo a una nueva adquisición de datos.

- *class* processData → dp

  Clase que gestiona diversos métodos referentes a manipulación de magnitudes vectoriales y escritura y lectura de archivos (tanto ASCII como CLASS)

  - *__init__* (pid, dataDir, datab)
    Utiliza el objeto datab para solicitar información de la base de datos, y tiene en cuenta la ruta del proyecto en el que nos encontremos.
  - setProcess (pid, dataDir)
    Establece la ruta del proyecto.
  - getSubscanTypeList (date, section, startScan, endScan, onoffFlag = 'OFF')
    Obtiene una lista de archivos que encajen con la descripción que se da.

- getAverageScanScaled (scanList,powerList, nchannels) ! Ya no se emplea
  Promedia una los vectores de una lista de ficheros y los calibra según el tiempo de integración.
- getAverageScan (scanList, nchannels)
  Promedia una los vectores de una lista de ficheros.
- smoothArray (listArray, channelSmooth)
  Suaviza un vector en función del número de canales que especifiquemos (channelSmooth)
- rms (lista,n=2)
  Calcula la desviación típica de un vector cuyo ajuste puede ser de orden 0, 1 ó 2.
- fromFileToArray (fileName)
  Devuelve un vector de un archivo ASCII.
- writeRawDataSpectrum (date, scan, section, onoffFlag) ! No se emplea
- getSpectrum (proj, date, section, startScan, endScan, startScanOn, numOns, fsw=False, intTimeSec=5, onoffFlag = 'ON', OTFcalibrate = True)
  Devuelve los parámetros necesarios para escribir un espectro en CLASS: pectralChannel,date,endScan, section, numOns, fsw, False, totalTime. Los modos de observación que analiza son: otf, onoff, offon, offonoff y fsw(cal2FSW=False)
- getFSWCalibratedSpectrum (proj, date, section, startScan, endScan, startScanOn, numOns, fsw=True, intTimeSec=5, smooth_reference=1, reference =[])
  Devuelve los parámetros necesarios para escribir un espectro en CLASS: pectralChannel,date,endScan, section, numOns, fsw, False, totalTime. Sólo gestiona el análisis de datos referentes al modo de observación: fsw(cal2FSW=True)
- getBestScanFile (date , onFileName, onoffFlag = 'OFF', scanOrTime = 'scan')
  Encuentra y devuelve el nombre del archivo de datos, más cercano en scans o tiempo al actual, que se ajusta al criterio especificado por los parámetros de entrada.
- processAll (fsw=False) ! Método a revisar
  Genera de nuevo los espectros en CLASS del proyecto especificado.
- processSingleScan (date, idScan) ! Método a revisar
  Introduce un espectro (todas las polarizaciones) en CLASS de la observación que indiquemos.
- processSingleSection (date, idScan, section,fsw)
  Introduce un sólo espectro (sólo una polarizació) en CLASS de la observación que indiquemos.
- fillArray (content, nchannels)
  Convierte un vector con longitud=nchannels en otro vector con el formato adecuado (el que CLASS "pide" para generar un espectro).
- getCalFactor (nchannels, hotdata, colddata, thot, tcold)
  Obtiene el factor de calibración.

- getCalibratedFSWSpectrum (nchannels, ondata, calf)
  Calibra los datos de integración especificados según el factor de calibración indicado.
- getCalibratedSpectrum (nchannels, ondata, offdata, calf)
  Realiza la substracción entre una pareja de datos y calibra la diferencia.
- writeGildasSpectrum (spectralChannel, date, idScan, section, nReps=1, fswFlag=False, writeRawSpec=False, totalTime=10)
  Introduce en el archivo de CLASS, especificado según el proyecto en el que nos encontremos, el espectro que le indicamos, con los valores pertinentes según los parametrós de entrada del método y los que haya en la base de datos para el scan que indicamos.

7 Anotaciones

Algunas de las mejoras/arreglos que se han llevado a cabo:

- El número de scans máximo por proyecto es de 10000
- Los espectros en Class muestran correctamente el tiempo de integración. Además de señalar la temperatura del sistema para cada polarización, se tiene en cuenta la calibración precedente para obtener el espectro final resultante.
- Se han eliminado muchísimos tiempos muertos innecesarios (como de configuración del back_end), llegándose a observar eficiencias temporales con el comando fswCal() de entre 50%-70%.