The ALMA QA2 Calibration Script Generator almaqa2csg.py (CSG) for speeding up and standardising the generation of scripts used for manual calibration of ALMA data. On this page you find the documentation of the public version of it (which may be a version or two behind the latest version in use at the ALMA observatory).

The CSG has a number of optional input parameters. However, the only necessary input parameter is the name of the ASDM (raw ALMA data set) to be calibrated. Based on the properties of the dataset, almaqa2csg creates a **"scriptForCalibration.py"** tailored to the needs of the particular case. The result is a nearly complete script draft which *may work as is* but should be inspected by the analyst to make sure all steps are correct. So, the simplest possible call to the CSG looks like this

```
import almaqa2csg as csg

csg.generateReducScript('uid___A002_X7c0875_Xd65')
```

This will import the given ASDM into an MS (named uid___A002_X7c0875_Xd65.ms in this case) and produce a calibration script draft file named "uid___A002_X7c0875_Xd65_scriptForCalibration.py" for it.

NOTE: The CSG expects the ASDM to be named by only the Execution Block (EB) UID, e.g., `uid___A002_X7c0875_Xd65` as above. Any extensions like ".asdm.sdm" need to be removed. Soft links are accepted.

The calibration scripts produced by the CSG use the now common **"stepping mechanism"** (which was added by D. Petry in 2012):
the idea is that the user of the script controls the execution of the different procedural steps of the calibration **without having to edit the script**,
thus avoiding the risk of introducing bugs in the script. The mechanism works as follows:

In the script header, the user can inspect a Python dictionary which contains the different steps of the script.
Example:

```
step_title = {0: 'Import of the ASDM',
              1: 'Fix of SYSCAL table times',
              2: 'listobs',
              3: 'A priori flagging',
              4: 'Generation and time averaging of the WVR cal table',
              5: 'Generation of the Tsys cal table',
              6: 'Generation of the antenna position cal table',
              7: 'Application of the WVR, Tsys and antpos cal tables',
              8: 'Split out science SPWs and time average',
              9: 'Listobs, and save original flags',
              10: 'Initial flagging',
              11: 'Putting a model for the flux calibrator(s)',
              12: 'Save flags before bandpass cal',
              13: 'Bandpass calibration',
              14: 'Save flags before gain cal',
              15: 'Gain calibration',
              16: 'Save flags before applycal',
              17: 'Application of the bandpass and gain cal tables',
              18: 'Run renormalization',
              19: 'Split out corrected column',
              20: 'Save flags after applycal'}
```

For a correct calibration, all these steps have to be executed in the given order.
But the user can execute one or a few steps at a time in order to inspect intermediate results or rerun certain steps after modification.
At the CASA prompt, the user sets the variable **"mysteps"** (note the "s" at the end!), e.g.

```
mysteps = [2,3,4,5,6]
```

Then the calibration script is run using

```
execfile('uid___A002_X7c0875_Xd65_scriptForCalibration.py')
```

and this will execute only the steps 2, 3, 4, 5, and 6 of the script and then stop. The user can then look at the various plots produced by the script or use the task plotms to inspect calibration tables.
Later, the user can either make modifications to the script and repeat steps or just continue setting, e.g.

```
mysteps = [4,5,6]
```

and calling execfile again.

NOTE: The order in which you enumerate the steps in the mysteps variable does not matter. The steps will always be executed in the order given in the script.
If you don't set mysteps, running the script will execute all steps from 0 to the last.

NOTE 2: **Important**: some steps will not only produce plots and caltables but also apply flagging! If you need to modify these steps and re-run them, **you need to revert the flagging first**!
For this purpose, the calibration script includes calls to the flagmanager task to save the state of flags before they are modified.
You can revert to the previous flagging state by running the flagmanager task in mode "restore" and providing the corresponding versionname which you can find in the calibration script.

NOTE 3: the **renormalization step** is only present if there are FDM SPWs in the EB. You are expected to run this step first as-is in order to check whether renormalization is needed. If so, you need to edit the script and set applyRenorm to True *and then run the step again* to apply the renormalization.

The CSG is somewhat computing intensive. The execution time depends on the size of the input dataset and can take several 10 minutes, especially when the MS has not yet been imported and importasdm has to be run.

Pay close attention to error messages on the terminal.

The CSG internally uses the analysisUtils and up to version 1.27, the CSG can work with CASA 5.1.1 or later. CASA 6 is recommended.
For QA2, use the same CASA version which is also presently used by the Pipeline.
Starting with version 1.28, the CSG requires a CASA version >= 6.2.1 with ALMA pipeline in order to have access to the renormalisation module.

If you want to use the CSG with a CASA version which does not contain the ALMA pipeline (i.e. the renorm module), or if you just don't want the renorm step to be generated even though FDM science SPWs are present**, you need to set the parameter includeRenorm=False .**

For Cycle 9, use version 1.30 or later with CASA 6.4.1 with ALMA pipeline.

Access

The CSG is contained in a single Python module **almaqa2csg** which is part of the "analysis_scripts.tar" package. This package also contains the "analysisUtils" which you will also need since the CSG uses a few of their functions.
See the Zenodo analysisUtils page (presently for version 2.49, 12 July 2023) (which provides a **citeable DOI**: 10.5281/zenodo.8140844) and the **Analysis Utilities CASA Guide** for details on how to install the package.

Be sure to check the Zeondo page for the presence of a later version. There are several releases per year.

Usage

With your sys.path set up to import the analysisUtils, type

```
import almaqa2csg as csg
help csg.generateReducScript
```

to obtain the following help (status version 2.3, 2023/02/01):

```
Help on function generateReducScript in module almaqa2csg:

generateReducScript(msNames='', step='calib', corrAntPos=True, timeBinForFinalData=0.0, refant='',
bpassCalId='', chanWid=1, angScale=0, run=False, lowSNR=False, projectCode='', schedblockName='',
schedblockUid='', queue='', state='', upToTimeForState=2, useLocalAlmaHelper=True, tsysChanTol=1, sdQSOflux=1,
runPhaseClosure=False, skipSyscalChecks=False, lazy=False, lbc=False, remcloud=False, bdfflags=True,
tsysPerField=False, splitMyScienceSpw=True, bpassCalTableName='', reindexMyScienceSpw=False,
useCalibratorService=True, calibratorServiceURL=None, allowHybrid=False, combineB2BLFspws=False,
includeRenorm=True)

    The ALMA QA2 calibration script generator

    msNames: a string or a list of strings of UIDs (either ASDM or MS) to process
             NOTE: rigorous regression testing is presently only done on single UIDs, not lists
              default=''
    step:    calib, fluxcal, wvr, calsurvey, SDeff, SDcalibLine, SDcalibCont, SDscience, SDampcal
              default='calib'
    corrAntPos: if True, then run correctMyAntennaPositions
              default=True
    timeBinForFinalData: a value in seconds (string, int, or float), passed to split
              default=0.
    refant:  the reference antenna to use (instead of automatic selection), must be a string
              default='', i.e. determine automatically
    bpassCalId: use the specified source for bandpass (rather than determine from the intents)
              default='', i.e. determine from the intents
    chanWid: integer, used by runCleanOnSource and searchForLines
              default=1
    angScale: deprecated

    run:     deprecated

    lowSNR:  Boolean passed to doBandpassCalibration to use whole spw for pre-bandpass phase-up
              default=False
    projectCode, schedblockName, queue, state, upToTimeForState: deprecated

    useLocalAlmaHelper: if True, run tsysspwmap inside generator, rather than in the resulting script
              default=True
    tsysChanTol: integer argument passed to tsysspwmap
              default=1
    sdQSOflux: flux density to use for quasar in single dish case (step='SDeff')
              default=1
    runPhaseClosure: deprecated

    skipSyscalChecks: if True, then don't check for negative Tsys problems
              default=False
    lazy: value of the 'lazy' parameter in importasdm. If True, reference the ASDM instead
            of copying the visibilities into the DATA column of the MS. Saves disk space.
              default=False
    lbc:    if True,  in bandpass calibration, use solint='inf,8MHz' instead of 'inf,20ch'
              default=False
    remcloud: if True, run the recipe remove_cloud prior to running wvrgcal
              default=False
    bdfflags: passed to importasdm to invoke the application of BDF flags
              default=True
    bpassCalTableName: to use instead of default name
              default='', i.e. use the bp table created for bpassCalId with the standard naming
    phaseDiff: deprecated (BWSW and B2B modes are recognized automatically)

    tsysPerField: passed to the perField parameter of tsysspwmap
          default=False
    splitMyScienceSpw: In the final split-out, only include the SPWs corresponding to intent OBSERVE_TARGET
              and BANDPASS.
              default=True
    reindexMyScienceSpw: perform reindexing in the split out after the apriori calibration.
              default=False
    useCalibratorService: if True, then, in the call to aU.getALMAFluxForMS in the setjy step, use
              aU.calibratorService(), otherwise use aU.getALMAFlux()
              default=True
    calibratorServiceURL: the URL to pass to aU.calibratorService() if useCalibratorService==True
              default: None - use the default of calibratorServiceURL in aU.getALMAFluxForMS()
    allowHybrid: if False, only the antennas of the dominant (most often occuring) antenna diameter are split
out.
               If True, all antennas are split out.
              default=False
    combineB2BLFspws: if True, and if the dataset uses band-to-band phase transfer, then combine the LF SPWs
               in gaincal and use appropriate spwmaps.
              default=False
    includeRenorm: if True, a step is added in the calibration script to perform renormalization
              default=True
```

Feedback, bug reports, feature requests

Feedback, bug reports, feature requests should be made by email to dpetry@eso.org .