Running CASA Functional Tests on Multi-MS

Sandra Castro

Version 1.1

This document is a quick guide on how to run CASA functional tests[1] using Multi-MSs[2]. The Multi-MS (MMS) data sets are not stored in the data repository as to avoid duplicating space. Each developer can create a reference set of Multi-MSs for their tests and use them when updating the code. The same should be made in the CASA test system, for the automated tests. This guide assumes the developer has installed the MPI packages in his development version or is using the latest CASA binaries. The Appendix shows how to install the MPI RPMs on your system. See the complete documentation on the MPI framework in CASA[3]. Throughout this text, replace "mpirun" with "mpicasa" if you are using the CASA binaries.

NOTE: If the user does not have the MPI libraries, CASA will fall-back to use simple_cluster. Note that in most of the functional tests, the execution will not be faster when running on MMS, mostly because the datasets of the tests are small and the overhead of setting up the cluster may be dominant.

1. Create Multi-MS for the tests

There are 3 ways of creating Multi-MS data sets for the functional tests (those run using the runUnitTest.py tool). 1) Create MMS automatically for the MSs used in a particular test script. 2) Create MMS for a given directory containing MSs. 3) Use the partition task to have full control and create the MMSs manually.

1.1 Automatically with the make_mmsdata script

Using the make_mmsdata.py script, which uses the convertToMMS() class (explained in next item) to create MMS. This script will create MMS for all MSs used in a defined test using default parameters to the partition task. The developer can modify some parameters of partition.

```
mpirun -n 5 casapy –c $INSTALL_DIR/make_mmsdata.py <options>
task1 task2 …
```

where $INSTALL_DIR =*<casa_install_dir>/python/<PYVER>/regressions/admin*

Options:

no option	Print this message and exit.
--all	Create MMS for all tasks in TASKLIST.

--ignore	From all tasks, do no create MMS for the given <tasks>
--list	Print the list of tasks from TASKLIST and exit.
--axis	partition separationaxis to use (spw, scan, auto); default=auto
--numsubms	Number of subMSs to use when creating MMS; default=4
<tasks>	Create MMS data for the given tasks. Additional tasks are separated by white spaces.

Examples

- Get the usage information.
```
casapy –c make_mmsdata.py
```

- Give any option to casapy before the *–c* argument.
```
casapy --nogui --log2term –c make_mmsdata.py
```

- Create MMS data for the gaincal tests using the default auto axis and 4 subMS.
```
mpirun -n 5 casapy –c make_mmsdata.py gaincal
```

- Create MMS data for all tasks defined in TASKLIST, except flagdata and split, and create 8 subMS.
```
mpirun -n 5 casapy –c make_mmsdata.py --numsubms=8 --
ignore flagdata split
```

- Create MMS data for all tasks defined in TASKLIST using the scan axis.
```
mpirun -n 5 casapy –c make_mmsdata.py --axis=scan --all
```

1.2 Semi-automatically with the convertToMMS() class

Using the convertToMMS() class inside CASA. This class will create MMS data for all the Measurement Sets of a given directory. It will ignore any non-MS data such as calibration tables. It will also create symbolic links from the MMS to a MS, to be easily recognized when running the functional tests. Start casapy with mpirun -n 5 and do the following:

```
CASA>: import partitionhelper as ph
>> Get the usage of the class
CASA>: ph.convertToMMS()

=================================================================
Options:
  inpdir <dir>              directory with input MS.
  mmsdir <dir>              directory to save output MMS. If not given, it will save
                            the MMS in a directory called mmsdir in the current
                            directory.
  axis='auto'               separationaxis parameter of partition
                            (spw,scan,auto).
  numsubms='auto'           number of subMSs to create in output MMS
  createmslink=False         if True it will create a link to the new MMS with
```

	extension .ms.
cleanup=*False*	if True it will remove the output directory before starting.

NOTE: this script will run using the default values of partition. It will try to
create an MMS for every MS in the input directory. It will skip non-MS directories
such as cal tables. If partition succeeds, the script will create a link to every
other directory or file in the output directory. This script might fail if run on
single dish MS because the datacolumn needs to be set in partition.

The script will not walk through sub-directories of inpdir. It will also skip
files or directories that start with a .
==

Example

- Create MMS for all MSs present in the given directory and save them in the default directory "mmsdir".

CASA>: convertoToMMS(inpdir='CASADATA/regressions/unittest/bandpass', createmslink=True)

1.3 Manually with partition

Run task partition manually to create Multi-MS by hand inside CASA and have more control on the parameters of the task. See help partition for more details.

Example

- Create MMS for the Four_ants_3C286.ms and select only the DATA column. Create flag backup and use the spw axis for the partition. Use the "listpartition" task to see the content of the MMS.

CASA >: partition('Four_and_3C286.ms', outputvis='mytest.mms', separationaxis='spw', datacolumn='DATA', flagbackup=True)

2. Modify the functional tests

In order to run the existing functional tests with a different data set, there is an option in runUnitTest.py, which will look for MSs in a different location other than that defined in the tests themselves. The script runUnitTest will set an environmental variable called TEST_DATADIR when it is called with the option --datadir. This variable can be read by the tests to use a different location for the input data sets.

Add the following lines in the beginning of the test script. See examples in test_flagdata.py.

```
# Path for data
datapath = os.environ.get('CASAPATH').split()[0] + "/data/regression/unittest/flagdata/"

# Pick up alternative data directory to run tests on MMSs
testmms = False
if os.environ.has_key('TEST_DATADIR'):
   DATADIR = str(os.environ.get('TEST_DATADIR'))+'/flagdata/'
   if os.path.isdir(DATADIR):
      testmms = True
      datapath = DATADIR
      print 'flagdata tests will use data from '+datapath
```

This assumes that the MMS data is stored under a sub-directory with the task name. Most of the existing functional tests follow the recommended way of storing MSs in the data repository, under $CASADATA/regression/unittest/<taskname>. Tests that read data from other locations need to be adjusted accordingly. One easier option is to create symbolic links to MSs from other locations to the standard one in $CASADATA/regression/unittest/<taskname>.

The following tests already support MMS such as described above: *test_bandpass, test_clearstat, test_concat, test_conjugatevis, test_cvel2, test_flagdata, test_fluxscale, test_gaincal, test_gencal, test_hanningsmooth2, test_listhistory, test_listobs, test_listvis, test_plotms, test_split2, test_uvcontsub, test_virtualconcat, test_vishead, test_visstat.* For these tests, one only needs to create MMS and run the tests with the --datadir option.

3. Run the tests on MMS

Run the tests as you normally do to check that they all pass. Create the MMSs as described in Section 1 and run the same tests on the new data sets. If the MMS are created under unittest_mms/<taskname>, run the script as follows:

```
mpirun -n 5 casapy –c $INSTALL_DIR/runUnitTest.py --
datadir=unittest_mms test_taskname
```

If you are using the casa binaries, use the "mpicasa" alias, which will find the MPI libraries installed in the package.

```
mpicasa -n 5 casapy –c $INSTALL_DIR/runUnitTest.py --
datadir=unittest_mms test_taskname
```

4. Troubleshooting

What to look for when the tests pass using normal MSs but fail on Multi-MSs. The first thing to look at is if the separation axis used to partition the MS is appropriate to the processing done by the task.

5. References

1) http://www.eso.org/~scastro/ALMA/CASAUnitTests.htm
2) https://svn.cv.nrao.edu/svn/casa/trunk/gcwrap/python/scripts/mpi4casa/CASA-4.3-Multi-MS-Structure-doc.pdf
3) https://svn.cv.nrao.edu/svn/casa/trunk/gcwrap/python/scripts/mpi4casa/CASA-4.3-MPI-Parallel-Processing-Framework-Installation-and-advance-user-guide.pdf

6. Appendix

Follow these steps for a quick installation of the MPI libraries in your CASA development build.

- yum install:
  - numactl-devel-2.0.9-2.el6.x86_64
  - casa01-openmpi-1.6.5-1.el6.x86_64
  - casa01-mpi4py-1.3.1-1.el6.x86_64
- Rebuild your CASA normally
- Test the installation by running the mpi tests:
  - mpirun -n 5 casapy -c runUnitTest.py test_mpi4casa
    (where 5 means 4 servers and 1 client)