Script for ALMA data packaging (Cycle 0/1/2)

EU ARC

September 11, 2014

Contents

1 Python modules and methods used

We decided to minimize the use of functions like `os.system` or `os.popen` (i.e., we avoided, whenever possible, direct calls to the command line), since either the input or output of the shell commands may depend on the environment where the script is being run (e.g., bash vs. tcsh), as well as the underlining operating system (linux/unix/macOS) and/or its version.

Hence, we have used functions in the `os` and `shutil` modules that are independent of the local shell (e.g., `os.walk`, `shutil.copy`, etc.) and work the same way in all the POSIX systems. This approach makes the script fully portable (virtually independent of the environment and operating system where it is being run).

2 Running the script

The script can be run either as a standalone script (called from CASA) or as a module to be imported. If it is run as an independent script, the command to be executed is (in CASA)

`execfile(''QA2_Packaging_module.py'')`

and if it is loaded as a module in another script, the import is made by adding the line

`from QA2_Packaging_module import *`

The function that executes the packaging is called `QA_Packager`, and depends on several parameters. If the script is run in standalone mode, these configuration parameters are set inside the script, so there is no configuration file nor arguments to be passed through the command line during the script invocation. If the script is loaded as a module, an example call to the packaging function would be

```
QA_Packager(origpath='./my_QA2', readme='./README.header.txt',
packpath='./2011.X.00YYY.Z', mode='fake', style='cycle2-nopipe')
```

In this example call, the script would make a *fake* copy (i.e., just generate empty files at the destination folder) to help the user checking the output before the actual packaging. In addition, any data previously saved in the destination folder would be removed before running the packaging (i.e. `append=''`; see below).

The arguments of the `QA_Packager` function are:

1. `origpath` → The location of the QA2 reduction folder.
2. `packpath` → The path to the destination folder (which should have the project code as name).
3. `readme` → The path to an ascii file with the text of the README header (this file can be named, for instance, `README.header.txt`).
4. `mode` → The copying mode (see below).
5. `style` → The packaging style (see below).
6. `append` → (optional, default = empty string) The appending mode (see below).

The parameter `mode` can take the following values:

- `mode = 'fake'` → This is the default. The script creates empty (i.e., dummy) files at the destination folder.
- `mode = 'copy'` → The files are copied in the normal way. This mode needs to be chosen when working across different file systems.

- `mode = 'hard'` → The script generates hard links in the destination folder. This way, the file-pointers at both the origin and the destination folders, refer to the same physical locations (i.e., inodes) in the disk.

- `mode = 'move'` → The files are moved from the origin to the destination (*not recommended*). Then, symbolic links are created at the origin. The symbolic links are never made to whole folders, but only to files (to avoid, for instance, an accidental deletion of whole folders at the destination path by removing the content of a linked folder).

- `mode = 'ticket'` → Similar to 'fake', but the files to be added to the JIRA ticket are hard-linked. This way, the packager will generate valid ticket tar files, but the measurement sets and tables will not be copied.

The parameter `style` is meant as a one-stop control for special features of the packaging which may change from Cycle to Cycle. This concerns the inclusion of MSs and of pipeline products. It can take the following values:

- `style = ''` → This is the default. With this value the parameter is essentially ignored and parameters like `noms` control what is happening.

- `style = 'cycle0'` or `style = 'cycle0-nopipe'` → MSs are included.

- `style = 'cycle1'` or `style = 'cycle1-nopipe'` → MSs are not included.

- `style = 'cycle2-nopipe'` → MSs are not included.

- `style = 'cycle2-pipe1'` → MSs are not included. Pipeline products are searched for and included.

In regard to the parameter `append`, it can take the following values:

- `append = ''` → This is the default. It removes any previous data at the destination folder before the packaging.

- `append = 'group'` → The script appends a new group id to the destination folder (so the other groups, if any, are not removed).

- `append = 'member'` → The script appends new member id(s) to the group with highest id (so the other groups, if any, and the other member ids, if any, are not removed).

- `append = 'product'` → The script appends the data to the already-existing member of highest id (in the group of highest id). If there were more than one member ids in the data to be packaged, the data of the first member id will be appended to the already-existing member id, and the rest of member ids will be appended in separate ids.

The rest of parameters are described in the following sections.

3 Arrangement of member ids

The script finds the folders with data and products by looking for the existence of an ASDM and/or one or several measurement sets. The search for ASDMs and MSs is not based on name templates, but on the data content. For instance, the subfolder `ASDMBinary` is always found in any ASDM, as well as the subfolder `DATA_DESCRIPTION` is always found in any MS. The script also identifies calibration tables by looking for the subfolder named `CAL_DESC`.

Classifying the data in terms of the content of the folders, instead of their names, makes the algorithm more robust to changes, for instance, in the naming convention of calibration tables.

Based on the location of the ASDM products, the script can *guess* if the dataset is made of one or several member ids of a group. It also looks for possible *combined products* (i.e., it looks for folders where there are measurement sets, but no ASDM-related stuff).

The data in the QA2 can be arranged in many different ways, but some minimum consistency should be ensured to allow the script classifying well the different members ids (provided that this feature is going to be used). There is no limitation to the number of ASDMs analyzed in each folder, but the basic assumption made by the script, in any case, is that all the direct products of each ASDM are located in the same directory.

In Fig. 1, we show the expected data structure if there is only one member id. The folders represented with diamonds are not mandatory (i.e., the combined products and/or the folders for the different ASDMs could be moved to the parent directory). The data represented with ovals are not required (i.e., there is no need to have combined products for groups with more than one member).

In Fig. 2, we show the expected data structure when there are several member ids. The plotting convention is the same as in the previous figure. If there is more than one member id, it is not necessary to reduce the data for these members in subfolders for different ASDMs. Notice, though, that if this strategy is followed with all the members in a dataset, there may be confusion with the cases corresponding to Fig. 1.

Finally, there is also the possibility of analyzing one single folder with the products of one or several ASDM. In that case, no search for different member ids is done. The initial folder structure of this simple case is that of Fig. 3.

A possible strategy is to run the script several times (one per folder) with `append = 'member'` (for instance) or run it a minimum number of times (i.e., identifying, in one run, all the member ids of each group). In any case, the parameter `append` controls if the destination folder is completely removed each time the script is run or, on the other hand, new member or group/member subfolders are created (so no data are removed from the destination folder). This appending feature may be useful if the script is

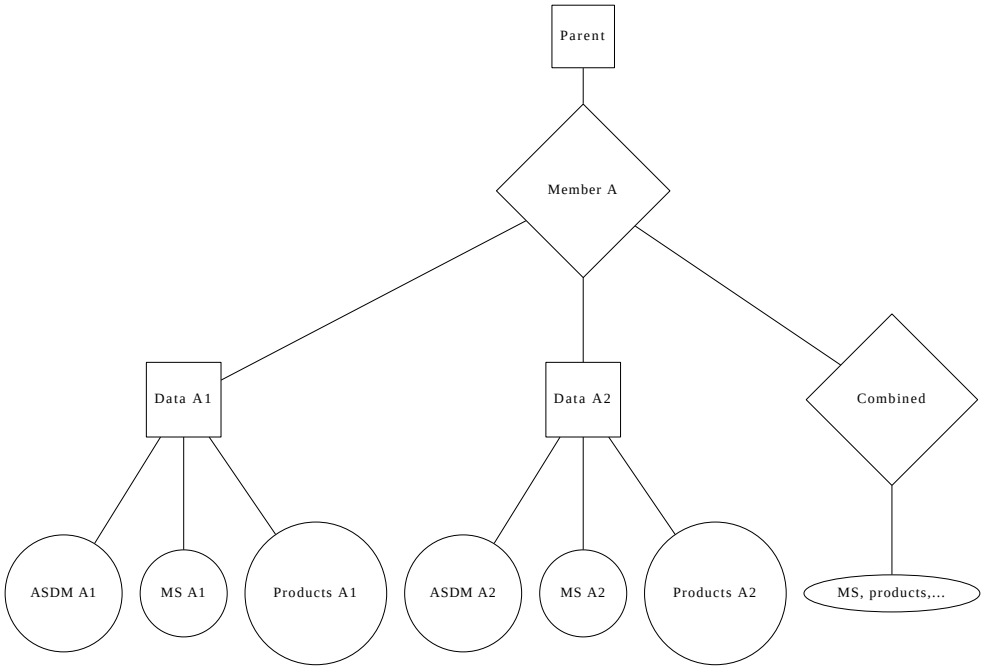

Figure 1: Initial folder structure recognized by the script as corresponding one single member id. Notice that different combinations are possible (since the folders shown in diamonds can be taken out from the structure). The name of each folder is not important for the script. Each folder can have products of several ASDMs.

going to be run as part of a larger batch processing script.

4 Naming convention

The script does not change the name of any file or folder with data or plots. The only renaming performed in the packaging corresponds to the group ids and member ids. The script sorts the names of the member folders in alphanumeric order, and assigns their destination directories with the standard convention `member_ouss_id` plus an integer (if there is more than one member found).

5 Copying the data

Once the data of the different member ids have been found, the script generates the final folder structure and begins the data packaging.

- All the calibration tables are copied to the folder named according to the *tablefolder* parameter. Different subfolders are used for the different ASDMs (these subfolders are named as the ASDM, but followed by the

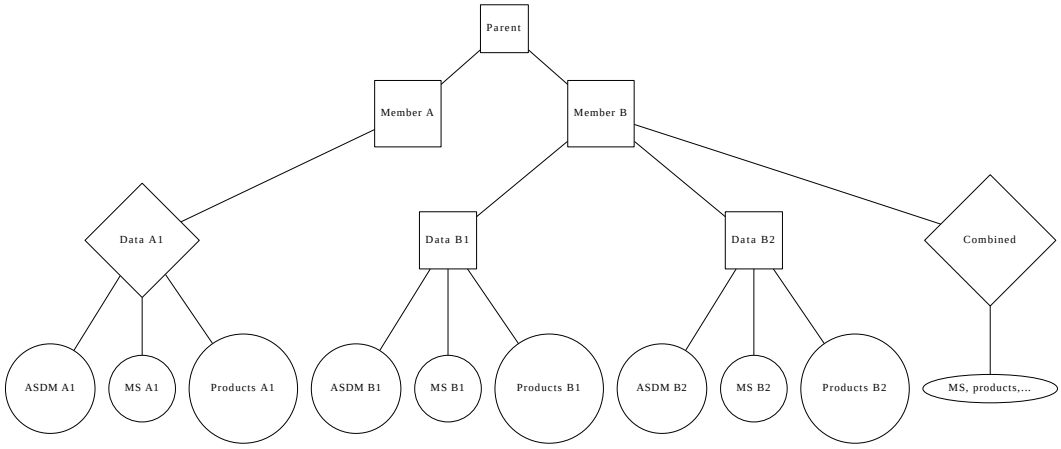

Figure 2: Initial folder structure recognized by the script as corresponding to different member ids (each of them with one or more ASDMs). The name of each folder is not important for the script.

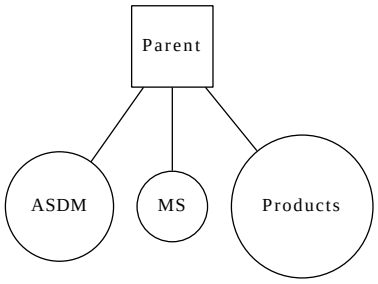

Figure 3: Initial folder structure for the most simple case

extension ".calibration"). If there is a calibration table that does not match this naming convention (it should never happen, but could happen for very special calibrations), it will also be saved, but just at the level of *tablefolder*.

- All the folders full of `png` images are assumed to be related to plots of the content of the calibration tables. Hence, they are copied to the `tableplotsfolder` folder on an ASDM/MS basis (to keep consistency with the folder structure of the tables) in folders named as the ASDM plus the extension ".calibration.plots".

- Depending on the content of the lists named *rawdatacopy* and *caldatacopy* (these are lists with the name extensions of the measurement sets to be packaged), the script will copy either the raw asdm, the a-priori-calibrated ms, and/or the fully-calibrated ms to the folders named according to the parameters *rawdatafolder* and *caldatafolder*, respectively.

- The scripts (i.e., files ending with `.py`) and the logs (i.e., files ending with `.log` or `.log.txt`) are saved in their corresponding subfolders. The name of the container folder in each data reduction is appended to the names of the log files, to ensure that no log file will be over-written.

- Finally, the files and folders whose names are described in the list `QA2_others` are assumed to be related to QA2. Hence, these files and folders (together with the checklist files, the `.tbl` folders, and the `html.tgz` files) are saved directly in the `qafolder` folder.

- If there were combined products, the `product`, `script`, and `log` folders are created and filled at the member-id level.

- Finally, the script creates a "ticket tar" file. If mode is 'fake', the tar file will be made of empty files, but if it is set to 'ticket', a valid ticket tar file will be created (and all files not to be added to the ticket will be copied in 'fake' mode).

6 Sanity checks

There are two lists named *required_folders_CAL* and *required_files_QA2*. These lists contain the extensions of the calibration tables and QA2-related files (respectively) that should appear in **all** the datasets. If the script cannot find any of them, it will raise a warning message (although the packaging will continue). Notice that these are not lists of all tables and files to be copied, but lists of the *required* ones. The script will always copy all the calibration tables found in the ASDM folder being treated, regardless of their name and extension.

The script also checks, on-the-fly, if there are files (or folders) with equal names that are going to be copied to the same location. In case there are, it raises a warning message.

Finally, the script checks if there is at least one script, log, and product for each ASDM.

7 External ephemeris tables

As long as ASDMs do not yet contain the Ephemeris table, ephemerides need to be packaged separately with the calibration products since they will have to be attached to the MSs using the task "fixplanets". The packager will pick up any files ending in ".eph" and package them into the "calibrated" directory.

The following checks are performed:

1. If more than one ephemeris table with the same name is found, it is verified that the two files are indeed identical. If not, the packaging is aborted since the ephemerides need to be renamed such that different files have different names.

2. The string "fixplanets" is searched in the calibration scripts.

   If it is found but no ephemeris tables were found, a warning is issued. It is just a warning since fixplanets can also be used for other purposes than attaching ephemerides.

   If it is not found, but ephemeris tables are found, a different warning is issued telling the user that potentially fixplanets calls may have been forgotten or the ephemeris tables are superfluous.

8 Packaging including Pipeline products - the Pipeline stager

Since August 2014, the Packager module includes a second function which serves to stage pipeline output for imaging.

If you have run the ALMA Pipeline and produced output which you want to package for further processing elsewhere, you can use the packager to create a "delivery style" package.

Also, if you have received such a "delivery style" package from elsewhere and have done the imaging part of the QA2 work and now want to package everything for final delivery to the PI, the packager enables you to do that as well.

Both use cases are described below.

Case 1 An SB was processed by the pipeline and is to be packaged for imaging at the ARCs or elsewhere

Then run the packager as follows:

```
QA_Pipeline_Stager(pipeline_root='<top dir of pipeline results>',
                   staged_root='<project ID == top dir of output package>',
                   mode='<hard, copy, fake>',
                   PIscript="<path to scriptForPI.py>")
```

Example:

```
QA_Pipeline_Stager('2013.100456.S-2014-08-08T123456', '2013.100456.S',
```

Case 2 A package as created in Case 1 above has arrived at the ARC (or a JAO analyst).

The analyst has run the scriptForPI to create the *.ms.split.cal(s) and verified that the scriptForPI is working in this case.

The analyst proceeded to perform the imaging in the "calibrated" directory which was created by the scriptForPI.

The resulting fits files reside in the "calibrated" directory and the scriptForImaging resides there as well or in the "script" directory.

The result is QA2_PASS and now the whole data is to be packaged for delivery.

Then run the packager as follows:

```
QA_Packager(origpath='<top dir of the analysis dir>',
            packpath='<project id = top dir of output package>',
            readme='<path to readme file>',
            mode='<hard,copy,fake,ticket>',
            gzip_caltables=True,
            style='cycle2-pipe1',
            PIscript="./scriptForPI.py")
```

Example:

```
QA_Packager(origpath='./2013.100456.S-analysis',
            packpath='./2013.100456.S',
            readme='myREADME.txt',
            mode='hard',
            gzip_caltables=True,
            style='cycle2-pipe1',
            PIscript="./scriptForPI.py")
```

9 Packager Unit Test

In the same repository directory where you find the packager module, you also find a "unit test" for it. This enables you to confirm that the basic functionality of the packager is working.

Usage:

```
tar xvzf QA2_Packaging_module.unit-test.tgz
cd QA2_Packaging_module.unit-test
cp ../QA2_Packaging_module.py .
start casapy and run
    execfile('QA2_Packaging_module.unit-test.py')
```

Acknowledgements

Thanks very much to Ivan Marti-Vidal (Onsala) and Anita Richards (Manchester) for producing the first useful version of the script. Thanks to Ivan Marti-Vidal and Adam Ginsburg (ESO) for continued support.