Espresso

Espresso is the system developed for managing correlation at ATNF. It is a lightweight system for managing data on your cluster, automating the correlation process, and providing simple archiving of the outputs. It is designed for correlation from standard linux disks (not direct from Mark5s or eVLBI). Espresso also provides a number of auxiliary scripts which may come in handy during correlation. A typical espresso session, as it is used at ATNF, is available here.

All the scripts will give help if invoked with the `-h` switch.

Installing espresso

The scripts come with your DiFX installation (2.0.2 and later), in `$DIFXROOT/applications/espresso`. The included `install.py` script will install them in your DiFX bin directory:

```
$DIFXROOT/applications/espresso/install.py $DIFXROOT
```

To work they need a cluster definition file. See the `corr_hosts.txt` file in `$DIFXROOT/applications/espresso` for an example.

The environment variable `$DIFX_MACHINES` should point at your version of the cluster definition file (corr_hosts.txt described above). As this correlator definition file is unrelated to the particular version of DiFX you are using, you probably want to store it in your home directory, or similar.

Espresso allows you to write the output data to a directory other than the one in which the correlation files are stored (this is useful for installations where the NFS disks are too small to store the output data). You should set the environment variable `$CORR_DATA` to point to the directory where you want the output data to be stored. The output data will be stored in a subdirectory of `$CORR_DATA` with the experiment name.

Espresso will automatically sniff the data areas given in the `$DIFX_MACHINES` file for baseband data. The baseband data should be stored in subdirectories of the given data areas with the following naming convention:

`<expname>-<tel>`

where `<expname>` is the name of the experiment, and `<tel>` is the telescope station code, as used in the .v2d file.

Running the Espresso Scripts

disk_report.py

```
disk_report.py > ~/disk.json
```

this script will sniff all the data areas given in $CORR_HOSTS and summarise the baseband data distributed across your cluster. You should save the output to a convenient place (e.g. ~/disk.json).

disk_exper.py

```
disk_exper.py <expname> ~/disk.json
```

this script extracts the telescope baseband data locations from the output of disk_report.py. It takes 2 arguments: the experiment for which you want a data summary (<expname>), and the file where you saved the output of disk_report. It will write a summary of the baseband data locations for each telescope in a file <expname>.datafiles (example).

lbafilecheck.py

```
lbafilecheck.py <expname>.datafiles
```

this script will do a parallel search of the baseband data locations in <expname>.datafiles to extract the full file list for correlation. These are written as a series of .filelist files (one per telescope). (Advanced users might wish to note that it is possible to restrict the files that are selected by use of the pattern match specified at the top of the file, as described here.)

In addition it creates a machines, threads and run file for MPI. The generic run file written by default will work in many cases, but if you need a bespoke mpi command (e.g. if you use SLURM, SGE or PBS to launch jobs), then there is a mechanism for providing your own prototype run file which lbafilecheck/espresso will modify with the appropriate job name.

espresso.py

```
espresso.py -a <expname>
```

running the script with these parameters will run the correlation for every job generated by running vex2difx <expname>.v2d. It will modify the machines and threads file for each job, automatically taking care of telescopes that are not present in some jobs. Output will be written to a subdirectory of $CORR_DATA.

In turn it will run:

- vex2difx
- calcif2
- errormon2
- mpifxcorr

All the auxiliary files (.calc, .im, .input, etc.) required for converting the output data to IDI fits are copied to the output directory (with modified internal paths). The log file will also be copied to the output directory when the job finishes. If there are any files already in the output directory which need to be overwritten, they will first be copied to a subdirectory (whose name matches the time that

the new correlation started).

Once initiated, the script may prompt you to enter a gmail address (and passwd) where progress notifications will be sent (not used for batch schedulers). Press `return` if you do not wish to use this.

At the start and end of the correlation, the script will pause to allow the operator to enter a summary message on how the correlation went. By default that message will be entered using the vim editor, but you may set the `$EDITOR` environment variable to another editor if you prefer.

On completion, `espresso.py` will print a list of jobs whose log files are missing the usual 'BYE' messages - these are typically jobs that did not complete.

The behaviour of the script can be modified with a number of command line switches. Information on these can be obtained with:

```
espresso.py -h
```

In the case where you do not wish to run all the jobs created by vex2difx, you may select a subset by giving those jobs as arguments (and dropping the -a switch), e.g.:

```
espresso.py v389b_1 v389b_2
```

would run the first 2 jobs created by: `vex2difx v389b.v2d`

You may also use a python regular expression to match the part of the job name after the '_', e.g.

```
espresso.py 'v278b_1[1-3]'
```

would run jobs v278b_11, v278b_12, v278b_13. (Note you may need to quote regular expressions to prevent the shell from expanding them.)

Multiple Passes

Some experiments need to be correlated multiple times, with different correlator parameters (e.g. separate line and continuum passes). To simplify the logistics of this, espresso has the concept of a correlator pass. Different passes need different .v2d files, but usually all other files remain the same (vex, filelists, etc.). In espresso, to correlate a multiple pass experiment you should rename the .v2d file to be '<expname>-<passid>.v2d'. Espresso recognises the '-' as being a separator between experiment name and pass identifier and will copy the output files appropriately. No other action should be required - the output from different passes will appear in the same output directory, but with unique names based on the 'passid' used for naming the .v2d file.

Auxiliary Tools

Espresso comes with a number of auxiliary tools to assist the weary correlator operator:

```
mjd2vex.py <date>          #converts the given <date> between the various
DiFX date formats (MJD, vex, ISO8601, VLBA).
```

```
getEOP.py <date>           #returns 5 days of EOPs around <date> in .v2d
format. <date> can be any DiFX date format.
updateclock.py             #update the clock entry in the .v2d file (given
residual clock offset and rate).
updatepos.py               #update a site position in the .vex file
(requires that $STADB points to a SCHED locations.dat file)
atcapos.py <date>          #returns the ATCA tied array position (pad name)
at <date>
plot_logtime.py <difxlog>  #extract the correlation and observation times
from a difx log file and plot the correlator speedup factor
joblen.py <.joblist>       #summarise the .joblist file produced by vex2difx
in human readable form.
```

Supplying the `-h` switch to any of the above should provide more usage details.

The following is deprecated but may be useful on occasion:

```
mk5scans.py <vexfile> <filelist>
```

will append start and end times to each filename entry in the .filelist file, by comparing the filename to the scan names in the given vex file. Obviously this will only work if your mark5 filenames include the vex scan name (this is very often the case). This is normally done (in a much more robust fashion) by `lbafilecheck.py`.

Some Notes on Espresso

Espresso automatically creates a machines and threads file for MPI. It assumes that the head node for correlation is the node on which you start the correlation (i.e. where you invoke espresso.py). The output data directory must be accessible from the head node.

If using a batch queueing system such as SLURM or PBS, then the head node is used only for logging and user interaction, it is not assumed to be available for computation.

In interactive mode (no batch scheduler), then by default it assumes that the head node and datastream nodes should also be used as compute nodes. You can override this with the `-H` switch to espresso.

If any of the nodes in `$CORR_HOSTS` are to be used only as datastream nodes, and never as compute nodes, then set the number of available compute threads in `$CORR_HOSTS` to 0 for that host.

Espresso requires the egenix mxdatetime module. It is installable with pip or easy_install:

```
pip install egenix-mx-base
```

or you can download from: http://www.egenix.com/products/python/mxBase/