VLBA Sensitivity Upgrade MEMO 23

A Guide to Software Correlation Using NRAO-DiFX Version 1.1

Walter Brisken

National Radio Astronomy Observatory

September 3, 2008

1 Introduction

This manual is intended for many different audiences. Typically a particular reader will only need to be concerned with a small portion of this guide, but there are a number of cross-references between sections. This manual assumes some familiarity with Mark5 units, Linux, and the general way in which a VLBI correlator is used. The following topics are discussed: running NRAO-DiFX, coexistence issues with the VLBA hardware correlator, explanation of various file/document types, and detailed installation instructions. This manual will be kept up to date with each official update to NRAO-DiFX. Please report any errors that are found in this manual to `wbrisken@nrao.edu` .

1.1 Notation

Text written in `typewriter` font represents literal text and is to be transcribed verbatim when typing and text in *italics* is to be substituted with other text, such as the specific value of the named variable. To be consistent with this notation, all mention of programs by name or filenames (and portions thereof) are written in `typewriter` font.

2 The NRAO Customized DiFX Software Correlator

This document is centered around the NRAO customization of the DiFX [1] software correlator and its supporting software. The name "NRAO-DiFX" is being given to the collection of support programs and the modified core of DiFX, `mpifxcorr`. Much of the contents here applies to other installations of DiFX as well, but keep in mind that not a lot of effort is made to generalize these instructions. Two things make the NRAO-customized version different from the stock DiFX distribution. The first is using VLBA style job scripts as the base for forming the input files for DiFX. The second is allowing correlation straight of the Mark5 units instead of processing only files that reside on the Unix filesystem. The later is expected to be a useful extension to many other users, so attempts are made to keep this functionality as general as possible. Additionally, no attempt is made to support the RPFITS output format. The customized mpifxcorr can be compiled with or without RPFITS support, however instructions for configuring `mpifxcorr` with RPFITS are not provided here. It is the goal of NRAO's software correlator effort to periodically merge NRAO extensions to `mpifxcorr` into the main development tree and to make these features useful in general. Fig. 1 shows the general data flow-path within the NRAO-DiFX software correlator system. All new parts of this diagram are described within this document.

2.1 NRAO-DiFX 1.0

The version 1.0 series of NRAO-DiFX bases job generation on the `.fx` (§9.7) files that `cjobgen` produces. This ensures a compatibility period during which both correlators can produce visibilities with expectations of functionally identical results, a feature critical for validation. This strategy also minimizes the required software effort at its earliest phases. Version 1.0 came with the following features:

1. A complete path from `.fx` job scripts to `.FITS` files

2. A command-line only interface
3. Documentation (you are reading it now)
4. Support for VLBA and Mark IV formats
5. Correlation directly off Mark5 modules
6. Support for all projects types except those using special modes, such as pulsars, space VLBI, and near field objects
7. Spectral and time resolution bounded only by practicality

While this version should handle most observations, fast frequency switching and geodesy experiments will produce a large number of output FITS files which may be annoying to observers and the archive. Version 1.0 was available on February 6, 2008.

2.2 NRAO-DiFX 1.1

Version 1.1 builds on version 1.0 and adds the following features:

1. Used version of `mpifxcorr` that has gone through code merge with the official version
2. Blanking of data replaced by headers (MarkIV format only)
3. Proper data weights
4. Initial Mark5B support
5. Support for oversampled data through decimation
6. Multicast status information for GUI interface
7. Correlation of moving and near field objects
8. Concatenation of multiple output files into a single or multiple FITS-IDI file(s)
9. Better support for jobs with multiple configuration tables
10. Playback off Mark5 modules with missing disks
11. Support for Amazon based Mark5 units
12. Completely replaced the “Makefile” system with better integrated alternative
13. Generation of delay model polynomials rather than tables, more like VLBA HW correlator
14. u, v, w values are derived from the delay model (and hence include corrections for aberration, near field observations, and other subtle effects) and are evaluated when writing the FITS file
15. NRAO-DiFX version accountability
16. Validation of data frames prior to decoding
17. Data evaluation (“sniffing”) built into FITS converter

This version was released on September 3, 2008. Features new to version 1.1 are marked with 1.1.

2.2.1 Bugs fixed

Here are listed some of the more important bug fixes:

1. The clock offset was used with the wrong sign in the IM table.
2. Printed precision of some important numbers (RA and Dec) was increased.
3. Autocorrelations were ordered incorrectly for observations with a single polarization.
4. The MarkIV format decoder had a 1 day off bug.
5. The MarkIV format decoder had a $64\times\textit{fanout}$ sample timing offset.
6. Several causes of crashes were fixed; no known crashes remain.
7. Missing VLBA monitor data was handled badly.
8. Due to OpenMPI peculiarity, some processing nodes would get most or all of the work in some cases, which cause the work being done on other nodes to be ignored. This was fixed by looking for results in a round-robin manner.
9. Integrations that contain data from two adjacent scans are stripped when writing FITS files.
10. Allow FITS files larger than 2GiB in size.

2.2.2 Known problems

Known bugs as of the NRAO-DiFX 1.1 release:

1. The last couple (typically 2) integrations of a job (not a scan) tend to have low weight due to a premature termination of data processing.
2. The data quality "sniffer" does not respect flags.

2.3 NRAO-DiFX 1.2

Version 1.2 is meant to be production usable. In addition to fixing bugs found in earlier versions, the following features are expected

1. Initial Mark5C support
2. Proper archive integration
3. Pulsar gating support
4. Integration with the DiFX Operator Interface (DOI)
5. Parallel processing of contemporaneous subarrays?

This version is expected by December 31, 2008

2.4 NRAO-DiFX 2.0

Version 2.0 will start allowing correlation of experiments that cannot be represented by `.fx` files and will be based mostly on vex files. Many of the programs described in this document will be upgraded or overtaken by more capable replacements. Other version 2.0 capabilities could include advanced pulsar processing, space VLBI support, multiple simultaneous phase centers, Science Data Model/Binary Data Format (e.g. EVLA and ALMA) or measurement set output format, ... Development of the 2.0 capabilities will occur in parallel with the 1.0 series features.

2.5 Other features to implement

Here is a list of other features to add to NRAO-DiFX that are not directly tied to any particular version:

1. Mark5B+ support (frames start at fraction of 1 ns boundaries)
2. Pulsar bins
3. Support for K5 format
4. Fast-forwarding over unneeded data in the native Mark5 module datastream
5. Phase cal extraction
6. Switched power extraction
7. Playback of two independent modules in one Mark5 unit
8. Non-bank-mode Mark5 support
9. Spectral selection between FFT and cross-correlation to allow for greater spectral resolution
10. Conversion of DiFX output to Mark4 correlator format
11. Support for antenna local oscillator offsets

2.6 NRAO-DiFX and AIPS

Only one task in AIPS, FITLD, has to deal with the telescope/correlator specific aspect of the FITS-IDI files that the VLBA correlator and DiFX generate. The FITS-IDI variant of FITS was first documented in AIPS Memo 102 [3], and more recently in AIPS Memo 113 [4], which will be generally available shortly. It has been modified for better support support of DiFX FITS output. In general, these changes make FITLD less telescope specific so the resulting FITS-IDI files from any NRAO-DiFX installation should be highly compatible with AIPS. Several changes have been made to the 31DEC08 AIPS as a result of DiFX testing:

1. Correction for digital *saturation* in auto-correlations is disabled for DiFX FITS files. See [2] for some details on this correction which is not needed for DiFX data.
2. Support for FITS-IDI files greater than 2 GiB in size.
3. Weather table was not populated properly.
4. FITS files with multiple UV tables would generate incomplete GEODELAY columns in CL tables (not relevant to DiFX).

It is recommended that your AIPS installation be kept up to date.

3 Cluster configuration

For production correlation, it is suggested that a dedicated user account be created; for the rest of this document it will be assumed to be "difx". This account should have few or no other uses in order to ensure that the environment is not disturbed. The user account must exist on all nodes in the cluster and ssh should be configured so that no password is required when logging into one node on the cluster from another. This user account must also exist on the Mark5 units that are used for playback. It is recommended that all computers in the cluster, including the Mark5s, run the same version of Linux to avoid library compatibility issues.

All of the nodes in the cluster, including the Mark5s, should be interconnected by a fast network and have NFS access to the directories from which correlation is to proceed. Complicated network topologies, such as having more than one cluster node attached to more than one network, can lead to unpredictable results as OpenMPI (the suggested MPI library to use with `mpifxcorr`) is network aggressive and will use any means possible to enhance performance, even if such antics are counterproductive. If your network topology is not simple be aware of any network-related issues and keep in mind that you might need to explicitly specify which network interfaces to use.

One node should be deemed the "head node". In general this node should have lots of hard disk space which is cross mounted to all the others and could serve as the network gateway to the remainder of the cluster. It is convenient, but not necessary, to locate all of the software and correlation directories physically on this node to improve the interchangeability of the other nodes. This node can participate in the actual correlation, either as the manager node, a processing node or both. By default, the head node will always be the manager node.

4 Environment variables

In addition to environment variables needed at build-time (§11), some others are needed at run time. These are:

1. `CALC_SERVER` contains the name of the computer running calcServer (§11.8). This is accessed only by program `calcif`.
2. `DIFX_ARCHIVE_ROOT` points to the base directory of the archive staging area.
3. `DIFX_GROUP_ID` the Unix group to use. If set, umask is changed to 002 and all new files/directories become group writable. 1.1
4. `DIFX_HEAD_NODE` contains the name of the cluster head node.
5. `DIFX_MACHINES` points to a file containing a list of cluster members and their capabilities. See §9.4
6. `DIFX_MESSAGE_GROUP` (optional) specifies, with `DIFX_MESSAGE_PORT`, the multicast group and port to be used for GUI and monitoring. 1.1
7. `DIFX_MESSAGE_PORT` (optional, *see above*) 1.1
8. `DIFX_VERSION` (optional, but recommended) the version of difx being used, e.g., `NRAO-DIFX-1.1` . 1.1
9. `GAIN_CURVE_PATH` (optional) points to a directory that contains keyin format files containing gain curves. This is used only by `difx2fits`. If not set, `difx2fits` will not create gain curve tables. This directory must be readable by the difx user. Every file in this directory will be read, assuming it is a keyin format gain curve, so nothing else should be stored here. This directory needs to be created by hand if it does not exist.
10. `JOB_ROOT` points to the base directory that is to contain copies of job scripts of projects to correlate. This directory must be visible by all nodes on the cluster.
11. `DIFX_LOG_DIR` points to the directory where logs shall be written. 1.1
12. `MARK5_DIR_PATH` points to a directory that is used to cache the contents of Mark5 modules. This directory must be readable and writable by the user running mpifxcorr. This directory needs to be created by hand if it does not exist. It will get populated automatically. If there are problems with playback of a module, the files in this directory can sometimes be useful.
13. `TESTS` points to a path containing test data projects.

Like the environment variables described in §11, these should all be set in shell initialization files and should be set whether the shell is used interactively or not. For the `difx` user account at NRAO, these are set in a file called `setup_difx` which is run upon login (see §4.1). Note that this file needs to be run whether the login is interactive or not; please consult the documentation for your shell if you have problems. To test if this file is being run in non-interactive sessions, try the following: `ssh` *computername* `env | grep DIFX` and make sure you see the environment variables you expect to.

4.1 Directory structure and versioning

The directory structure of the NRAO deployment of NRAO-DiFX is outlined in Fig. 2. The aim is to cleanly programs, libraries, and other version-specific files from data in a way that switching from one version (e.g., 1.1) to another (e.g., the development version) is simple, accountable, and complete, in order to assure that a self-consistent set of software is used for an entire project. Each NRAO-DiFX version has its own root directory, such as `/home/swc/NRAO-DiFX-1.1` . All files associated with this version are under this directory. No data or files associated with any other NRAO-DiFX version shall be placed within.

Setting up a particular version is quite simple. Assuming the `bash` shell:

`. /home/swc/NRAO-DiFX-1.1/setup_difx`

This script contains the following:

```
export PATH=/home/swc/NRAO-DiFX-1.1/bin:/users/difx/bin:/bin:/usr/bin
export LD_LIBRARY_PATH=/home/swc/NRAO-DiFX-1.1/lib:/home/swc/NRAO-DiFX-1.1/lib/intel
export PKG_CONFIG_PATH=/home/swc/NRAO-DiFX-1.1/lib/pkgconfig
export JOB_ROOT=/home/swc/difx/projects
export TESTS=/home/swc/difx/tests
export MARK5_DIR_PATH=/home/swc/difx/directories
export CALC_SERVER=swc000
export GAIN_CURVE_PATH=/home/swc/difx/gaincurves
export IPPROOT=/opt/intel/ipp/5.3.3.075/ia32
export DIFX_MACHINES=/home/swc/difx/machines.difx
export DIFX_HEAD_NODE=swc000
export DIFX_ARCHIVE_ROOT=/home/swc/difx/archive
export MANPATH=/usr/share/man:/opt/local/man:/home/swc/NRAO-DiFX-1.1/share/man
export DIFX_VERSION=NRAO-DIFX-1.1
export DIFX_GROUP_ID=vlba_difx
alias src='pushd /home/swc/NRAO-DiFX-1.1/src'
echo "NRAO-DiFX version 1.1 is selected"
```

The "difx" account is set up to execute this script upon login. Note that the settings here are useful for both compilation of the various NRAO-DiFX components as well as using them. Each installed version of NRAO-DiFX will have its own setup file like this. Selecting which version is to be used is a simple as running the correct setup file. To change to the development version:

`. /home/swc/NRAO-DiFX-trunk/setup_difx`

It is highly recommended that one set the `DIFX_VERSION` environment variable and make sure that for each installed version of NRAO-DiFX this is set differently. It may also be desirable to customize this for your correlator. For example, one may set it to `USNO-NRAO-DIFX-1.1` . This string will be stored in intermediate files and the output FITS files and will be able to identify more exactly where the data were correlated.

5 Sharing resources with hardware correlator (VLBA Specific)

The Mark5 units attached to the VLBA correlator are used in three different ways for correlation: 1. An installed module can be played through the VLBA hardware correlator, 2. The CPU(s) inside the Mark5 unit can be used for software correlator processing, and 3. An installed module can be played into the software correlator. Functions 1 and 2 can be done simultaneously without conflict. Likewise, functions 2 and 3 can also be done simultaneously without conflict. Functions 1 and 3 cannot be done at the same time. In fact, the software correlator will refuse to use Mark5 units that are currently "Online" with regard to the hardware correlator. The Mark5 units need to be explicitly set to playback on a particular correlator. See §§5.1 & 5.2 for instructions. These two functions cleanly stop and start, respectively, the `Mark5A` program that is needed for the hardware correlator. It is only the playback of the Mark5 modules that has any chance of conflict between the software and hardware correlators. The hardware correlator can be correlating, idle, in standby, or completely off during software correlation.

5.1 Taking a Mark5 unit

Two actions must be taken to safely transfer a Mark5 unit from serving the hardware to serving the software correlator. Here "safely" implies no loss of data and no need for an unnecessary reboot; if done properly, no reboots of any computer should be needed. First the Mark5 unit must be taken Offline from the hardware correlator `SYSDISCS` screen. Multiple units can be taken Offline simultaneously by highlighting multiple units. Like usual, units currently assigned to a job cannot be taken offline. Second, the Mark5A program must be stopped on the Mark5 unit(s). This can be done with the `mk5take` program (§8.20). For example, to "take" units 8, 9 and 11, one would issue the following command from the shell prompt

```
mk5take 08 09 11
```

Any number of units can be taken offline at once with `mk5take`. Note — if a Mark5 unit is rebooted, `mk5take` will need to be run on that unit again once it comes to life as the `Mark5A` will be automatically started on each reboot (except for units fx01 through fx03 which never run the `Mark5A` program). You may need to wait 15 seconds or so after reboot is complete for this to work.

5.2 Returning a Mark5 unit

Once a Mark5 unit is again needed for use with the hardware correlator, it must be returned. It is important to make sure the software correlator is not using the particular Mark5 unit for playback before continuing. First the Mark5A program must be restarted, which can be done with the `mk5return` program (§8.19), e.g.,

```
mk5return 08 09 11
```

Again, any number of Mark5 units can be returned in one call. After a few seconds the effectiveness of this can be tested with the `mk5status program`; all units expected to be used for the hardware correlator should be in the READY state. Once this is the case, the units can once again be put Online with the `SYSDISCS` screen. No reboots should be needed if things go well.

6 Running DiFX at the VLBA correlator

In this section are instructions for using the NRAO adapted DiFX to correlate data that has already been prepared for correlation by the VLBA hardware correlator. Some aspects of this section may still apply to correlation of other data. Currently no graphical user interface exists, so these instructions are command line only; these instructions will change when the operator interface is complete. This section assumes that the software is properly installed and environment variables are set appropriately for use which will be the case for correlator operators. While there are several steps in performing the software correlation, nothing is too complicated. The procedure below does not go into any detail about what is actually happening at each step, please find details in other sections of this manual. Only steps 9 though 12 below require the correlator-specific hardware so the pre- and post- correlation steps could all be run by the analysts rather than operators. In the example that follows, project BM264 segment G will be correlated:

1. Log into the head node as user: `ssh -l difx $DIFX_HEAD_NODE`
2. Change directories to the job root directory: `cd $JOB_ROOT`
3. Copy the job scripts and monitor data over: `getjobs bm264g` (§8.9)
4. Enter the project directory: `cd bm264g`
5. At this time it would be convenient to open another terminal, log into `$DIFX_HEAD_NODE` as difx, and change to the `$JOB_ROOT` directory so that it is easy to monitor the progress in one window while running the correlation in the other.
6. Generate the DiFX input files: `job2difx *.fx` (§8.10).
7. Look over the list of jobs to be done: `joblist` (§8.12).
8. Determine which modules are needed, either by looking at the `.fx` files or with the use of `jobdisks` (§8.11). Install these modules into Mark5 units, making sure not to put two modules that are needed by the same job into the same unit.
9. Offline the units using the hardware correlator `SYSDISCS` screen and "take" them (§5.1). Note that if no hardware correlation is to occur it is okay to "take" all of the Mark5 units for convenience.
10. At this point one could start the correlation of all the jobs with a single command: `startdifx *.input` (§8.22), but it is recommended to start just one job first to make sure things are working: e.g. `startdifx job1240.000.input` .
11. When output stops and the command prompt is given back, the processing is either done or something failed. If a job is encountered that requires a module that is not mounted, processing will stop with a warning that the module could not be found. At this point, you should load the desired module and again type `startdifx *.input` to continue processing.
12. When done, return the Mark5 units if they are needed by the hardware correlator (§5.2).
13. Generate fits files for all correlated jobs: `difx2fits -d` (§8.5).

If a job is encountered and a required module for that job is not installed in a Mark5 unit that has been "taken", the correlation process will stop. The needed module will be listed in an error report, or can be identified using the `jobdisks` (§8.11) program. Once the module has been installed, start at step 10 again and the correlation should proceed from where it left off.

6.1 Monitoring the correlation process

Most of the time in correlation is spent running `mpifxcorr`. This program sends information to the terminal window in which it runs. As long as output continues at least every 30 seconds, things are probably OK. Error messages are for the most part fairly obvious; if unusual output is seen and `mpifxcorr` stops, it is often a good idea to scroll up several pages as the root of the problem may have scrolled of the page, especially when using many nodes for correlation.

The easiest way to monitor the progress of correlation is to periodically run `jobstatus` (§8.13) in a different window. Make sure the current working directory of the shell is the project directory of the project being correlated. The use of `cpumon` (§8.4), `mk5mon` (§8.18), and `errormon` (§8.7) will provide additional feedback. The Difx Operator Interface will supersede these tools.

6.2 Common failure modes

Here is a list of some common failure modes and some hints to identifying and solving them.

6.2.1 Module moved

If a required module has been removed or moved since `genmachines` has run, `mpifxcorr` will not be able to correlate. In this case DiFX will fail, spitting out a substantial amount of debug information. You can try again by running `genmachines` *baseFilename*`.input` to force the recreation of the `.machines` file. If this program fails, it will report an error that may aid in diagnostics. Note that this scenario will not happen if `startdifx` (§8.22) is used to run the correlator.

6.2.2 Mark5 unit hung

Unfortunately, there are still some instabilities with Mark5 units that result in various kinds of hangs; some units appear more sensitive than others. Often a failed Mark5 can be identified with the last few lines of error messages output from `mpifxcorr`. To verify, first attempt to `ssh` into that unit. If that is successful, try running `mk5list` (§8.17) on that unit to make sure the driver has not become corrupt. If logging into the mark5 unit works but `mk5list` fails, try resetting the Streamtor card with:

`mk5control reset` *unitNumber*

where *unitNumber* is, for example, 07 for mark5fx07 or 23 for mark5fx23. The Mark5 state shown in `mk5mon` should change to "Resetting". If it does not, then it is likely a reboot is needed.

If none of the above works, try rebooting the particular Mark5 unit and starting over. Note: as currently configured, a Mark5 unit will restart the `Mark5A` upon boot, so you will need to use `mk5take` to stop that before attempting software correlation on that unit again. Make sure to give the Mark5 unit enough time to initialize the `Mark5A` program before running `mk5take` (i.e., wait for module lights to cycle).

A possibly more reliable way to identify a hung Mark5 unit is to start a new instance of `mk5mon` (§8.18) in a terminal and issue the following command:

```
mk5control getvsn mark5
```

A hung Mark5 will not show up in the list of units.

6.3 Correlating data files

The operating instructions up to this point have focused on correlation directly off Mark5 modules. Correlation off files is also supported, as is a mixed mode where files and modules are correlated together. The scripts described in this document don't (to date) make correlation of files easy, but it is possible to do so by hand editing files. It is expected that enhancements to the scripts will make correlation from files much easier in NRAO-DiFX version 1.1. Two files will need manipulation: `.input` and `.machines`. In the `.input` file, every entry in the `DATASTREAM` table that corresponds to a disk file needs the `DATA SOURCE` value changed from `MODULE` to `FILE`. The `.machines` file will likely have to be constructed completely by hand. See §9.15 for a detailed description of the format of that file. Note that it is no longer necessary for the data files to be visible to all cluster computers – they can reside on local drives that are not exported, including USB or Firewire drives, but this requires that the datastream nodes listed in the `.machines` file be in the order in which the antennas are listed in the `.input` file.
Note: you must use the `-n` option to `startdifx` when starting the correlation or the hand-edited `.machines` file will be overwritten.

7 Some comments on channels

This section discusses the accountability of channel identification through the entire NRAO-DiFX system. While much of this discussion will not be of use outside NRAO, the terminology discussed here might help explain other portions of this document. The subject of this section is baseband channels, not individual frequency channels that the correlator produces from the baseband channels.

Baseband channels are individual digital data streams containing a time-series of sampled voltages representing data from a particular portion of the spectrum from one polarization. Each baseband channel is assigned a recorder channel number. For a given baseband data format (i.e., VLBA, Mark4, Mark5B, ...) a particular recorder channel number is assigned to a fixed number of tracks or bitstreams. This mapping is contained in the `track` row in the `format` table of the `.fx` job script and can be different for each antenna. This mapping is also reflected in the `.input` file in the `datastream` table.

DiFX correlates baseband channels from multiple antennas to produce visibilities. From each correlated baseline, one or two basebands from one telescope will be correlated against one or two basebands of another, resulting in up to four products for a particular sub-band. This is to allow full polarization correlation. Each sub-band (called an IF in AIPS) is given a sub-band number; in general 1 or 2 recorder channels map to each sub-band. Note that an observation can simultaneously observe some sub-bands consisting of only one baseband and some with two basebands. In cases such as this the matrix containing the visibility products on a particular baseline will be large enough in each dimension (i.e, polarization product, sub-band) to contain all of the results, even if this consumes more storage than necessary; flags are written that invalidate portions of the visibility matrix that are not produced by the correlator.

8 Reference guide to programs and utilities

This section has usage information for the numerous programs and scripts used in the NRAO-DiFX system. Basic help information for most or all of these programs can be gotten by typing the program name with either no command line arguments or with a `-h` option. In the usage descriptions below, arguments in square brackets [ ] are optional and can often include multiple different parameters. Cases where 1 or more arguments of a certain type (such as files) can be passed to the program, the usage instructions will look like *arg*1 [$\cdots$ *arg*N], with the implication that N arguments of this type were passed. In cases where 0 arguments of that type is also allowed, that first argument will also be in square brackets. If it is not obvious from the program name, the software package containing the program follows the section header.

Note that several VLBA specific programs are discussed in this manual that are not documented here, such as `cjobgen`, `tsm`, and `plotbp`. These are preexisting programs that may be documented elsewhere and are less likely to be useful outside VLBA operations.

8.1 calcif (package : job2difx) ***deprecated – do not use***

Program `calcif` takes one or more `.calc` file created by `job2difx` and computes an appropriate delay model (and related quantities) for each for use in correlation. It connects via RPC to a running copy of `CalcServer` which must be running on a computer called `$CALC_SERVER`, or on the specified computer if the `-s` option is used. If the output files (specified in the `.calc` file) exist and are current (have newer modification times than the `.calc` file, then the files will not be recreated unless the force option is used. Note that the output files are first assembled in `/tmp` and moved to the destination directory when they are complete in order to avoid incomplete files; old versions of the output files are removed before this process begins.

Usage: `calcif` [*options*] { `-a` | *calcFile1* [*calcFile2* [$\cdots$]] }

options can be:

- `-h` or `--help` : print usage information and exit
- `-y` or `--yes` : needed for `calcif` to actually run 1.1
- `-a` or `--all` : run on all `.calc` files found in the current directory 1.1
- `-v` or `--verbose` : print more verbose logging/debug info
- `-f` or `--force` : rerun even if output files exist and are current 1.1
- `-s` *server* or `--server` *server* : connect to *server*, not `$CALC_SERVER`.

calcFile is a `.calc` file (§9.14), such as one generated by `job2difx` (§8.10).

Example 1: `calcif job1420.000.calc job1421.000.calc` 1.1

Example 2: `calcif -s kepler job1420.000.calc`

Example 3: `calcif -a` 1.1

The new option `-y` or `--yes` is in place to encourage users to use `calcif2` rather than `calcif`, but to retain the possibility of using the later in cases where it is desired. It is expected that `calcif` may cease to exist in future NRAO-DiFX releases. If so, `calcif2` may be renamed `calcif`.

Three files are produced by `calcif`:

1. `.uvw` Projected baseline vectors (§9.25)
2. `.delay` Geometric delays (§9.9)
3. `.rate` Time derivatives of geometric delays and atmospheric delays (§9.20)

The first two of these are required for correlation and construction of `FITS` files (§9.12). The later is required if full model accountability is desired.

8.2 calcif2 (package : job2difx) 1.1

Program `calcif2` is intended to completely replace `calcif`. It is used in the same way as `calcif` and boasts some important improvements. Instead of calling CALC for every tabulated model row, `calcif2` computes a 5th degree polynomial every 120 seconds (typically), very closely resembling the delay model generation used at the VLBA hardware correlator. These polynomials are then evaluated at each model point. This results in a tremendous speedup at negligible loss of accuracy. By default `calcif2` will call CALC three times for each model point and calculates more accurate u, v, w coordinates from delay measurements made over a small patch of the sky:

$$(u, v, w) = \left(c\frac{d\tau}{dl}, c\frac{d\tau}{dm}, c\tau \right) \tag{1}$$

where l, m are angular coordinates (in radians) relative to the delay center on the sky, τ is the delay at the delay center and c is the speed of light.

It connects via Remote Procedure Call (RPC) to a running copy of `CalcServer` which must be running on a computer called `$CALC_SERVER`, or on the specified computer if the `-s` option is used. If the output files (specified in the `.calc` file) exist and are current (have newer modification times than the `.calc` file, then the files will not be recreated unless the force option is used.

Usage: `calcif2` [*options*] { `-a` | *calcFile1* [*calcFile2* [· · ·]] }

options can be:

`-h` or `--help` : print usage information and exit

`-a` or `--all` : run on all `.calc` files found in the current directory

`-v` or `--verbose` : print more verbose logging/debug info

`-q` or `--quiet` : print less verbose logging/debug info

`-f` or `--force` : rerun even if output files exist and are current

`-n` or `--noaber` : don't perform aberration corrections

`-z` or `--allow-neg-delay` : don't zero delays that are negative (i.e. shadowed)

`-s` *server* or `--server` *server* : connect to *server*, not `$CALC_SERVER`.

`-o` *order* or `--order` *order* : make polynomials with *order* + 1 terms (default 5).

`-i` *int* or `---interval` *int* : make a polynomial every *int* seconds (default 120).

`--override-version` ignore difx version clashes *calcFile* is a `.calc` file (§9.14), such as one generated by `job2difx` (§8.10).

Example 1: `calcif2 job1420.000.calc job1421.000.calc`

Example 2: `calcif2 -s kepler job1420.000.calc`

Example 3: `calcif2 -a -i 60`

Four files are produced by `calcif2`:

1. `.uvw` Projected baseline vectors (§9.25)
2. `.delay` Geometric delays (§9.9)
3. `.rate` Time derivatives of geometric delays and atmospheric delays (§9.20)
4. `.im` Polynomial interferometer model file, to be used by `difx2fits` (§9.13)

The first two of these are required for correlation and construction of `FITS` files (§9.12). The third is required (unless the fourth is present) in addition if full model accountability is desired. If the fourth file is present, its contents will override those of the first three when making `FITS` files.

8.3 CalcServer

Program `CalcServer` contains the Goddard Space Flight Center CALC package version 9.1, used to compute geometric delay models for VLBI applications. It is a repackaged version of the same source code that is used to compute models on the VLBA correlator. It is configured to run as a server. All of its interactions are via RPC calls from other programs, such as `calcif`, which could be running on the same or different computer. This program only needs to be started once on a given machine using the `startCalcServer` script. It should probably be set to start automatically upon boot of the machine on which `CalcServer` runs. Environment variable `$CALC_SERVER` should be set to the name of the computer on which `CalcServer` is running.

Start: `startCalcServer`

Test: `checkCalcServer $CALC_SERVER`

Stop: `killall CalcServer`

Note that `CalcServer` must be installed (with `make install`) to be usable as the paths for various files are permanently set in the executables at compile time. At this time it seems `CalcServer` cannot be compiled for 64-bit machines.

8.4 cpumon (package : difxmessage) 1.1

Program `cpumon` is a program that listens for `difxLoad` messages multicast from the Mark5 units and displays the information; updating the display as new messages are received.

Usage: `cpumon`

Make sure the terminal is at least 80 characters wide and is at least as tall as there are computers that may transmit information. To quit, use ctrl-C, and you may need to enter “reset” at the command line to get your prompt back. The columns displayed are:

1. Computer name
2. CPU load averaged over 10 seconds

3. Memory usage / Total memory
4. Network receive rate (Mbps)
5. Network transmit rate (Mbps)

8.5 difx2fits

Program `difx2fits` creates a FITS output file from the SWIN format visibilities created by `mpifxcorr` and several other files carrying information about the observation. When run, `difx2fits` requires the following files to be present:

1. *baseFilename*`.difx/`
2. *baseFilename*`.input`
3. *baseFilename*`.uvw`
4. *baseFilename*`.delay`

This minimal requirement means that `difx2fits` can be used even outside the NRAO DiFX environment, though it is possible that certain combinations of different configurations within the `.input` file will not be properly supported at this time. Several other files are optional and are typically used to populate calibration and ancillary tables:

1. *baseFilename*`.calc`
2. *baseFilename*`.rates`
3. *baseFilename*`.im`
4. *baseFilename*`.flag`
5. `flags`
6. `pcal`
7. `tsys`
8. `weather`
9. `$GAIN_CURVE_PATH/`

With the exception of the gain curve files, all the input files to `difx2fits` are expected to be in the current working directory. As the visibility file (`.difx`) is read, any records that are all zero are omitted. The number count of these dropped records is reported as “invalid records” when `difx2fits` finishes writing the **UV** table. With `difx2fits` versions since 2.0 (which is included as part of NRAO-DiFX 1.1) multiple correlator output files can be combined into a single destination `FITS` file; this feature is still new, so please check the results carefully!

Usage: `difx2fits` [*options*] { `-d` | *baseFilename1* [$\cdots$*baseFilenameN*] [*outFile*] }

options can be:

`-a` *chanavg* or `--average` *chanavg* : average *chanavg* spectral channels

`-b` *chan* or `--beginchan` *chan* : convert channels starting at zero-based channel *chan*

`-h` or `--help` : print usage information and exit

`-n` or `--no-model` : don’t write model (ML) table

`-o` *nchan* or `--outchans` *nchan* : write a total of *nchan* channels to FITS

`-s` *scale* or `--scale` *scale* : scale visibility data by *scale*

`-v` or *–verbose* : increase verbosity of output; use twice or thrice to get even more

`-d` or *–difx* : run on all `.difx` files found in the directory 1.1

`-k` or *–keep-order* : don't sort the antennas by name 1.1

`-1` or *–dont-combine* : make a separate FITS file for each input job 1.1

`-x` or *–dont-sniff* : don't generate sniffer output files 1.1

`--override-version` ignore difx version clashes 1.1

baseFilename is the prefix of the jobfile to convert; it is OK to use the .difx filename instead

outFile is the name of the `FITS` file to produce; if not provided one will be made based on the project code

Example 1: `difx2fits job9020.000 9020.FITS`

Example 2: `difx2fits -v -v -d`

Unless disabled with the `--dont-sniff` or `-x` flag, four "sniffer" output files (`.acb`, `.apd`, `.wts` and `.xcb`) will be written for each `.FITS` file produced. These files are used by `difxsniff` and its associated programs to produce data plots that are used to assess data quality.

8.6 difxsniff (package : job2difx)

Program `difxsniff` is a reimplementation of the analyst's program `sniff.pd` to be more appropriate for software correlation where the sniffer data is generated at the same time as the FITS files. It uses the same underlying set of plotting programs (`plotwt`, `plotbp`, and `plotapd`) as `sniff.pd` did. It should be run in a project directory as it will create a subdirectory (if not existing already) which by default is called `sniffer/`*refant* within the current directory. All files created by `difxsniff` will be placed in this directory, overwriting existing files with the same filenames. Unlike `sniff.pd`, `difxsniff` is a purely non-interactive command line program. Note that although `.FITS` files are provided to `difxsniff`, it is the associated files ending in `.apd`, `.wts`, `.acb` and `.xcb` that are actually read.

Usage: `difxsniff` [*options*] *refants* *FITS*1 [$\cdots$ *FITS*N]

options can be:

`-h` or `--help` : print usage information and exit

refants is a list reference antennas, separated by spaces

FITS is a FITS file created by `difx2fits`; multiple FITS files can be specified together

Example 1: `difxsniff LA *.FITS`

Example 3: `difxsniff NL FD *.FITS`

8.7 errormon (package : difxmessage) 1.1

Program `errormon` listens for multicast messages of the `difxError` variety and simply prints their contents to the terminal.

Usage: `errormon`

8.8 genmachines (package : job2difx)

Program `genmachines` uses the information in a `.input` file and a file containing information about the members of the compute cluster (such as the file pointed to by `$DIFX_MACHINES`) to produce a `.machines` file (§9.15) needed by `mpifxcorr`. Note that `genmachines` is not intended to be run by hand anymore as `startdifx` does this, if necessary. If playback directly off Mark5 units is to be done, `genmachines` will send a multicast request to all Mark5 units on the correlator requesting an inventory of loaded Mark5 modules. The `mk5daemon` process on each unit will respond with another multicast message containing the loaded modules and the status of the unit, i.e., whether busy or available to be used. This information is collected by genmachines which will look for availability of all the modules and detect conflicts (i.e., two needed modules loaded in the same unit). If all needed modules are found and enough resources remain for the computations, a `.machines` file and a `.threads` file are written. Note that the `.machines` file contains a certain number of comment lines so that the use of Unix command `wc -l` can be used to determine exactly how many processes will be started. It is suggested to run this program immediately before starting the software correlator to minimize the chance that the Mark5 units change their status or that information about the modules whereabouts becomes stale; it is thus discouraged to run with `*.input`.

Usage: `genmachines` [*options*] *input*1 [··· *input*N]

options can be:

`-h` or `--help` : print usage information and exit

`-v` or `--verbose` : be more verbose

`-o` or `--overheadcores` *ohc* : leave at least *ohc* on each compute node unscheduled 1.1

`-m` *file* or `--machinesfile` *file* : use *file* instead of `$DIFX_MACHINES`

`-n` or `--no-threads` : don't write a `.threads` file.

input is a `.input` file; multiple files can be specified, each producing its own `.machinesfile`

8.9 getjobs (package : job2difx)

Program `getjobs` can be used to create a new software correlation project directory and copy `.fx` files (job scripts) and a `cal.vlba` file from `/home/vlbiobs` into the new project directory. If the `cal.vlba` file is found, it is `gunzip`ed if needed and run through `vlog` (§8.24).

Usage: `getjobs` [*options*] *project* [*rootDir*]

options can be:

`-h` or `--help` : print usage information and exit

`-f` or `--force` : allow overwrite of files

project is the full name of a project, in lower case, including segment

rootDir is the an optional parameter specifying the directory into which the new project directory will be created. Default is the current working directory if none is provided.

Example 1: `getjobs bw088r`

Example 2: `getjobs bg160 $JOB_ROOT`

8.10 job2difx

Program `job2difx` takes one or more `.fx` job script files (created by `cjobgen`) and creates one or more `.input` files. For each `.input` file created, a matching `.calc` file is also generated. In almost all cases, this program will be called with `*.fx` as input, unless certain jobs (such as clock searches or pilots) are to be ignored.

A `.fx` job file can be split into multiple `.input` files for a couple reasons. The most general reason is that of an incompatible change of subarrays, a common occurence in geodetic observations. Currently most of the infrastructure set up around NRAO's DiFX implementation can only support one subarray; projects utilizing multiple (possibly changing) subarrays are handled at this stage by dividing those jobs into multiple smaller ones each containing only one subarray. An integer index are used to identify amongst the possible multitude of `.input` files created from one input file: the *subjob*. The *subjob* index starts at 0 and increments each time subarray membership changes in an incompatible manner or when certain frequency setups change. The new *base filename* is created by starting with the name of the `.fx` file, removing the `.fx` extension, and adding *.subjob.subarray*. Each job-specific file (such as `.calc`, `.input`, `.difx`, `.uvw`, ...) is this base filename with the appropriate extension appended. Note – NRAO-DiFX 1.0 used two indexes (subjob and subarray); now one index incorporates both of these functions.

Usage: `job2difx` [*options*] *jobFile*1 [$\cdots$ *jobFile*N]

options can be:

`-h` or `--help` : print usage information and exit

`-s` or `--single-config` : split jobs at each configuration change 1.1

`-p` or `--multi-pass` : turn each separate configurations into interleaved jobs 1.1

`-m` or `--multi-config` : attempt to make file with multiple configurations; FLAKEY! 1.1

`-v` or `--verbose` : increase verbosity 1.1

`A=`*antList* : use only antennas in comma separated list *antList*

`nchan=`*nChan* : override number of output channels

`fftsize=`*nFFT* : override FFT size to use

`tint=`*tInt* : override integration time (in seconds)

`minsub=`*minSubarraySize* : set minimum subarray size (default 3)

`spectrunc=`*specTrunc* : set amount of spectral truncation to perform before decimation; legal values are 1, 2, 4, 8 & 16

jobFile is a `.fx` file; many can be supplied

Example 1: `job2difx *.fx`

Example 2: `job2difx A=FD,KP,MK *.fx`

Example 3: `job2difx tint=0.5 nchan=1024 *.fx`

Important note: in cases where multiple `.fx` files are made to cover the same time range due to limitations of the VLBA hardware correlator (i.e., processing of 512 Mbps recordings), these `.fx` files must be passed to `job2difx` together. In these cases, the output base filename is derived from the `.fx` file with the lowest job number.

8.11 jobdisks (package : job2difx)

Program `jobdisks` looks through job files to see which modules (disks) are needed for correlation. It can read through `.fx` files, as created by `cjobgen`, or through `.input` files, as used by `mpifxcorr`, though a mixture of the two is not allowed. There are two modes of operation. By default, a matrix of all modules for all stations is displayed, with a `--` symbol indicating that a particular station is not used in a particular job. An asterisk (`*`) indicates a module change. The second mode, instigated with command line argument `-c`, summarizes only module changes. Running without any arguments will cause `jobdisks` to look at job files within the current directory, prioritizing on `.input` files if any exist and falling back on `.fx` files otherwise. Listings for a subset of jobs can be made by specifying particular files.

Usage: `jobdisks` [*options*] [*file*1] · · · [*file*N]

options can be:

`-h` or `--help` : print usage information and exit

`-c` or `--changes` : print module changes only

file is a `.fx` or `.input` file; mixed types are not supported. Multiple input files may be supplied.

Example 1: `jobdisks`

Example 2: `jobdisks job1420*.input`

Example 3: `jobdisks *.fx`

Example 4: `jobdisks -c`

8.12 joblist (package : job2difx)

Program `joblist` prints useful information about DiFX correlator jobs to *stdout*. Six columns of output are produced:

1. Job file base filename
2. File indicator, showing a particular character for each one of the files associated with that job that is found within a pair of square brackets, [ ]:

   `c` `.calc` file (§9.8)
   `m` `.machines` file (§9.15)
   `t` `.threads` file (§9.21)
   `u` `.uvw` file (§9.25)
   `d` `.delay` file (§9.9)
   `r` `.rate` file (§9.20)
   `i` `.im` file (§9.13) 1.1
   `v` `.difx` file (§9.10)

3. Band code of first scan in file
4. Observation duration of correlation (in minutes)
5. Recording mode triplet; three integers(data rate(Mbps), number of baseband channels & quantization bits) separated by dashes
6. Comma separated list of antennas

One line is printed for each `.input` file found in the list of directories provided (or current directory if not listed).

Usage: `joblist` [*options*] [*dir*1] · · · [*dir*N]

options can be:

`-h` or `--help` : print usage information and exit

dir is a directory for which to print job information (default is current shell directory). Multiple directories can be specified.

Example 1: `joblist`

Example 2: `joblist $JOB_ROOT/*`

8.13 jobstatus (package : job2difx)

Program jobstatus lists the current correlation progress for each DiFX job in one or more directories. This program is normally run without any command line arguments from within the project directory. For each job, the base filename is listed with 5 or 6 additional columns of data. These columns are

1. Observation duration (minutes)
2. Record mode triplet (*Mpbs-nChan-nBit*)
3. Number of stations in job
4. Speed up factor (ratio of correlation time to observe time), or zero if correlation has not yet begun.
5. Percentage complete
6. Number of minutes remaining (only if Percentage complete isn't 0% or 100%)

Below these lines, five more lines containing information about the group of jobs as a whole is are presented. The contents of these lines are:

1. Total job time : Minutes of observe time in listed jobs
2. Fraction complete : Percentage in time through the entire project
3. Job time remaining : Minutes of observation left to be correlated
4. Wall time remaining : Minutes of real time needed to complete jobs
5. Average speedup : Ratio of total correlation time to run time, up to current point

Note that the speedup and time remaining values are estimates and don't include model calculation, conversion to FITS, and job startup time.

Usage: `jobstatus` [*options*] [*dir*1] · · · [*dir*N]

options can be:

`-h` or `--help` : print usage information and exit

dir is a directory for which to print job information (default is current shell directory). Multiple directories can be specified.

Example 1: `jobstatus`

Example 2: `jobstatus $JOB_ROOT/*`

8.14 mk5control (package : mk5daemon) 1.1

`mk5control` is a program that sends XML messages of type `DifxCommand` to the `mk5daemon` programs that run on the software correlator cluster members. This program is a superset of `mk5take` and `mk5return`, allowing any allowed command to be sent.

Usage: `mk5control` [*options*] *command unit*1 · · · *unit*N

options can be:

`-h` or `--help` : print usage information and exit.

command is the (non-case-sensitive) command to be executed — see list below.

unit is the number of a correlator mark5 unit, a range, `all` for all software correlator cluster members, `mark5` for all mark5 units, or `swc` for all software correlator compute nodes.

Example 1: `mk5control stopmark5a 07 08 09 11 14`

Example 2: `mk5control resetmark5 14-24`

Example 3: `mk5control startmark5a mark5`

The list of supported *command* types is below. All commands are not case sensitive.

- `GetVSN` Request a `Mark5Status` XML document to be multicast from the *unit*
- `ResetMark5` Execute `SSReset` and `ssopen` — cures many/most mark5 hangs
- `StartMark5A` Start the Mark5A program (see §8.19)
- `StopMark5A` Stop the Mark5A program (see §8.20)
- `Clear` Clear the stat of the Mark5 unit and get the VSNs, can be dangerous if other programs are currently using the StreamStor card
- `Reboot` Reboot the machine
- `Poweroff` Shut down the machine
- `StopMk5Daemon` Stop the `mk5daemon` program — you probably never need to do this
- `GetDir` Extract the directory from the modules in both banks and save to files in $MARK5_DIR_PATH
- `GetDirA` Same as above, but look only at bank A
- `GetDirB` Same as above, but look only at bank B
- `Test` Used in debugging — for developers only

8.15 mk5daemon (package : mk5daemon) 1.1

`mk5daemon` is a program that started automatically at boot time on all of the software correlator cluster nodes (not only the Mark5 units!) that performs a number of operations in support of the software correlator.
The functions that mk5daemon performs are:

- **Logging**

  All received multicast messages, significant internal functions, and interactions of the `Mark5A` program are logged to human readable log files. These log files are restarted at the beginning of each day. By default these log files are saved in `/tmp`.

- **Control of Mark5A**

  The `Mark5A` program (written by Haystack) is the principle program used to access the Mark5 systems at the VLBA stations and the hardware correlator. DiFX directly accesses the StreamStor card via a library level programming interface. Since only one program is allowed to do this (or face a crash of varying degree of seriousness), access to the StreamStor card must be carefully managed. One function of `mk5daemon` is to maintain knowledge of who "owns" the StreamStor card at a given time to prevent conflicts. The starting and stopping of the Mark5A program can be requested by two messages of type `DifxCommand` : `startmark5a` and `stopmark5a`. When these commands are received by `mk5daemon`, the requested action is taken unless StreamStor conflict is likely. This type of command and others can be sent to `mk5daemon` with the `mk5control` program (§8.14).

- **CPU, memory, and network monitoring**

  Every 10 seconds, `mk5daemon` looks in the `/proc` directory to get information about the CPU load, memory usage, and network traffic. These numbers are multicast in a `DifxLoad` message and logged.

- **Module VSN and state determination**

  Receipt of a multicast `getvsn` command will result in `mk5daemon` multicasting out a `Mark5Status` message containing information on the VSNs of the inserted modules as well as the state of the Mark5 unit. When `Mark5A` is running, a socket is opened to this program and the `bank_set?` query is issued, which returns the VSNs, regardless of the activity. When `Mark5A` is not running, `mk5daemon` either directly determines the VSNs through a StreamStor API library call if the Mark5 unit is idle, or doesn't respond if the Mark5 unit is busy. With each `Mark5Status` message that is multicast from `mk5daemon` the state of the Mark5 unit is included. See §10 for details on these XML messages.

Normally `mk5daemon` is started automatically, either by `/etc/rc.local` or by a script in `/etc/init.d` . The command line options supported are:

Usage: `mk5daemon` [*options*]

options can be:

`-h` or `--help` : print usage information and exit

`-l` *logPath* or `--log-path` *logPath* : put logs in directory *logPath*, not `/tmp`.

`-n` or `--no-mark5a` : don't run `Mark5A` upon start

Please be sure not to have multiple instances of `mk5daemon` running at any one time on any individual Mark5 or correlator unit!

8.16 mk5dir (package : mark5daemon) 1.1

Program `mk5dir` extracts the directory from a module. Normally one would not call this program directly but would use the `getdir` option of `mk5control`.

Usage: `mk5dir` [*options*] { *bank* — *VSN* }

options can be:

`-h` or `--help` : print usage information and exit

`-v` or `--verbose` : increase verbosity: print directory to screen

bank is one of `A`, `B` or `AB`

VSN is a valid 8-character VSN of a loaded module

8.17 mk5list

Program `mk5list` prints to the terminal the VSNs of the modules installed in a Mark5 unit. This program is largely deprecated at this point and is not normally used in operations but may be convenient for diagnostic purposes.

Usage: `mk5list`

8.18 mk5mon (package : difxmessage) 1.1

Program `mk5mon` is a program that listens for `mark5Status` messages multicast from the Mark5 units and displays the information; updating the display as new messages are received.

Usage: `mk5mon`

Make sure the terminal is at least 110 characters wide and is at least as tall as there are Mark5 units that may transmit information. To quit, use ctrl-C, and you may need to enter "reset" at the command line to get your prompt back. The columns being displayed are:

1. Mark5 unit name
2. VSN of module in Bank A
3. VSN of module in Bank B
4. State of the Mark5 unit
5. Playback rate, if playing, in Mbps
6. Playback position, in bytes from beginning of module
7. Scan number of data being played, if playing
8. Scan name of data being played, if playing

8.19 mk5return (package : mk5daemon)

Program `mk5return` sends a multicast `startmark5a` command to the `mk5daemon` program running on the specified Mark5 units to start a `Mark5A` process if one is not running. This should be done after use of the specified Mark5 units for software correlation and before attempting to "Online" or use said units for hardware correlation. See §5.2 for more information.

Usage: `mk5return` [*options*] *unit*1 ··· *unit*N

options can be:

`-h` or `--help` : print usage information and exit

unit is the number of a correlator mark5 unit, or `all` for all units.

Example 1: `mk5return 07 08 09 11 14`

Example 2: `mk5return 14-24` 1.1

Example 3: `mk5return all`

8.20 mk5take (package : mk5daemon)

Program `mk5take` sends a multicast `stopmark5a` command to the `mk5daemon` program running on the specified Mark5 units to stop any `Mark5A` process that is running. Note – before running this, make sure the Mark5 units to be "taken" are not in use by the hardware correlator and are "Offline". See §5.1 for more information.

Usage: `mk5take` [*options*] *unit*1 · · · *unit*N

options can be:

`-h` or `--help` : print usage information and exit

unit is the number of a correlator mark5 unit, a range, or `all` for all units.

Example 1: `mk5take 07 08 09 11 14`

Example 2: `mk5take 14-24` 1.1

Example 3: `mk5take all`

8.21 mpifxcorr

The core of the DiFX software correlator is the program called `mpifxcorr`. This program is uses parallel computing to make correlation practical on a cluster of ordinary computers. This program runs on all the machines listed in the `.machines` file that is passed to `mpirun` — the program that starts `mpifxcorr`. It should be initiated from the cluster head node from within the project directory. The usage line below is appropriate for use with OpenMPI (§11.1) and within the NRAO-DiFX context; other incantations may provide better results depending on the setup. See the OpenMPI documentation for more details.

Usage: `mpirun -np` *nProcess* `--bynode --hostfile` *otherOptions machinesFile* `mpfixcorr` *inputFile*

nProcess is the number of processes to start; found with `wc -l` *machineFile*

machinesFile the `.machineFile`

inputFile the `.input` to run; the full path to this file needs to be given, so prepending the file with `` `pwd`/ `` is typical

otherOptions can be any additional option to `mpirun`; `startdifx` uses the `--mca btl ûdapl,openib --mca mpi_yield_when_idle 1` to suppress some warning messages and be less aggressive on networking

Within the NRAO-DiFX framework, the user should never have to directly start `mpifxcorr` as this is done more simply with `startdifx` or via the Difx Operator Interface.

8.22 startdifx (package : job2difx) 1.1

Starting `mpifxcorr` generally requires a lengthy command, inspiring the creation of startdifx which vastly simplifies use of the DiFX correlator. In addition to spawning the `mpifxcorr` processes, `startdifx` can orchestrate some of the preparatory work (for example running `calcif` and `genmachines`) and optionally run `difx2fits` to create a `.FITS` file for each job. This program is meant to work within the NRAO-DiFX environment and would probably require modification to be useful in other situations.

Usage: `startdifx` [*options*] *input1* [*input2* · · ·]

options can be:

`-h` or `--help` : print usage information and exit

`-f` or `--force` : proceed on files even if correlator output already exists and is up to date

`-a` or `--automachines` : run `genmachines` only if no `.machines` file exits

`-g` or `--genmachines` : run `genmachines` unconditionally (default)

`-n` or `--nomachines` : don't run `genmachines`

`-d` or `--dont-calc` : don't run `calcif` even if needed – will skip file

`-F` or `--fits` : run difx2fits on output of each job separately

`--override-version` : ignore potential difx version conflicts

*input*N is a `.input` file, or its prefix

Example 1: `startdifx job1420.000.input`

Example 2: `startdifx -f -n job1420.000 job1421.000`

Example 3: `startdifx -F *.input`

8.23 stopmpifxcorr (package : mpifxcorr)

If software correlation is in progress and it is desired to stop it, it is best to gently stop it rather than killing it abruptly. In most circumstances this can be accomplished with `stopmpifxcorr`. This program must be run on the machine running the *manager* process of the software correlator. If multiple `mpifxcorr` processes are found running on a machine, `stopmpifxcorr` will not proceed unless the `-f` option is used.

Usage: `stopmpifxcorr` [*options*]

options can be:

`-h` or `--help` : print usage information and exit

`-f` or `--first-pid` : send stop message to the numerically first process ID found

`-q` or `--quite` : don't produce much output

8.24 vlog (package : job2difx)

Program `vlog` takes as input a calibration file (`cal.vlba`; §9.3). It is part of the job2difx package (§11.9). This file is parsed to produce four files containing formatted arrays that are convenient for use in the construction of FITS tables: `flag`, `pcal`, `tsys`, and `weather` (§§9.6-9.23). This program is named after AIPS task `vlog` that does nearly the same thing.

Usage: `vlog` *calFile* [*antennaList*]

calFile is the `cal.vlba` file produced by `tsm` to be processed.

antennaList is an optional comma-separated list of antennas to process. If omitted, all antennas with calibration data will be processed.

Running with no command line arguments will print usage information to the terminal and exit. Normally `vlog` will be run automatically when `getjobs` (§8.9) is used to copy jobs.

9 Description of various files

In the descriptions that follow, the locations of some files is given as `/home/vlbiobs`, meaning the directory `/home/vlbiobs/astronomy/`*mmmyy*/*project* or one of its subdirectories. Here *mmmyy* is the month and year of the project's observation (i.e., `jan08`) and project is the full project name, with segment, in lower case, such as `bw088n`. In what follows, the "software correlator project directory" (sometimes "project directory") refers to the directory from which software correlation is to proceed. This directory can be created by `getjobs` (§8.9). File names beginning with a period (e.g., `.acb`) represent file name extensions, typically (but not always) to job file bases, such as `job121.000` . Examples of many of the following file types for a particular VLBA correlator job are stashed at `http://www.aoc.nrao.edu/~wbrisken/NRAO-DiFX-1.1/` .

9.1 .acb 1.1

When generation of sniffer output files is not disabled, each `.FITS` file written by `difx2fits` will be accompanied by a corresponding `.acb` file. This file contains auto-correlation spectra for each antenna for each source. In order to minimize the output data size, spectra for the same source will only be repeated once per 15 minutes. The file contains many concatenated records. Each record has the spectra for all baseband channels for a particular antenna and has the following format. Note that no spaces are allowed within any field. Values in `typewriter` font without comments are explicit strings that are required.

Line(s)	Value	Units	Comments
1	`timerange:`		
	MJD	integer ≥ 1	MJD day number corresponding to line
	start time	string	e.g., `13h34m22.6s`
	stop time	string	e.g., `13h34m52.0s`
	`obscode:`		
	observe code	string	e.g., MT831
	`chans:`		
	n_{chan}	≥ 1	number of channels per baseband channel
	`x`		
	n_{BBC}	≥ 1	number of baseband channels
2	`source:`		
	source name	string	e.g., `0316+413`
	`bandw:`		
	bandwidth	MHz	baseband channel bandwidth
	`MHz`		
3 to 2+n_{BBC}	`bandfreq:`		
	frequency	GHz	band edge (SSLO) frequency of baseband channel
	`GHz polar:`		
	polarization	2 chars	e.g. `RR` or `LL`
	`side:`		
	sideband	`U` or `L`	for upper or lower sideband
	`bbchan:`		
	bbc	`0`	Currently not used but needed for conformity
3+n_{BBC} to 2+$n_{\text{BBC}}(n_{\text{chan}}+1)$	*antenna number*	≥ 1	antenna table index
	antenna name	string	
	channel number	≥ 1	$=$ chan $+$ (bbc $-$ 1) $\cdot\, n_{\text{chan}}$ for chan, bbc ≥ 1
	amplitude	≥ 0.0	

The above are repeated for each auto-correlation spectrum record. This file can be plotted directly with `plotbp` or handled more automatically with `difxsniff`.

9.2 .apd 1.1

When generation of sniffer output files is not disabled, each `.FITS` file written by `difx2fits` will be accompanied by a corresponding `.apd` file. This file contains fringe fit solutions typically every 30 seconds for the entire experiment. These solutions are not of calibration quality but are sufficient for use in evaluating the data quality.

The first line in the file is the observation code, e.g., `MT831` .

Each subsequent line has the same format with the following fields:

Key	Units/allowed values	Comments
MJD	integer ≥ 1	MJD day number corresponding to line
hour	≥ 0.0, < 24.0	hour within day
source number	integer ≥ 1	source table index
source name	string	name of source; no spaces allowed
ant1 number	integer ≥ 1	antenna table index for first antenna
ant2 number	integer ≥ 1	antenna table index for second antenna
ant1 name	string	name of antenna 1; no spaced allowed
ant2 name	string	name of antenna 2; no spaced allowed
$n_{\rm BBC}$	integer ≥ 1	number of baseband channels, $n_{\rm BBC}$
		The next four columns are repeated $n_{\rm BBC}$ times
delay	ns	the fringe fit delay
amplitude	≥ 0.0	the amplitude of fringe fit peak
phase	degrees	phase of fringe fit peak
rate	Hz	the fringe fit rate

9.3 cal.vlba

Monitor data that gets attached to FITS files is extracted by `tsm` into a file called *project*`cal.vlba` where *project* is the name of the project, i.e., `bg167` or `bc120a`. A single file contains the monitor data for all antennas for the duration of the project. The file is left in `/home/vlbiobs` and is compressed with `gzip` after some time to save disk space, resulting in additional file extension `.gz`. The program `getjobs` (§8.9) can be used to copy this file from its original location in `/home/vlbiobs` and uncompress the file if needed. A program called `vlog` (sec §8.24) reads this file and produces files called `flag`, `pcal`, `tsys`, and `weather` in the software correlator project directory.

9.4 $DIFX_MACHINES

Environment variable `DIFX_MACHINES` should point to a file containing a list of machines that are to be considered elements of the software correlator. Program `genmachines` (§8.8) uses this file and information within a `.input` file to populate the `.machines` file needed by `mpifxcorr`. Because usually only one node in a cluster has direct access to a particular Mark5 module (or data from that module), the ordering of computer names in the `.machines` file is important. Rows in the `$DIFX_MACHINES` file contain up to three items, the last one being optional. The first column is the name of the machine. The second column is the number of processes to schedule on that machine (typically the number of CPU cores). The third column is a 1 if the machine is a Mark5 unit and 0 otherwise. If this column is omitted, the machine will be assumed to be a Mark5 unit if the first 5 characters of the computer name are '`mark5`', and will be assumed not to be otherwise. Comments in this file begin with an octothorpe (#). Lines with fewer than two columns (after excision of comments) are ignored.

9.5 .flag

The program `job2difx` may write a `.flag` file for each `.input` file it creates. This file is used by `difx2fits` to exclude nonsense baselines that might have been correlated. This can occur when multiple subarrays are coming and going. The format of this text file is as follows. The first line contains an integer, n — the number of flag lines to follow. The next n lines each have three numbers: MJD_1, MJD_2 and ant. The first two floating point numbers determine the time range of the flag in Modified Julian Days. The last integer number is the antenna number to flag — a zero-based index corresponding to the `TELESCOPE` table of the corresponding `.input` file.

9.6 flag

A file called `flag` is created when program `vlog` operates on the `cal.vlba` file. This file contains lists of antenna-based flags generated by the on-line system that should be applied to the visibility data. This file contains two kinds of lines. Comment lines begin with an octothorpe (#) and contain no vital information. Flag lines always consist of exactly 5 fields:

1. Station name abbreviation, e.g., `LA`
2. Beginning of flagged period (day of year, including fractional portion)
3. End of flagged period (day of year, including fractional portion)
4. Record channel affected; -1 for all record channels, otherwise a zero-based index.
5. Reason for flag, enclosed in single quotes, truncated to 24 characters

The flag rows are sorted first by antenna, and then start time.

9.7 .fx

The program `cjobgen` is used to create job scripts (with filename extension `.fx`) for use with the VLBA hardware correlator. Although the functionality of `cjobgen` will eventually be replaced, it is convenient to use the `.fx` files it creates in the interim as they contain all the information required to calculate a delay model and drive the software correlator. These files are converted by `job2difx` (§8.10) to produce `.input` files to control correlation and `.calc` files to drive the delay model generator. Note that there will not in general be a 1 to 1 relationship between `.fx` files and `.input` or `.calc` files due to combination of multiple passes and splitting due to frequency changes. The program `getjobs` (§8.9) can be used to copy these files from their original location in `/home/vlbiobs`.

9.8 .calc

The main use of the `.calc` file is to drive the geometric model calculations but this file also serves as a convenient place to store information that is contained in the `.fx` file but not in the `.input file` and is needed for `.FITS` file creation. In the NRAO-DiFX system, one `.calc` file is created by `job2difx` (§8.10) for each `.input` file. This file is read by `calcif` (or `calcif2`) (§§8.1&8.2) to produce a tabulated delay model, u, v, w values, and estimates of atmospheric delay contributions.

In brief, the parameters in this file that are relevant for correlation include time, locations and geometries of antennas, pointing of antennas (and hence delay centers) as a function of time and the Earth orientation parameters relevant for the correlator job in question. Additional parameters that are stuffed into this file include spectral averaging, project name, and information about sources such as calibration code and qualifiers. In the NRAO application of DiFX, source names are faked in the actual `.input` file in order to allow multiple different configurations for the same source. A parameter called *realname* accompanies each source name in the `.calc` file to correctly populate the source file in `.FITS` file creation.

The syntax of this file is similar to that of the `.input` file. The file consists entirely of key-value pairs separated by a colon. The value column is not constrained to start in column 21 as it is for the files used by `mpifxcorr`. There are five sections in the `.calc` file; these sections are not separated by any explicit mark in the file.

The first section contains values that are fixed for the entire experiment and at all antennas — all data in this section is scalar. In the following table, all numbers are assumed to be floating point unless further restricted. The keys and allowed values in this section are summarized below. Optional keys are identified with a $\star$.

Key	Units/allowed values	Comments
JOB ID	integer ≥ 1	taken from `.fx` file
$\star$ JOB START TIME	MJD + fraction	start time of original `.fx` file
$\star$ JOB STOP TIME	MJD + fraction	end time of original `.fx` file
OBSCODE	string	observation code assigned to project
$\star$ SESSION	short string	session suffix to OBSCODE, e.g., `A` or `BE`
$\star$ DIFX VERSION	string	version of correlator, e.g. `NRAO-DIFX-1.1`
$\star$ SUBJOB ID	integer ≥ 0	subjob id assigned by `job2difx` (§8.10)
$\star$ SUBARRAY ID	integer ≥ 0	subarray id assigned by `job2difx`
START MJD	MJD + fraction	start time of this subjob
START YEAR	integer	calendar year of START MJD
START MONTH	integer	calendar month of START MJD
START DAY	integer	day of calendar month of START MJD
START HOUR	integer	hour of START MJD
START MINUTE	integer	minute of START MJD
START SECOND	integer	second of START MJD
INCREMENT (SECS)	integer	seconds between computed model points (*inc*)
$\star$ SPECTRAL AVG	integer ≥ 1	number of channels to average in FITS creation
$\star$ START CHANNEL	integer ≥ 0	start channel number (before averaging)
$\star$ OUTPUT CHANNELS	integer ≥ 1	total number of channels to write to FITS
	$> 0.0, < 1.0$	fraction of total channels to write to FITS
$\star$ TAPER FUNCTION	string	currently only `UNIFORM` is supported
DELAY FILENAME	string	filename, including path, of `.delay` file to create
UVW FILENAME	string	filename, including path, of `.uvw` file to create
RATE FILENAME	string	filename, including path, of `.rate` file to create
IM FILENAME	string	filename, including path, of `.im` file to create

The second section contains antenna(telescope) specific information. After an initial parameter defining the number of telescopes, there are *nTelescope* sections (one for each antenna), each with the following six parameters. Lowercase t in the table below is used to indicate the telescope index, an integer ranging from 0 to *nTelescope* - 1. Note that in cases where units are provided under the Key column, these units are actually part of the key.

Key	Units/allowed values	Comments
NUM TELESCOPES	integer ≥ 1	number of telescopes (*nTelescope*).
		The rows below are duplicated *nTelescope* times.
TELESCOPE t NAME	string	upper case antenna name abbreviation
TELESCOPE t MOUNT	string	the mount type: altz, equa, xyew, or xyns
TELESCOPE t OFFSET (m)	meters	axis offset in meters
TELESCOPE t X (m)	meters	X geocentric coordinate of antenna at date
TELESCOPE t Y (m)	meters	Y geocentric coordinate of antenna at date
TELESCOPE t Z (m)	meters	Z geocentric coordinate of antenna at date
$\star$ TELESCOPE t SHELF	string	shelf location of module to correlate 1.1

Note that the antenna locations are currently taken from the `.fx` job script written by `cjoggen` and are valid for the date of observation.

The third section contains scan specific information. Except for one initial line specifying the number of scans, *nScan*, this section is composed of nine parameters per scan. Each parameter is indexed by s which ranges from 0 to *nScan* - 1.

Key	Units/allowed values	Comments
NUM SCANS	integer ≥ 1	number of scans (*nScan*).
		The rows below are duplicated *nScan* times.
SCAN s POINTS	≥ 1	duration of scan in units of *inc*
SCAN s START PT	integer ≥ 0	start time of scan in units of *inc* since MJD START
SCAN s SRC NAME	string	systematic name to match that used in `.input file`
SCAN s REAL NAME	string	name to use for source in the FITS output file
SCAN s SRC RA	radians	J2000 right ascension
SCAN s SRC DEC	radians	J2000 declination
SCAN s CALCODE	string	usually uppercase letters
SCAN s QUAL	integer ≥ 0	source qualifier

The fourth section contains Earth orientation parameters (EOP). Except for one initial line specifying the number of days of EOPs, *nEOP*, this section is composed of five parameters per day of sampled EOP values. Each parameter is indexed by e which ranges from 0 to *nEOP* - 1.

Key	Units/allowed values	Comments
NUM EOP	integer ≥ 1	number of tabulated EOP values (*nEOP*)
		The rows below are duplicated *nEOP* times.
EOP e TIME (MJD)	MJD + fraction	time of sample; fraction almost always zero
EOP e TAI_UTC (sec)	integer seconds	leap seconds accrued at time of job start
EOP e UT1_UTC (sec)	seconds	UT1 - UTC
EOP e XPOLE (arcsec)	arc seconds	X coordinate of polar offset
EOP e YPOLE (arcsec)	arc seconds	Y coordinate of polar offset

The final (completely optional) section has a table for positions and velocites of spacecraft. Each spacecraft is indexed by s and each row thereof by r.

Key	Units/allowed values	Comments
⋆ NUM SPACECRAFT	integer ≥ 0	number of spacecraft (*nSpacecraft*)
		Everything below is duplicated *nSpacecraft* times.
SPACECRAFT s NAME	string	name of spacecraft
SPACECRAFT s ROWS	integer ≥ 1	number of data rows, $nRow_s$ for spacecraft s
		The row below is repeated $nRow_s$ times.
SPACECRAFT s ROW r	7 numbers	tabulated data — see below

Each data vector of data consists of seven double precision values: time (mjd), x, y, and z (meters), and $\dot{x}$, $\dot{y}$, and $\dot{z}$ (meters per second). These values should be separated by spaces.

9.9 .delay

The `.delay` files contain tabulated interferometer model delays for each antenna for an entire DiFX job. This file type is typically produced by `calcif2` within NRAO-DiFX. Note that the values of the delays in this file have the opposite sign as compared to those generated by CALC and those stored in `.FITS` files, that is, a more positive delay implies "closer to the source"; negative delays are behind the Earth center, and hence for ground-based antennas are below the horizon.

The file consists entirely of key-value pairs separated by a colon. There are two sections in the `.delay` file; these sections are not separated by any explicit mark in the file.

The first section contains values that are fixed for the entire experiment — all data in this section is scalar. In the following table, all numbers are assumed to be floating point unless further restricted. The keys and allowed values in this section are summarized below:

Key	Units/allowed values	Comments
START YEAR	integer	calendar year of START MJD
START MONTH	integer	calendar month of START MJD
START DAY	integer	day of calendar month of START MJD
START HOUR	integer	hour of START MJD
START MINUTE	integer	minute of START MJD
START SECOND	integer	second of START MJD
INCREMENT (SECS)	integer	seconds between computed model points (*inc*)
NUM TELESCOPES	integer	number of telescopes, *nTelescope* with delay data
		the following line is repeated *nTelescope* times
TELESCOPE t NAME:	string	t starts at 0

The second section contains the scan-based information. First is a line indicating the number of scans to follow. Then for each scan, numbered by s ranging from 0 to *nScan* - 1, there are 3 lines containing information about the scan, including the number of sampled points within that scan, *nPoint*. Finally there are $nPoint_s + 3$ lines containing the tabulated delays (in microseconds), numbered -1 through $nPoints_s + 1$, indexed with p. Note that this includes one sample before the start of the scan and at least one after the scan end, allowing for a quadratic interpolation across the entire scan. This information is summarized in the following table:

Key	Units/allowed values	Comments
NUM SCANS	integer ≥ 1	number of scans (*nScan*).
		The rows below are duplicated *nScan* times.
SCAN s POINTS	≥ 1	duration of scan in units of *inc* ($nPoints_s$)
SCAN s START PT	integer ≥ 0	start time of scan in units of *inc* since MJD START
SCAN s SRC NAME	string	systematic name to match that used in `.input file`
RELATIVE INC p	array; see below	$nPoint_s + 3$ of these lines per scan

Like for the `.rate` and `.uvw` files, the values reported in this file extend one full mode increment (*inc*) before and after the actual duration of the scan, and hence will overlap in time by $2\times$ *inc* with consecutive scans.

This file is typically produced by `calcif2`.

9.10 .difx

The SWIN format visibilities written by `mpifxcorr` are written to a directory with extension `.difx`. Typically there will be a single file in this directory, but it is possible that the output data will be split into multiple smaller files if the first output file gets too large or if correlation is continued from a point midway through correlation (feature yet to be implemented).

These files contain visibility data records. Each record contains the visibility spectrum for one polarization of one baseband channel of one baseline for one integration time. Each starts with a text header and is followed by binary data. The text header uses the typical DiFX parameters format with the “Key” starting at the beginning of a text line and ending with a colon, and the value starting in the 21st column of text. The header rows occur in the following order:

Key	Units/allowed values	Comments
BASELINE NUM	integer	$= (a_1 + 1) * 256 + (a_2 + 1)$ for $a_1, a_2 \geq 1$
MJD	integer	date of visibility centroid
SECONDS	float	seconds since beginning of MJD
CONFIG INDEX	≥ 0	index to `.input` file configuration table
SOURCE INDEX	≥ 0	index to `.delay` file scan number
FREQ INDEX	≥ 0	index to `.input` frequency table
POLARISATION PAIR	2 of (`R`, `L`, `X`, `Y`)	e.g., `RR` or `RL`
DATA WEIGHT	≥ 0.0	data weight for spectrum; typically ~ 1
U (METRES)	meters	u component of baseline vector
V (METRES)	meters	v component of baseline vector
W (METRES)	meters	w component of baseline vector

Following the end-of-line mark for the last header row begins binary data in the form of (real, imaginary) pairs of 32-bit floating point numbers. The `.input` file parameter `NUM CHANNELS` indicates the number of complex values to expect. In the case of upper sideband data, the first reported channel is the "zero frequency" channel, that is its sky frequency is equal to the value in the frequency table for this spectrum. The Nyquist channel is not retained. For lower sideband data, the last channel is the "zero frequency" channel. That is, in all cases, the spectrum is in order of increasing frequency and the Nyquist channel is excised.

9.11 .dir

Reading directory information off Mark5 modules can take a bit of time (measured in minutes usually). Since the same modules are often accessed multiple times, the directories are cached in `$MARK5_DIR_PATH/` . In this directory, there will be one file per module that has been used, named *VSN*`.dir`, where *VSN* is the volume serial number of the module, i.e. NRAO−023. The format of these files is as follows: The first line contains three fields: *VSN*, the number of scans on the module, *nScan*, and either `A` or `B` indicating the last bank the module was installed in. Then there are *nScan* rows containing information about each scan, each with 11 columns. Values are floating point unless otherwise noted.

Key	Units/allowed values	Comments
Start byte	64-bit integer bytes	offset of the scan on the Mark5 module
Length	64-bit integer bytes	length of the scan
Start day	integer MJD	the modified Julian day of the scan start
Start time	integer seconds	the scan start time
Frame num	integer	frame number since last second tick
Frames per sec	integer	number of frames per second
Scan Duration	seconds	the duration of the scan
Frame size	integer bytes	the length of one data frame, including headers
Frame offset	integer bytes	the offset to the start of the first entire frame
Tracks	integer	the number of data tracks
Format	integer	0 for VLBA format, 1 for Mark4 format
Name	string	scan name, usually including the project code and station

Note: The directory format used for NRAO-DiFX differs from that used by the hardware correlator (found in */home/fxcorr/mark5*) and are not interchangeable.

9.12 .FITS

The `.FITS` files discussed here are produced by **`difx2fits`**. They aim to conform to the same table structures as the FITS-IDI files produced by the VLBA correlator. The format is described in AIPS Memo 102, "The FITS Interferometry Data Interchange Format", however, this memo is a bit out of date and the data structures described are not in exact agreement with those made by the VLBA correlator; in all cases the format of data produced by the VLBA hardware correlator is favored where the two disagree. The tables in these FITS files have a nearly 1 to 1 relationship with the tables that are seen within AIPS, though their two letter abbreviations differ. The following tables are produced by `difx2fits`:

Table	Description
AG	The array geometry table
SO	The source table
AN	The antenna table
FR	The frequency table
ML	The model table
CT	The correlator (eop) table
MC	The model components table
SO	The spacecraft orbit table 1.1
UV	The visibility data table
FG	The flag table
TS	The system temperature table
PH	The phase calibration table (pulse cals and state counts)
WR	The weather table
GN	The gain curve table
GM	The pulsar gate model table 1.1

Not all of these tables will always be written.

9.13 .im 1.1

The `.im` file contains polynomial models used by `difx2fits` in the creation of `FITS` files. After a header that is similar to that of a `.rate` file, the contents are organized hierarchically with scan number, sub-scan interval, and antenna number being successively faster-incrementing values. The keys and allowed values in this section are summarized below: Note that the values of the delay polynomials in this file have the opposite sign as compared to those generated by CALC and those stored in `.FITS` files. Keys preceded by $\star$ are optional.

	Key	Units/allowed values	Comments
⋆	CALC SERVER	string	name of the calc server computer used
⋆	CALC PROGRAM	integer	RPC program ID of the calc server used
⋆	CALC VERSION	integer	RPC version ID of the calc server used
	START MJD	MJD + fraction	start time of this subjob
	START YEAR	integer	calendar year of START MJD
	START MONTH	integer	calendar month of START MJD
	START DAY	integer	day of calendar month of START MJD
	START HOUR	integer	hour of START MJD
	START MINUTE	integer	minute of START MJD
	START SECOND	integer	second of START MJD
	POLYNOMIAL ORDER	2, 3, 4 or 5	polynomial order of interferometer model *order*
	INTERVAL (SECS)	integer	interval between new polynomial models
	ABERRATION CORR	`UNCORRECTED` / `APPROXIMATE` / `EXACT`	level of u, v, w aberration correction
	NUM TELESCOPES	integer ≥ 1	number of telescopes (*nTelescope*) The row below is duplicated *nTelescope* times.
	TELESCOPE t NAME	string	upper case antenna name abbreviation
	NUM SCANS	integer ≥ 1	number of scans (*nScan*). Everything below is duplicated *nScan* times.
	SCAN s SRC NAME	string	systematic name to match that used in `.input file`
	SCAN s NUM POLY	≥ 1	number of polynomials covering scan ($nPoly_s$) Everything below is duplicated *nPoly* times.
	SCAN s POLY p MJD	integer ≥ 0	the start MJD of this polynomial
	SCAN s POLY p SEC	integer ≥ 0	the start sec of this polynomial Everything below is duplicated *nTelescope* times.
	ANT a DELAY (us)	*order*+1 numbers	terms of delay polynomial
	ANT a DRY (us)	*order*+1 numbers	terms of dry atmosphere
	ANT a WET (us)	*order*+1 numbers	terms of wet atmosphere
	ANT a U (m)	*order*+1 numbers	terms of baseline u
	ANT a V (m)	*order*+1 numbers	terms of baseline v
	ANT a W (m)	*order*+1 numbers	terms of baseline w

9.14 .input

This section describes the `.input` file format used by `mpifxcorr` to drive correlation. Because NRAO-DiFX 1.0 uses a non-standard branch of `mpifxcorr` some of the data fields will differ from those used in the official version, either in parameter name or in the available range of values. Currently the parameters must be in the order listed here. To get the most out of this section it is advisable to look at an actual file while reading. An example file is stashed at `http://www.aoc.nrao.edu/~wbrisken/NRAO-DiFX-1.1/` . In the tables below, numbers are assumed to floating point unless otherwise stated.

Note that the input file format has undergone a few minor changes since NRAO-DiFX version 1.0.

9.14.1 Common settings table

Below are the keywords and allowed values for entries in the common settings table. This table begins with header

```
# COMMON SETTINGS ##!
```

This is always the first table in a `.input` file.

Key	Units/allowed values	Comments
DELAY FILENAME	string	name and full path to `.delay` file
UVW FILENAME	string	name and full path to `.uvw` file
CORE CONF FILENAME	string	name and full path to `.threads` file
EXECUTE TIME (SEC)	integer seconds	observe time covered by this `.input` file
START MJD	integer MJD	start date
START SECONDS	integer seconds	start time
ACTIVE DATASTREAMS	integer ≥ 2	number of antennas (*nAntenna*)
ACTIVE BASELINES	integer ≥ 1	number of baselines to correlate (*nBaseline*)
VIS BUFFER LENGTH	integer ≥ 1	the number of concurrent integrations to allow 1.1
OUTPUT FORMAT	boolean	always `SWIN` here
OUTPUT FILENAME	string	name of output `.difx` directory

Typically, $nBaseline = nAntenna \cdot (nAntenna - 1)/2$. Autocorrelations are not included in this count.

9.14.2 Configurations table

Below are the keywords and allowed values for entries in the configurations table. This table begins with header

```
# CONFIGURATIONS ###!
```

Two indexes are used for repeated keys. The index over datastream (antenna) is *d*, running from 0 to *nAntenna* - 1 and the index over baseline is *b*, running from 0 to *nBaseline* - 1.

Key	Units/allowed values	Comments
NUM CONFIGURATIONS	integer ≥ 1	number of modes in file (*nConfig*)
CONFIG SOURCE	string	name of configuration
INT TIME (SEC)	seconds	integration time
NUM CHANNELS	integer ≥ 1	number of channels (FFT size, *nFFT*, is twice this)
CHANNELS TO AVERAGE	integer ≥ 1	not yet supported (set to 1)
OVERSAMPLE FACTOR	integer ≥ 1	total oversampling factor of baseband data 1.1
DECIMATION FACTOR	integer ≥ 1	portion of oversampling to handle by decimation 1.1
BLOCKS PER SEND	integer ≥ 1	number of FFT sizes to send at a time to a core
GUARD BLOCKS	integer ≥ 0	number of extra blocks to send for overlap
POST-F FRINGE ROT	boolean	fringe rotate after FFT? Always *FALSE* here
QUAD DELAY INTERP	boolean	use quadratic, not linear, delay interpolation
WRITE AUTOCORRS	boolean	enable auto-correlations; *TRUE* here
PULSAR BINNING	boolean	enable pulsar mode; *FALSE* for now
PULSAR CONFIG FILE	string	(*only if BINNING is True*) see § 9.19
DATASTREAM *d* INDEX	integer ≥ 0	DATASTREAM table index, starting at 0
BASELINE *b* INDEX	integer ≥ 0	BASELINE table index, starting at 0

9.14.3 Frequency table

Below are the keywords and allowed values for entries in the frequency table which defines all possible sub-bands used by the configurations in this file. Each sub-band of each configuration is mapped to one of these through a value in the datastream table (§9.14.5). Each entry in this table has three parameters which are replicated for each frequency table entry. This table begins with header

```
# FREQ TABLE #######!
```

The table below uses *f* to represent the frequency index, which ranges from 0 to *nFreq* - 1.

Key	Units/allowed values	Comments
FREQ ENTRIES	integer ≥ 1	number of frequency setups (*nFreq*)
FREQ (MHZ) *f*	MHz	sky frequency at band edge
BW (MHZ) *f*	MHz	bandwidth of sub-band
SIDEBAND *f*	`U` or `L`	net sideband of sub-band

9.14.4 Telescope table

Below are the keywords and allowed values for entries in the telescope table which tabulates antenna names and their associated peculiar clock offsets, and the time derivatives of these offsets. Much of the other antenna-specific information is stored in the datastream table (§9.14.5). Each datastream of each configuration is mapped to one of these through a value in the datastream table. Each entry in this table has three parameters which are replicated for each telescope table entry. This table begins with header

```
# TELESCOPE TABLE ##!
```

The table below uses *a* to represent the antenna index, which ranges from 0 to *nAntenna* - 1.

Key	Units/allowed values	Comments
TELESCOPE ENTRIES	integer ≥ 1	number of antennas (*nAntenna*)
TELESCOPE NAME *a*	string	abbreviation of antenna name
CLOCK DELAY (us) *a*	μsec	clock error at start of associated `.delay` file
CLOCK RATE(us/s) *a*	μsec/sec	rate at which antenna clock is drifting

9.14.5 Datastream table

The datastream table begins with header

```
# DATASTREAM TABLE #!
```

The table below uses *f* to represent the frequency index, which ranges from 0 to *nFreq* - 1. A second index, *i*, is used to cover the range 0 to *nBB* - 1, where the total number of basebands is given by $nBB \equiv \sum_f nPol_f$. In the NRAO-DiFX system, all sub-bands must have the same polarization structure, so $nBB = nFreq \cdot nPol$.

Key	Units/allowed values	Comments
DATASTREAM ENTRIES	integer ≥ 1	number of antennas (*nDatastream*)
DATA BUFFER FACTOR	integer ≥ 1	
NUM DATA SEGMENTS	integer ≥ 1	
TELESCOPE INDEX	integer ≥ 0	telescope table index of datastream
TSYS	Kelvin	if zero (normal in NRAO usage), don't scale data by *tsys*
DATA FORMAT	string	data format
QUANTISATION BITS	integer ≥ 1	bits per sample
DATA FRAME SIZE	integer ≥ 1	bytes in one frame(or file) of data 1.1
DATA SOURCE	string	`FILE` (see §6.3) or `MODULE` for Mark5 playback 1.1
FILTERBANK USED	boolean	currently only `FALSE`
NUM FREQS	integer ≥ 0	number of different frequencies for this datastream
FREQ TABLE INDEX *f*	integer ≥ 0	
CLK OFFSET *f* (us)	μsec	
NUM POLS *f*	1 or 2	for NRAO usage, all such parameters must be the same
INPUT BAND *i* POL	*R* or *L*	polarization identity
INPUT BAND *i* INDEX	integer ≥ 1	index to frequency setting array above; *nBB* per entry

9.14.6 Baseline table

In order to retain the highest level of configurability, each baseline can be independently configured at some level. This datastream table begins with header

```
# BASELINE TABLE ###!
```

The baseline table has multiple entries, each one corresponding to a pair of antennas, labeled `A` and `B` in the table. For each of *nBaseline* baseline entries, *nFreq* sub-bands are processed, and for each a total of *nProd* polarization products are formed. Indexes for each of these dimensions are b, f and p respectively, each starting count at 0. Within the NRAO-DiFX context, all baselines must have the same *nFreq* and *nProd*, though this is not a requirement of `mpifxcorr` in general.

Key	Units/allowed values	Comments
BASELINE ENTRIES	integer ≥ 1	number of entries in table, *nBaseline*
D/STREAM A INDEX b	integer ≥ 0	datastream table index of first antenna
D/STREAM B INDEX b	integer ≥ 0	datastream table index of second antenna
NUM FREQS b	integer ≥ 1	number of frequencies on this baseline, $nFreq_b$
POL PRODUCTS b/f	integer ≥ 1	number of polarization products, $nProd_b$
D/STREAM A BAND p	integer ≥ 0	index to frequency array in datastream table
D/STREAM B BAND p	integer ≥ 0	same as abovem, but for antenna `B`, not `A`

9.14.7 Data Table

In the following table, d is the datastream index, ranging from 0 to *nDatastream* - 1 and f is the file index ranging from 0 to $nFile_d$.

Key	Units/allowed values	Comments
D/STREAM d FILES	integer ≥ 1	number of files $nFile_d$ associated with datastream d
FILE d/f	string	name of file or module associated with datastream d

For datastreams reading off Mark5 modules, *nFile* will always be 1 and the filename is the *VSN* of the module being read.

9.15 .machines

The `.machines` file is used by `mpirun` to determine which machines will run `mpifxcorr`. This is a text file containing a list of computers, one to a line possibly with additional options listed, on which to spawn the software correlator process. As a general rule the MPI rank, a unique number for each process that starts at 0, are allocated in the order that the computer names are listed. This general rule can break down in cases where the same computer name is listed more than once; the behavior in this case depends on the MPI implementation being used. MPI rank 0 will always be the manager process. Ranks 1 through *nDatastream* will each be a datastream process. Additional processes will be computing (core) processes. If more processes are specified for `mpirun` with the `-np` option than there are lines in this file, the file will be read again from the top, so the processes will be assigned in a cyclic fashion (again, this depends somewhat on the MPI implementation and the other parameters passed to `mpirun`; for NRAO-DiFX with OpenMPI, this assumes `--bynode` is used). If the program `startdifx` is used to start the correlation process, the number of processes to start is determined by the number of lines in this file. If wrapping to the top of this file is desired, dummy comment lines (beginning with #) can be put at the end of the `.machines` file to artificially raise the number of processes to spawn. Within NRAO-DiFX, this file is typically produced by `genmachines`. Keep in mind that this file is directly read by the MPI execution program `mpirun` and the format of the file may differ depending on the MPI implementation that you are using. With OpenMPI appending `slots=1 max-slots=1` to the end of each line ensures that a single instance of `mpifxcorr` is run on that machine. If both a datastream process and a core process are to be run on the same computer, then using options `slots=1 max-slots=2` might be appropriate.

9.16 .log 1.1

When generation of sniffer output files is not disabled, each `.FITS` file written by `difx2fits` will be accompanied by a corresponding `.log` file. This file contains a summary of the contents of that `.FITS` file. It is analogous to the `logfile.lis` file produced by the old `FITSsniffer` program. This file is free-form ASCII that is intended for viewing by human eyes, and is should not be used as input to any software as the format is not guaranteed to remain constant.

9.17 pcal

A file called `pcal` is created when program `vlog` operates on the `cal.vlba` file. This file contains three measurements: the cable length calibration, pulse calibration, and state counts. This file contains two kinds of lines. Comment lines begin with an octothorpe (#) and contain no vital information. Data lines always contain 8 fixed-size fields:

1. Station name abbreviation, e.g., `LA`
2. Time centroid of measurement (day of year, including fractional portion)
3. Duration of measurement (days)
4. Cable calibration measurement (picoseconds)
5. Number of polarizations with measurements (hereafter called *nPol*)
6. Number of sub-bands with measurements (hereafter called *nBand*)
7. Number of pulse cal tones detected per band per polarization, possibly zero (hereafter called *nTone*)
8. Number of state count states measured per band per polarization, possibly zero (hereafter called *nState*)
9. Number of record channels at time of measurement ($\leq$ *nPol * nBand)*

Following these eight fields are two variable-length arrays of numbers. The first variable-length field is the pulse cal data field consisting of *nPol*nBand*nTone* groups of four numbers. The first member of this group is the recorder channel number (zero-based) corresponding to the measurement. The second member of this group is the tone sky frequency (MHz). The third and fourth are respectively the real and imaginary parts of the tone measured at the given sky frequency. The order in which the groups are presented (in 'C' language array syntax, as used throughout this document) is [*nPol*][*nBand*][*nTone*]. Note that if there are fewer than *nPol*nBand* record channels, the record channel will be -1 for some groups. The second variable-length field is the state count data. For each band of each polarization, *nState* + 1 values are listed. The first number is the record channel number or -1 if that polarization/band combination was not observed or monitored. The remainder contain state counts. *nState* can be either 0 or 2^{nBit}, where *nBit* is the number of quantization bits. The order in which these groups are listed is [*nPol*][*nBand*].

9.18 .polyco 1.1

A polyco file contains a single polynomial for pulse phase that is valid for a fraction (up to 100%) of a job file. An additional numeric suffix is appended to the filename specifying the polynomial index for a particular `.pulsar` file that shares the same base name. The format of the file is the same as a TEMPO pulsar file [5].

9.19 .pulsar 1.1

Within the scope of NRAO-DiFX, all pulsar configuration files will adopt the `.pulsar` suffix. There will be a separate .pulsar file for each pulsar-configuration combination in each .fx file that is converted to DiFX input file format by `job2difx`; a series of 4 numbers preceding the suffix indicates which source number, frequency id, correlator table entry, and configuration the file belongs to. Multiple subjobs derived from a single `.fx` file may share `.pulsar` files.

Key	Units/allowed values	Comments
NUM POLYCO FILES	integer ≥ 1	number of polyco files to look at (*nPoly*)
		The row below is duplicated *nPoly* times.
POLYCO FILE *p*	string	filename of p^{th} polyco file; see § 9.18
NUM PULSAR BINS	integer ≥ 1	number of pulsar bins to consider (*nBin*)
SCRUNCH OUTPUT	boolean	TRUE for regular "gating"; FALSE not yet supported
		The two rows below are duplicated *nBin* times.
BIN PHASE END *b*	float	pulse phase (in periods) of end of bin *b*
BIN WEIGHT *b*	float ≥ 0	weight to apply to bin *b*

9.20 .rate

In order to deliver model information to `difx2fits` that is not needed by `mpifxcorr`, an additional file is written by `calcif2`. The format of this file is most similar to that of the `.uvw` files, however the payload in this file is very different. Note that the values of the delays in this file have the opposite sign as compared to those generated by CALC and those stored in `.FITS` files.

The file consists entirely of key-value pairs separated by a colon. There are three sections in the `.rate` file; these sections are not separated by any explicit mark in the file.

The first section contains values that are fixed for the entire experiment and at all antennas — all data in this section is scalar. In the following table, all numbers are assumed to be floating point unless further restricted. The keys and allowed values in this section are summarized below. Keys preceded by $\star$ are optional.

Key	Units/allowed values	Comments
$\star$ CALC SERVER	string	name of the calc server computer used
$\star$ CALC PROGRAM	integer	RPC program ID of the calc server used
$\star$ CALC VERSION	integer	RPC version ID of the calc server used
START MJD	MJD + fraction	start time of this subjob
START YEAR	integer	calendar year of START MJD
START MONTH	integer	calendar month of START MJD
START DAY	integer	day of calendar month of START MJD
START HOUR	integer	hour of START MJD
START MINUTE	integer	minute of START MJD
START SECOND	integer	second of START MJD
INCREMENT (SECS)	integer	seconds between computed model points (*inc*)

The second section contains antenna(telescope) specific information. This section is absolutely identical to the telescope section describe for the `.calc` file (see second table in §9.8) so its description will be omitted here.

The third and final section contains the scan-based information. First is a line indicating the number of scans to follow. Then for each scan, numbered by *s* ranging from 0 to *nScan* - 1, there are 5 lines containing information about the scan, including the number of sampled points within that scan, *nPoint*. Finally there are $nPoint_s + 3$ lines containing the tabulated data, numbered -1 through $nPoints_s + 1$, indexed with *p*. Note that this includes one sample before the start of the scan and at least one after the scan end, allowing for a quadratic interpolation across the entire scan. This information is summarized in the following table:

Key	Units/allowed values	Comments
NUM SCANS	integer ≥ 1	number of scans (*nScan*). The rows below are duplicated *nScan* times.
SCAN *s* POINTS	≥ 1	duration of scan in units of *inc* ($nPoints_s$)
SCAN *s* START PT	integer ≥ 0	start time of scan in units of *inc* since MJD START
SCAN *s* SRC NAME	string	systematic name to match that used in `.input file`
SCAN *s* SRC RA	radians	J2000 right ascension
SCAN *s* SRC DEC	radians	J2000 declination
RELATIVE INC *p*	array; see below	$nPoint_s + 3$ of these lines per scan

Like for the `.delay` and `.uvw` files, the values reported in this file extend one full mode increment (*inc*) before and after the actual duration of the scan, and hence will overlap in time by 2× *inc* with consecutive scans. The `RELATIVE INC` lines contain triplets of values for each antenna. The first member of the triplet is the time derivative of the geometric delay, in μs/s. The second and third elements are the estimated dry and wet atmospheric delay terms, respectively, in μs.

This file is typically produced by `calcif2`.

9.21 .threads

The `.threads` file tells `mpifxcorr` how many threads to start on each processing node. Within NRAO-DiFX, this file is typically produced by `genmachines`. The `.threads` file has a very simple format. The first line starts with `NUMBER OF CORES:`. Starting at column 21 is an integer that should be equal to the number of processing nodes (*nCore*) specified in the corresponding `.machines` file. Each line thereafter should contain a single integer starting at column 1. There should be *nCore* such lines.

9.22 tsys

A file called `tsys` is created when program `vlog` operates on the `cal.vlba` file. This file contains measurements of the system temperature and name of receiver for each baseband channel. This file contains two finds of lines. Comment lines begin with an octothorpe (#) and contain no vital information. Data lines always contain 4 fixed-size fields:

1. Station name abbreviation, e.g., `LA`
2. Time centroid of measurement (day of year, including fractional portion)
3. Duration of measurement (days); currently set to zero for lack of information
4. Number of baseband channels recorded (*nRecChan*)

Following these 4 fields are *nRecChan* pairs, one for each baseband channel. The first element of each pair is the system temperature and the second is the receiver name (e.g., `4cm`, or `7mm`).

9.23 weather

A file called `weather` is created when program `vlog` operates on the `cal.vlba` file. This file contains tabulated values of various meteorological measurements. This file contains two finds of lines. Comment lines begin with an octothorpe (#) and contain no vital information. Data lines always contain 8 fixed-size fields:

1. Station name abbreviation, e.g., `LA`
2. Time of measurement (day of year, including fractional portion)
3. Ambient temperature (Centigrade)

4. Pressure (mbar)
5. Dew point (Centigrade)
6. Wind speed (m/s)
7. Wind direction (degrees)
8. Rain (cm)
9. Wind gust (m/s)

9.24 .wts 1.1

When generation of sniffer output files is not disabled, each `.FITS` file written by `difx2fits` will be accompanied by a corresponding `.wts` file. This file contains statistics of the data weights, typically dominated by the completeness of records as determined by the data transport system, over a typically 30 second long period.

The first line is simply the observe code, e.g., `MT831` .

Each additional line in the file is a complete record for a given antenna for a given interval, containing information for each baseband channel separately. The format of these lines is as follows:

Key	Units/allowed values	Comments
MJD	integer ≥ 1	MJD day number corresponding to line
hour	≥ 0.0, < 24.0	hour within day
antenna number	≥ 1	antenna table index
antenna name	string	
n_{BBC}	≥ 1	Number of baseband channels
mean weight	≥ 0.0	This column repeated n_{BBC} times
min weight	≥ 0.0	This column repeated n_{BBC} times
max weight	≥ 0.0	This column repeated n_{BBC} times

This file can be used directly with plotting program `plotwt` or used more automatically with `difxsniff`.

9.25 .uvw

The `.uvw` file contains tabulated baseline vectors. The values in this file are interpolated to produce values valid for integration centroids and included in the visibility file output by `mpifxcorr`. This file consists entirely of key-value pairs separated by a colon. There are three sections in the `.uvw` file; these sections are not separated by any explicit mark in the file.

The first section contains values that are fixed for the entire experiment and at all antennas — all data in this section is scalar. In the following table, all numbers are assumed to be floating point unless further restricted. The keys and allowed values in this section are summarized below:

Key	Units/allowed values	Comments
START YEAR	integer	calendar year of START MJD
START MONTH	integer	calendar month of START MJD
START DAY	integer	day of calendar month of START MJD
START HOUR	integer	hour of START MJD
START MINUTE	integer	minute of START MJD
START SECOND	integer	second of START MJD
INCREMENT (SECS)	integer	seconds between tabulated baseline vectors (*inc*)

The second section contains antenna(telescope) specific information with the following format:

Key	Units/allowed values	Comments
NUM TELESCOPES	integer ≥ 1	number of telescopes (*nTelescope*). The rows below are duplicated *nTelescope* times.
TELESCOPE *t* NAME	string	upper case antenna name abbreviation
TELESCOPE *t* MOUNT	string	the mount type: altz, equa, xyew, or xyns
TELESCOPE *t* X (m)	meters	X geocentric coordinate of antenna at date
TELESCOPE *t* Y (m)	meters	Y geocentric coordinate of antenna at date
TELESCOPE *t* Z (m)	meters	Z geocentric coordinate of antenna at date

The third and final section contains the scan-based information. First is a line indicating the number of scans to follow. Then for each scan, numbered by s ranging from 0 to *nScan* - 1, there are 5 lines containing information about the scan, including the number of sampled points within that scan, *nPoint*. Finally there are $nPoint_s$ + 3 lines containing the tabulated data, numbered -1 through $nPoints_s$ + 1, indexed with p. Note that this includes one sample before the start of the scan and at least one after the scan end, allowing for a quadratic interpolation across the entire scan. This information is summarized in the following table:

Key	Units/allowed values	Comments
NUM SCANS	integer ≥ 1	number of scans (*nScan*). The rows below are duplicated *nScan* times.
SCAN *s* POINTS	≥ 1	duration of scan in units of *inc* ($nPoints_s$)
SCAN *s* START PT	integer ≥ 0	start time of scan in units of *inc* since MJD START
SCAN *s* SRC NAME	string	systematic name to match that used in `.input file`
SCAN *s* SRC RA	radians	J2000 right ascension
SCAN *s* SRC DEC	radians	J2000 declination
RELATIVE INC *p*	array; see below	$nPoint_s$ + 3 of these lines per scan

Like for the `.delay` and `.rate` files, the values reported in this file extend one full mode increment (*inc*) before and after the actual duration of the scan, and hence will overlap in time by 2× *inc* with consecutive scans. The `RELATIVE INC` lines contain triplets of values for each antenna, representing in order the u, v, and w components of the baseline vector, all expressed in meters from the earth center.

This file is typically produced by `calcif2`.

9.26 .xcb 1.1

When generation of sniffer output files is not disabled, each `.FITS` file written by `difx2fits` will be accompanied by a corresponding `.xcb` file. This file contains cross-correlation spectra for each antenna for each baseline. In order to minimize the output data size, spectra for the same source will only be repeated once per 15 minutes. The file contains many concatenated records. Each record has the spectra for all baseband channels for a particular baseline and has the following format which is very similar to that of the `.acb` files. Note that no spaces are allowed within any field. Values in `typewriter` font without comments are explicit strings that are required.

Line(s)	Value	Units	Comments
1	`timerange:`		
	MJD	integer ≥ 1	MJD day number corresponding to line
	start time	string	e.g., `13h34m22.6s`
	stop time	string	e.g., `13h34m52.0s`
	`obscode:`		
	observe code	string	e.g., MT831
	`chans:`		
	n_{chan}	≥ 1	number of channels per baseband channel
	`x`		
	n_{BBC}	≥ 1	number of baseband channels
2	`source:`		
	source name	string	e.g., `0316+413`
	`bandw:`		
	bandwidth	MHz	baseband channel bandwidth
	`MHz`		
3 to $2+n_{\text{BBC}}$	`bandfreq:`		
	frequency	GHz	band edge (SSLO) frequency of baseband channel
	`GHz polar:`		
	polarization	2 chars	e.g. `RR` or `LL`
	`side:`		
	sideband	`U` or `L`	for upper or lower sideband
	`bbchan:`		
	bbc	`0`	Currently not used but needed for conformity
$3+n_{\text{BBC}}$ to $2+n_{\text{BBC}}(n_{\text{chan}}+1)$	*ant1 number*	≥ 1	number of first antenna
	ant2 number	≥ 1	
	ant1 name	string	
	ant2 name	string	
	channel number	≥ 1	$= \text{chan} + (\text{bbc} - 1) \cdot n_{\text{chan}}$ for chan, bbc ≥ 1
	amplitude	≥ 0.0	
	phase	degrees	

The above are repeated for each cross correlation spectrum record. This file can be plotted directly with `plotbp` or handled more automatically with `difxsniff`.

10 XML message types 1.1

The `difxmessage` library (§11.5) implements a system for sending and receiving messages using XML format. This section documents the "difxMessage" XML document type that is used for interprocess communication during correlation within NRAO-DiFX. These messages are sent via UDP multicast and are thus restricted to fit within one standard-sized Ethernet packet (~1500 bytes). Various logging and monitoring programs (`mk5mon`, `cpumon`, and `errormon`, all eventually to be replaced by a single interactive operator interface) can accept these messages and perform actions based on their content. Several different message types are derived from the following XML base type:

```
<?xml version="1.0" encoding="UTF-8"?>
<difxMessage>
  <header>
    <from>from</from>
    <to>to</to>
    <mpiProcessId>mpiId</mpiProcessId>
    <identifier>idString</identifier>
    <type>messageType</type>
  </header>
  <body>
    <seqNumber>seqNum</seqNumber>
    body
  </body>
</difxMessage>
```

The italicized fields are as follows:

from the hostname of the sender.

to the intended recipient of the XML document. Note that this field is typically not included for report-only messages as it's intended purpose is for directing commands to particular recipients. Also note that multiple *to* fields can be present in a message. Three "shortcuts" are currently allowed: `all` causes all receiving programs (such as `mk5daemon`) on all software correlator cluster members to respond; `mark5` causes all Mark5 units to respond; and `swc` causes all non-Mark5 units to respond.

mpiId the MPI process id of the sender. If there are D (typically 10) datastream processes (i.e. Mark5s playing back), then *mpiId* takes on the following numbers:

value	`mpifxcorr` process type
< 0	a process not associated with `mpifxcorr`
0	the manager process of `mpifxcorr`
1 to D	one of the datastream processes
$\geq D + 1$	one of the core (computing) processes

idString an additional string identifying the source of the message. For messages sent from `mpifxcorr`, this will be the job id, for example `job3322.000`. Other programs will typically set this field to the name of the program sending the message.

messageType the type of message being sent:

value	description of message type
`DifxLoadMessage`	CPU and memory usage (usually sent by `mk5daemon`).
`DifxErrorMessage`	an error message.
`Mark5StatusMessage`	status of the mark5 unit and modules.
`DifxStatusMessage`	status of the `mpifxcorr` program.
`DifxInfoMessage`	correlation progress information.
`DifxWeightMessage`	playback weights for each station being correlated.
`DifxCommand`	a command message.

Each of these message types is described in sections that follow.

seqNum the sequence number (starting at 0) of messages coming from the particular program running on the particular host. The number advances by 1 for each sent message and can be used to detect lost packets.

body message contents that are specific to the particular *messageType*. See sections that follow.

A "C" language library for generating, multicasting, receiving, and parsing XML documents of this type is used within some of the programs, including `mpifxcorr` (the core of the DiFX [1] software correlator) and `mk5daemon` (a program that runs on each Mark5 unit that is responsible for multicast communication when `mpifxcorr` is not running), that transact these XML documents. The default multicast group to be used is 224.2.2.1 and the default port is 50200, though these can be overridden with environment variables `DIFX_MESSAGE_GROUP` and `DIFX_MESSAGE_PORT` respectively.

10.1 DifxLoadMessage

This section describes messages with *messageType* = `DifxLoadMessage`. These messages contain CPU and memory utilization information and are voluntarily sent by various nodes of the cluster, to be received by the operator interface.

The *body* of this message type contains:

```
<difxLoad>
  <cpuLoad>cpuLoad</cpuLoad>
  <totalMemory>totalMemory</totalMemory>
  <usedMemory>usedMemory</usedMemory>
</difxLoad>
```

The italicized fields are as follows:

cpuLoad CPU utilization on the cluster node. It is a floating point value containing the average number of processes scheduled at one time.

totalMemory total memory on node, in kiB.

usedMemory used memory on node, in kiB.

10.2 DifxErrorMessage

This section describes messages with *messageType* = `DifxErrorMessage`. These messages come from mpifxcorr or the head node agent and contain an error message string and severity code that should be displayed to the operator and logged.

The *body* of the message contains:

```
<difxError>
  <errorMessage>message</errorMessage>
  <severity>severity</severity>
</difxError>
```

The italicized fields are as follows:

message a string containing the error message.

severity an integer indicating the severity. The severity scale is based on that from the EVLA and has values with the following meanings:

value	meaning
0	processing has failed; a restart is needed
1	data from one or more station is affected badly
2	data from one or more station may be affected; e.g., low weights
3	minor error of no consequence to ongoing processing
4	informational only

10.3 Mark5StatusMessage

This section describes messages with *messageType* = `Mark5StatusMessage`. This message type cones from either mpifxcorr or a mk5agent (or perhaps another program that makes heavy use of Mark5 units and wishes to volunteer status information).

The *body* of the message contains:

```
<mark5Status>
  <bankAVSN>vsnA</bankAVSN>
  <bankBVSN>vsnB</bankBVSN>
  <statusWord>statusWord</statusWord>
  <activeBank>activeBank</activeBank>
  <state>state</state>
  <scanNumber>scanNumber</scanNumber>
  <scanName>scanName</scanName>
  <position>position</position>
  <playRate>playRate</playRate>
  <dataMJD>dataMJD</dataMJD>
</mark5Status>
```

The italicized fields are as follows:

vsnA the VSN of the module in bank A.

vsnB the VSN of the module in bank B.

statusWord a hexidecimal number with the following bits: *FIXME TBD*

activeBank the active bank, either A or B.

state the state of the Mark5 unit:

state	meaning
`Opening`	the StreamStor card is being opened.
`Open`	the StreamStor was successfully opened and is ready for use.
`Close`	the StreamStor has been closed.
`GetDirectory`	the unit is recovering the directory or finding data.
`GotDirectory`	the unit successfully found needed data on the module.
`Play`	the unit is playing back data.
`PlayStart`	the unit is about to start playback.
`PlayInvalid`	the unit is playing data, but the data is invalid.
`Idle`	the unit is not doing anything; no process has control of it.
`Error`	the unit is unusable due to an error.
`Busy`	the unit is busy and cannot respect commands.
`Initializing`	the StreamStor card is initializing.
`Resetting`	the unit is resetting the StreamStor card.
`Rebooting`	the unit is about to reboot.
`Poweroff`	the unit is about to turn off.
`NoData`	the unit is not playing data since there is none that is appropriate.
`NoMoreData`	the unit has played all the data for the job and is stopped.

scanNumber the directory index number for the current scan. This number starts at 1.

scanName the name associated with the current scan.

position the byte position being accessed. Note that this number can be very large ($> 2^{46}$).

playRate the time-averaged playback rate in Mbps.

dataMJD the date stamp (MJD + fraction) of the most recently read data.

10.4 DatastreamStatusMessage

FIXME – to be implemented

10.5 DifxStatusMessage

This section describes messages with *messageType* = `DifxStatusMessage`. This message type is only produced by `mpifxcorr` or the programs immediately responsible for starting and stopping it.

The *body* of the message contains:

```
<difxStatus>
  <state>state</state>
  <message>message</message>
  <visibilityMJD>visibilityMJD</visibilityMJD>
  <weight antId=antId weight=weight>
</difxStatus>
```

The italicized fields are as follows:

state the state of `mpifxcorr`, which must be one of the following:

state	meaning
`Spawning`	the `mpifxcorr` processes are being started (not sent by `mpifxcorr`).
`Starting`	all the processes are ready to begin.
`Running`	the correlator is running.
`Ending`	the correlator has reached the end of the job.
`Done`	the correlation has completed.
`Aborting`	correlation is stopping early due to an error.
`Terminating`	correlation is stopping early due to signal.
`Terminated`	correlation has stopped early.
`MpiDone`	all of the MPI processes have ended (not sent by `mpifxcorr`).

message a string containing information for the operator.

visibilityMJD the time-stamp (MJD + fraction) of last completed visibility record.

antId the antenna id for the associated weight, ranging from 0 to $N_{\mathrm{ant}} - 1$.

weight the data weight for the associated antenna, ranging from 0 to 1. Note that in each XML document of this type there will in general be one *weight* value for each antenna being correlated.

10.6 DifxInfoMessage

This section describes messages with *messageType* = `DifxInfoMessage`. This document type simply contains information for the operator about the status of correlation. Note that this message type is likely to disappear as DifxErrorMessage with *severity* = 4 is essentially the same.

The *body* of the message contains:

```
<difxInfo>
  <message>message</message>
</difxInfo>
```

The italicized field is as follows:

message a string containing information for the operator.

10.7 DifxCommand

This section describes messages with *messageType* = `DifxCommand`. These messages require the *to* field to be set and cause the intended recipient to take an action.

The *body* of the message contains:

```
<difxCommand>
  <command>command</command>
</difxCommand>
```

The italicized field is as follows:

command the command to execute. Commands are not case sensitive and should be among the following:

command	action
`GetVSN`	cause the mark5 unit to multicast loaded VSNs if possible.
`GetLoad`	request CPU and memory usage to be reported.
`ResetMark5`	cause `SSReset` and `ssopen` to be run to reset StreamStor.
`StartMark5A`	start the Mark5A program.
`StopMark5A`	stop the Mark5A program.
`Clear`	reset the mk5daemon; useful sometimes if `mpifxcorr` crashes.
`Reboot`	causes machine to reboot.
`Poweroff`	causes machine to power down.

11 Installation and upgrade guide

This is a module-by-module installation guide that takes one through the installation of the NRAO adaptation of DiFX. The sections below should be followed more or less in order. Before you begin installing code, you should take a few moments to prepare your environment. First choose a top level source directory, here called *sourcedir*. Also choose an installation top level directory, called *prefixdir*, which should be visible to all the nodes in the cluster. Into this directory, subdirectories such as `bin`, `lib`, `include` will be created containing the installed code from the many packages you will need. At this time four environment variables need creation or expansion:

1. `IPPROOT`: §11.2. Set this to something trivial (such as `.`) until IPP has been installed, then remember to change it as appropriate.
2. `LD_LIBRARY_PATH`, a standard environment variable containing a dynamic library search path. Add *prefixdir*`/lib` and `$IPPROOT/sharedlib` to this path.
3. `PATH`, a standard environment variable containing the execution path. Add *prefixdir*`/bin` to this path.
4. `PKG_CONFIG_PATH`, a search path for package installation information. Add *prefixdir*`/lib/pkgconfig` to this path.

Note that all of these environment variables (in addition to those described in §4) are required at run-time as well as compile-time, so it is advisable to put these path commands into your shell initialization file and start a new shell at this point. Note that these variables will be needed not only in interactive shells, but also non-interactive ones, so be sure that these are set no matter how the shell is invoked.

To download, compile and install the software, you will need the standard gnu tools (gcc, libtool, autoconf, automake, make, ...), python, subversion, and of course ssh. Be aware that many distributions don't install by default all of these needed tools (xubuntu for example installs very few development tools by default. Relatively few external libraries are used. It also assumes you have an account that allows access to the subversion repository at `https://svn.atnf.csiro.au`.

The `make install` steps may require root permission, depending on the *prefixdir* you have chosen. If so, become root before each `make install`. It is advisable not to compile code as root. Be wary of errors along the way; occasional warnings may be issued, but if the building proceeds, things are probably okay. Please report any build issues to `wbrisken@nrao.edu`. Be warned that these instructions may change.

1.1 All of the subversion repositories below point to a `NRAO-DiFX-1.1` branch of the repository. This is in order to provide a relatively stable source tree that allows development to continue on the main development branch (called `trunk`). In order to check out code that is on this development branch, simply replace `branches/NRAO-DiFX-1.1` with `trunk` in all of the `svn` commands below. *Caveat emptor:* the `trunk` branch code may at any time refuse to compile, be unstable, lack documentation, or produce incorrect results. Don't let this stop you if you are an intrepid developer or want to see ongoing development in progress!

If you do not have SVN access, `.tar.gz` files are available from `http://www.aoc.nrao.edu/~wbrisken/NRAO-DiFX-1.1/`.

11.1 OpenMPI

The core of DiFX uses Message Passing Interface (MPI) for inter-node communication. Many MPI libraries exist; we choose to use OpenMPI as it is simple to install, runs well, and appears to have good community support.

1. Download the latest source distribution from `http://www.open-mpi.org/` (ver. 1.2.7 as of this writing)
2. Decompress the contents into perhaps *sourcedir*; enter the newly created directory
3. `./configure --prefix=`*openmpiprefix* where *openmpiprefix* could be the same as *prefixdir*, but does not have to be.
4. Run `make` and finally `make install` to put the parts where they belong.

11.2 Intel Performance Primitives

Intel CPUs support an increasing variety of vector math instructions. The Intel Performance Primitives (IPP) makes exploiting these capabilities on any recent generation CPU simple. An inexpensive license must be purchased to make use of these. More information can be found on `http://www.intel.com`.

Once installed, set environment variable `IPPROOT` to point to its install prefix, which will look something like: `/opt/intel/ipp/5.3.3.075/ia32`; you want to choose the directory containing `lib`, `include`, etc. Remember to change this in your shell initialization file as well. This install directory should be visible to all nodes in the cluster.

11.3 FFTW

The FFTs performed by `mpifxcorr` are done using the Intel Performance Primitives library, but FFTs done in an optional piece of`difx2fits` and the utility `m5spec` that comes with mark5access use FFTW, a standard, fast, freely available FFT library. This library is probably installed for you with any modern Linux distribution, but you should check to make sure it is recent enough; version 3.0 and up are supported, but version 3.1.2 or newer is recommended. If this library is not installed and the extra functionality that requires FFTW is not installed, follow the instructions below:

1. Download the latest source distribution from `http://www.fftw.org` (ver. 3.1.2 as of this writing)
2. Decompress the contents into perhaps *sourcedir*; enter the newly created directory
3. `./configure --prefix=`*prefixdir*
4. Run `make` and finally `make install` to put the parts where they belong.

11.4 difxio

Parsing of text files can be tedious. The library difxio makes parsing difx-style files simple. It also contains functionality to completely represent the configuration of a DiFX correlation, simplifying format conversions. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/common/difxio/branches/NRAO-DiFX-1.1 difxio`
   Note: don't forget the `difxio` *at the end of the line!*
3. Enter the new directory `cd difxio`
4. View the README file. Note the next 5 instructions only need to be done once in this directory, even after updating the repository. You can `man` the commands if you want to know what they do.
5. `aclocal`
6. `libtoolize --copy --force`
7. `autoconf`
8. `autoheader`
9. `automake -a`
10. Generate the Makefile: `./configure --prefix=`*prefixdir*
11. Build it: `make`
12. Install it: `make install`

You can test for successful installation by running `pkg-config --cflags difxio`. If you get a sensible answer, things are probably good. If you wish to upgrade the installation:

1. `cd` *sourcedir*`/difxio`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

Note that doing this upgrade may break other packages that depend on it, such as `difx2fits` and `calcif`, forcing a recompile of these programs.

11.5 difxmessage (optional) 1.1

The library difxmessage implements in the C language XML generation and parsing and multicast sending and receiving functionality that is used for communication between various parts of the NRAO-DiFX system. See §10 for details on the XML documents supported. The communication model is based on that of the EVLA. This package is optional; if not built, you will not be able to use `mk5daemon` or any program packaged with it, or `genmachines`. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/common/difxmessage/branches/NRAO-DiFX-1.1 difxmessage`

3. Enter the new directory `cd difxmessage`
4. View the README file. Note the next 5 instructions only need to be done once in this directory, even after updating the repository. You can `man` the commands if you want to know what they do.
5. `aclocal`
6. `libtoolize --copy --force`
7. `autoconf`
8. `autoheader`
9. `automake -a`
10. Generate the Makefile: `./configure --prefix=`*prefixdir*
11. Build it: `make`
12. Install it: `make install`

You can test for successful installation by running `pkg-config --cflags difxmessage`. If you get a sensible answer, things are probably good. If you wish to upgrade the installation:

1. `cd` *sourcedir*`/difxio`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

Note that doing this upgrade may break other packages that depend on it, such as `mpifxcorr` and `mk5daemon`, forcing a recompile of these programs.

11.6 mark5access

mark5access is a library to parse various VLBI baseband data formats, including Mark4, VLBA, and Mark5B, with other formats to be added. This is needed to decode these various formats from within mpifxcorr. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/common/mark5access/branches/NRAO-DiFX-1.1 mark5access`
3. Enter the new directory `cd mark5access`
4. View the README file. Note the next 5 instructions only need to be done once in this directory, even after updating the repository.
5. `aclocal`
6. `libtoolize --copy --force`
7. `autoconf`
8. `autoheader`
9. `automake -a`

10. Generate the Makefile: `./configure --prefix=`*prefixdir*
11. Build it: `make`
12. Install it: `make install`

You can test for successful installation by running `pkg-config --cflags mark5access`. If you get a sensible answer, things are probably good. If you wish to upgrade the installation:

1. `cd` *sourcedir*`/mark5access`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

Note that doing this upgrade may break other packages that depend on it, such as `mpifxcorr`, forcing a recompile of these programs.

11.7 mpifxcorr

The core of the DiFX software correlator is `mpifxcorr`. Installation and running this program requires that MPI (§11.1), IPP (§11.2), difxio and mark5access all be installed. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/mpifxcorr/branches/NRAO-DiFX-1.1 mpifxcorr`
3. Enter the new directory `cd mpifxcorr`
4. View the README file. Note the next 4 instructions only need to be done once in this directory, even after updating the repository.
5. `aclocal`
6. `autoconf`
7. `autoheader`
8. `automake -a`
9. Generate the Makefile: `./configure --prefix=`*prefixdir* `CXX=`*openmpiprefix*`/bin/mpicxx`
10. Build it: `make`
11. Install it: `make install`

If successfully installed, the command `which mpifxcorr` should return *prefixdir*/bin/mpifxcorr. If you wish to upgrade the installation:

1. `cd` *sourcedir*`/mpifxcorr`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

11.8 calcserver

The Goddard Space Flight Center CALC package version 9.1 is used to calculate the delay models needed for time-alignment of the raw data. This software is wrapped in a program that exposes the capabilities of CALC via a Remote Procedure Call (RPC) and this program runs as a server. An environment variable `CALC_SERVER` should be set that contains the name of the computer running `calcserver`. Within NRAO-DiFX, the only program that makes use of this server is `calcif2` (§8.2). To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/utilities/calcserver/branches/NRAO-DiFX-1.1 calcserver`
3. Enter the new directory `cd calcserver`
4. View the README file.
5. `aclocal`
6. `libtoolize --copy --force`
7. `autoconf`
8. `automake -a`
9. Generate the Makefile: `./configure --prefix=`*prefixdir*
10. Build it: `make`
11. Install it: `make install`

11.9 job2difx

The `job2difx` package contains several programs that are useful for DiFX input file creation and managing correlation. These programs are designed to work together with others in the NRAO customized software correlator system. The `calcif` program which creates `.uvw`, `.delay`, and `.rate` files from an input file (the `.calc` file – generated by job2difx) is also contained in this package. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/utilities/job2difx/branches/NRAO-DiFX-1.1 job2difx`
3. Enter the new directory `cd job2difx`
4. View the README file. Note the next 4 instructions only need to be done once in this directory, even after updating the repository.
5. `aclocal`
6. `autoconf`
7. `autoheader`
8. `automake -a`
9. Generate the Makefile: `./configure --prefix=`*prefixdir*
10. Build it: `make`

11. Install it: `make install`

If successfully installed, the command `which job2difx` should return *prefixdir*/bin/job2difx. Several other programs should also be installed, including: `calcif`, `genmachines`, `getjobs`, `jobdisks`, `joblist`, `jobstatus`, `difxsniff`, `mk5take`, `mk5return`, and `vlog`. If you wish to upgrade the installation:

1. `cd` *sourcedir*/`job2difx`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

11.10 difx2fits

The initial NRAO adaptation of DiFX is designed to interface as seamlessly as possible into our existing infrastructure and habits. This means generation of FITS-IDI output files for compliance with AIPS. The program `difx2fits` takes many input files (see Fig. 1) and produces a FITS file for every DiFX input file. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/utilities/difx2fits/branches/NRAO-DiFX-1.1 difx2fits`
3. Enter the new directory `cd difx2fits`
4. View the README file. Note the next 4 instructions only need to be done once in this directory, even after updating the repository.
5. `aclocal`
6. `autoconf`
7. `autoheader`
8. `automake -a`
9. Generate the Makefile: `./configure --prefix=`*prefixdir*
10. Build it: `make`
11. Install it: `make install`

If successfully installed, the command `which difx2fits` should return *prefixdir*/bin/difx2fits. If you wish to upgrade the installation:

1. `cd` *sourcedir*/`difx2fits`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

11.11 mk5daemon (optional) 1.1

The optional package `mk5daemon` relies on package `difxmessage` and is only really needed for installations requiring playback off Mark5 modules. Root permission is required for proper installation and running of this program. See §8.15 for a description of the main program, `mk5daemon`, that comes with this package. Other useful scripts are included here. To install:

1. `cd` *sourcedir*
2. Check out the subversion repository:
   `svn co https://svn.atnf.csiro.au/difx/utilities/mk5daemon/branches/NRAO-DiFX-1.1 mk5daemon`
3. Enter the new directory `cd mk5daemon`
4. View the README file. Note the next 4 instructions only need to be done once in this directory, even after updating the repository.
5. `aclocal`
6. `autoconf`
7. `autoheader`
8. `automake -a`
9. Generate the Makefile: `./configure --prefix=`*prefixdir*
10. Build it: `make`
11. Install it: `make install`
12. Ensure that this program starts at boot. This requires the following to occur on each computer in the cluster. Note that the instructions may vary depending on your operating system. The program is likely to get started before NSF is started, so `mk5daemon` should be installed locally on each compute in the cluster. On each machine, run as root: `cp` *prefixdir*`/bin/mk5daemon` *localdir* `; echo` *localdir*`/mk5daemon >> /etc/rc.local` . Here *localdir* is a directory on the particular machine, such as `/usr/bin` This only needs to be run

If successfully installed, the command `which mk5daemon` should return *prefixdir*/bin/mk5daemon. If you wish to upgrade the installation:

1. `cd` *sourcedir*`/mk5daemon`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`
5. Copy it to the local disk (as root): `cp -f` *prefixdir*`/bin/mk5daemon` *localdir*

12 Acknowledgements

Many people have helped in significant ways to get NRAO-DiFX where it is today: Craig West for getting me interested in software correlators to begin with; Adam Deller for writing DiFX and helping adapt it to the needs of NRAO-DiFX; Steve Tingay and Matthew Bailes for allowing/encouraging Adam to write DiFX and helping support some of my travel in Australia; Chris Phillips for contributing code, bug reports, bug fixes, and ideas; Walter Alef for hosting the first DiFX workshop in 2007; Miguel Guerra for helping define the XML structures and his work on the upcoming operator interface; Claire Chandler, Jon Romney and Craig Walker for project oversight, advice, and voices of reason; Steven Durand for buying me computer and Mark5 equipment; David Boboltz, Mark Claussen, Vivek Dhawan, Alan Fey, Vincent Fish, Ed Fomalont, Miller Goss, Amy Mioduszewski, John Morgan, Roopesh Ojha, Loránt Sjouwerman and Craig Walker for working directly with or examinating the output of NRAO-DiFX and providing valuable feedback; Eric Greisen for making AIPS work well with NRAO-DiFX output; John Benson for working with me to get data into the VLBA archive; David Halstead and James Robnett for useful discussions on clustering; Doug Gerrard, Bob McGoldrick, Adrian Rascon and K. Scott Rowe for assembling and maintaining the correlator; Juan Cordova, Paul Dyer, Lisa Foley, Heidi Gerhardt, Ken Hartley, Alan Kerr, Jim Ogle, Paul Padilla, Peggy Perley, Tony Perreault, Betty Ragan, Meri Stanley and Anthony Sowinski for providing operations assistance; and Emma Goldberg for helping me work around the idiosyncrocies of LaTeX and convince it to typeset this document. If I left anyone of this list that should be there, and there are probably several of you in that catagory, I apologize — let me know and I'll make sure to add you for the next version's document.

13 Contact information

Please contact Walter Brisken <`wbrisken@nrao.edu`> for any problems with the documentation or software described in the manual.

References

[1] *DiFX: A software correlator for very long baseline interferometry using multi-processor computing environments*, Deller, A. T., Tingay, S. J., Bailes, M. & West, C., 2007, PASP 119, 318.

[2] *Summary of the b-factor for the VLBA FX correlator*, Kogan, L., VLBA Scientific Memo 12.

[3] *The FITS Interferometry Data Interchange Format*, Flatters, C., AIPS Memo 102.

[4] *The FITS Interferometry Data Interchange Convention*, Greisen, E., AIPS Memo 113 *in prep.*

[5] *Tempo*, `http://www.atnf.csiro.au/research/pulsar/tempo/`

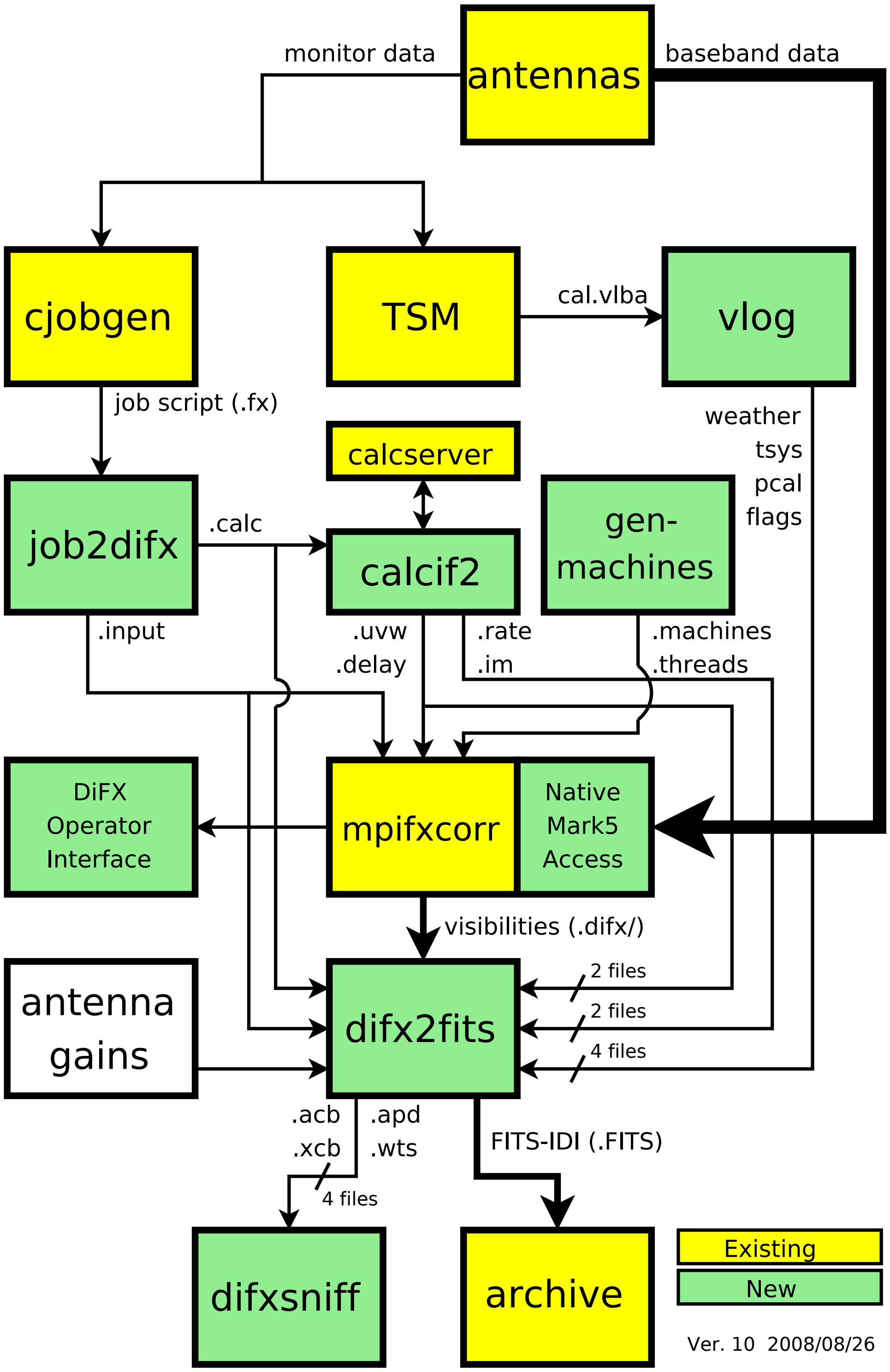

Figure 1: *The NRAO-DiFX software correlator block diagram as implemented for the VLBA*

```
/home/swc/  difx root directory: a 1.5 TB partion on swc000
├ difx/  all data related to operations ends up under this directory
│ ├ archive/  staging area for data going to archine
│ ├ directories/  cached Mark5 module directories; environment variable
│ │ │              MARK5_DIR_PATH points here
│ │ ├ NRAO-120.dir
│ │ ├ NRAO-123.dir
│ │ └ VIPSU-05.dir
│ ├ gaincurves/  directory of symlinks to gain curve files; environment
│ │ │             variable GAIN_CURVE_PATH points here
│ │ └ gain.ar  symlink to /home/jansky3/vlbaops/TCAL/gain.ar
│ ├ projects/  active project data; environment variable JOB_ROOT
│ │ │           points here
│ │ ├ bc120e/
│ │ └ mt831/
│ └ tests/  active test data; environment variable TESTS points here
├ NRAO-DiFX-1.1/  version 1.1 files; set prefix to this during installation
│ ├ bin/  ver 1.1 programs
│ │ ├ calcif2
│ │ ├ job2difx
│ │ ├ mpifxcorr
│ │ └ startdifx
│ ├ lib/  libraries for ver 1.1
│ ├ setup_difx  script to set environment for version 1.1
│ └ src/  frozen source code for ver 1.1
│   ├ calcserver/
│   ├ difx2fits/
│   ├ difxio/
│   ├ difxmessage/
│   ├ job2difx/
│   │ └ calcif/
│   ├ mark5access/
│   ├ mk5daemon/
│   ├ mpifxcorr/
│   └ openmpi-1.2.7/
└ NRAO-DiFX-trunk/  development branch
  ├ bin/  programs
  ├ lib/  libraries
  ├ setup_difx  script to set environment for development version
  └ src/  source
```

Figure 2: *The directory structure of the VLBA software correlator.* This figure showing the organization of the NRAO-DiFX directory structure is not comprehensive, but is hopefully complete enough to illustrate the general layout. For example, only the major top level subdirectories of `NRAO-DiFX-trunk` are shown. Entries ending in '/' are themselves directories.