

# A Guide to the VLBA DiFX Correlator

## Version 2.3

*Walter Brisken*

National Radio Astronomy Observatory

March 3, 2014

## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Notation . . . . .	6
<b>2</b>	<b>The DiFX correlator</b>	<b>6</b>
2.1	NRAO-DiFX 1.0 . . . . .	7
2.2	NRAO-DiFX 1.1 . . . . .	7
2.2.1	Bugs fixed . . . . .	8
2.2.2	Known problems . . . . .	8
2.3	DiFX 1.5.0 . . . . .	8
2.3.1	Bugs fixed . . . . .	9
2.4	DiFX 1.5.1 . . . . .	9
2.4.1	Bugs fixed . . . . .	9
2.5	DiFX 1.5.2 . . . . .	10
2.5.1	Bugs fixed . . . . .	10
2.5.2	Known problems . . . . .	11
2.6	DiFX 1.5.3 . . . . .	11
2.6.1	Bugs fixed . . . . .	12
2.7	DiFX 1.5.4 . . . . .	13
2.7.1	Bugs fixed . . . . .	14
2.8	Known bugs . . . . .	14
2.9	DiFX 2.0.0 . . . . .	14
2.9.1	New features . . . . .	14
2.10	Known bugs . . . . .	14
2.11	DiFX 2.0.1 . . . . .	14
2.11.1	New features . . . . .	15
2.11.2	Bug fixes . . . . .	15
2.12	DiFX 2.1 . . . . .	15
2.12.1	New features . . . . .	15
2.12.2	Bug fixes . . . . .	16
2.13	DiFX 2.1.1 . . . . .	17
2.14	DiFX 2.2 . . . . .	17
2.14.1	New features . . . . .	17
2.14.2	Bug fixes . . . . .	17
2.15	DiFX 2.3 . . . . .	18
2.15.1	New features . . . . .	18
2.15.2	Bug fixes . . . . .	18
2.16	Features left to implement . . . . .	18
2.17	DiFX and AIPS . . . . .	19

<b>3</b>	<b>Hardware and operating system configuration</b>	<b>19</b>
3.1	Cluster configuration . . . . .	19
3.2	Environment variables . . . . .	20
3.3	Directory structure and versioning . . . . .	21
<b>4</b>	<b>Running DiFX at the VLBA correlator (VLBA Specific)</b>	<b>22</b>
4.1	Software correlation based on vex files . . . . .	22
4.1.1	Directory contains undecoded scans . . . . .	24
4.1.2	Directory read fails on partial module . . . . .	24
4.1.3	Mark5 unit hangs while reading directory . . . . .	24
4.1.4	Mark5 directory reading fails partway through . . . . .	24
4.1.5	Mark5 unit hung . . . . .	26
4.1.6	Module moved . . . . .	26
4.2	Correlating data files . . . . .	26
4.3	The VLBA database . . . . .	26
4.4	Archiving . . . . .	28
4.5	Ownership and permissions of files . . . . .	28
4.6	Copying baseband data . . . . .	29
4.6.1	Performing the data copy . . . . .	29
4.6.2	Guidelines to users . . . . .	29
4.7	Some comments on channels . . . . .	30
<b>5</b>	<b>DiFX and pulsars</b>	<b>30</b>
5.1	Pulse ephemeris . . . . .	30
5.2	Bin configuration file . . . . .	31
5.2.1	Binary gating . . . . .	31
5.2.2	Matched-filter gating . . . . .	31
5.2.3	Pulsar binning . . . . .	32
5.3	Preparing correlator jobs . . . . .	32
5.4	Making FITS files . . . . .	32
<b>6</b>	<b>Conventions</b>	<b>33</b>
6.1	Clock offsets and rates . . . . .	33
6.2	Geometric delays and rates . . . . .	33
6.3	Antenna coordinates . . . . .	33
6.4	Baseline coordinates . . . . .	33
6.5	Visibility phase . . . . .	34
<b>7</b>	<b>Reference guide to programs and utilities</b>	<b>34</b>
7.1	calcif2 . . . . .	34
7.2	CalcServer . . . . .	35
7.3	checkmpifxcorr (package : mpifxcorr) . . . . .	36
7.4	checkdir (package : mk5daemon) . . . . .	36
7.5	condition (package : difx_db) . . . . .	36
7.6	condition_watch (package : difx_db) . . . . .	37
7.7	cpumon (package : difxmessage) . . . . .	37
7.8	db2vex (package : difx_db) . . . . .	38
7.9	diffDiFX.py (package : vis2screen) . . . . .	39
7.10	difx2fits . . . . .	39
7.11	difx2mark4 . . . . .	41
7.12	difxarch (package : difx_db) . . . . .	42
7.13	difxbuild . . . . .	42

7.14	difxcalculator (package : difxio)	42
7.15	difxclean (package : difx_db)	43
7.16	difxcopy (package : difx_db)	43
7.17	difxlog (package : difxmessage)	44
7.18	difxqueue (package : difx_db)	44
7.19	difxsniff (package : calcif2)	46
7.20	difxspeed (package : vex2difx)	46
7.21	difxusage (package : difx_db)	46
7.22	DiFX Operator Interface	47
7.23	e2ecopy (package : difx_db)	48
7.24	errormon (package : difxmessage)	48
7.25	genmachines (package : calcif2)	48
7.26	getshelf (package : difx_db)	49
7.27	jobdisks (package : calcif2)	49
7.28	joblist (package : calcif2)	50
7.29	jobstatus (package : calcif2)	51
7.30	listcpus (package : calcif2)	52
7.31	makefits (package : difx2fits)	52
7.32	makemark4 (package : difxdb)	53
7.33	m5bstate (package : mark5access)	53
7.34	m5d (package : mark5access)	53
7.35	m5findformats (package : mark5access)	54
7.36	m5fold (package : mark5access)	54
7.37	m5pcal (package : mark5access)	55
7.38	m5spec (package : mark5access)	56
7.39	m5test (package : mark5access)	56
7.40	m5time(package : mark5access)	57
7.41	m5timeseries (package : mark5access)	57
7.42	m5tsys (package : mark5access)	57
7.43	mk5cat (package : mk5daemon)	57
7.44	mk5control (package : mk5daemon)	58
7.45	mk5cp (package : mk5daemon)	59
7.46	mk5daemon (package : mk5daemon)	59
7.47	mk5dir (package : mark5daemon)	61
7.48	mk5erase (package : mark5daemon)	61
7.49	mk5mon (package : difxmessage)	62
7.50	mpifxcorr	63
7.51	oms2v2d (package : vex2difx)	63
7.52	padVDIF (package : vdifio)	64
7.53	plotapd (package : SniffPlots)	64
7.54	plotbp (package : SniffPlots)	64
7.55	plotwt (package : SniffPlots)	64
7.56	printDiFX.py (package : vis2screen)	64
7.57	printVDIF (package : vdifio)	65
7.58	recover (package : mk5daemon)	65
7.59	startdifx (package : calcif2)	66
7.60	statemon (package : difxmessage)	66
7.61	stopmpifxcorr (package : mpifxcorr)	67
7.62	stripVDIF (package : vdifio)	67
7.63	testdifxinput (package : difxio)	67
7.64	testdifxmessagereceive(package : difxmessage)	68

7.65	testmod (package : mk5daemon)	68
7.66	testseqnumbers (package : difxmessage)	69
7.67	vdifbstate (package : vdifio)	69
7.68	vdifd (package : vdifio)	70
7.69	vdiffold (package : vdifio)	70
7.70	vdifspec (package : vdifio)	71
7.71	vex2difx	72
	7.71.1 VDIF issues	73
	7.71.2 Mark5B issues	73
	7.71.3 Media specification	74
	7.71.4 Pulsars	74
7.72	vexpeek (package : vex2difx)	75
7.73	vlog (package : calcif2)	75
7.74	vmux (package : vdifio)	75
7.75	vsn (package : mk5daemon)	76
7.76	vsum (package : vdifio)	77
7.77	zerocorr (package : mark5access)	77
<b>8</b>	<b>Description of various files</b>	<b>77</b>
8.1	.acb	78
8.2	.apc	78
8.3	.apd	79
8.4	.binconfig	79
8.5	.bootstrap	80
8.6	cal.vlba	81
8.7	.calc	81
8.8	.difx/	84
	8.8.1 Visibility files	84
	8.8.2 Pulse cal data files	85
	8.8.3 Switched power files	85
8.9	.difxlog	85
8.10	.speed	86
8.11	.speed.out	86
8.12	\$DIFX_MACHINES	87
8.13	.dir	87
8.14	.filelist	87
8.15	.FITS	88
8.16	.fitslist	88
8.17	.flag	89
8.18	flag	89
8.19	.im	89
8.20	.input	90
	8.20.1 Common settings table	91
	8.20.2 Configurations table	91
	8.20.3 Rule table	92
	8.20.4 Frequency table	92
	8.20.5 Telescope table	92
	8.20.6 Datastream table	93
	8.20.7 Baseline table	94
	8.20.8 Data Table	94
8.21	.joblist	94
8.22	.jobmatrix	95

8.23	.lag	95
8.24	.log	95
8.25	.machines	96
8.26	.mark4list	96
8.27	.oms	97
8.28	.params	97
8.29	pcal	97
8.30	.polyco	97
8.31	.shelf	98
8.32	.threads	98
8.33	tsys	98
8.34	weather	98
8.35	.wts	99
8.36	.v2d	99
8.37	.xcb	105
8.38	.vex, .skd, .vex.obs, & .skd.obs	106
8.39	.vis	106
8.40	.zerocorr	107
<b>9</b>	<b>XML message types</b>	<b>107</b>
9.1	DifxAlertMessage	109
9.2	DifxCommand	109
9.3	DifxLoadMessage	110
9.4	DifxParameter	110
9.5	DifxSmartMessage	111
9.6	DifxTransientMessage	112
9.7	DifxStart	112
9.8	DifxStatusMessage	113
9.9	DifxStop	114
9.10	Mark5DriveStatsMessage	114
9.11	Mark5StatusMessage	115
9.12	Mark5VersionMessage	116
<b>10</b>	<b>Database tables</b>	<b>117</b>
10.1	The DIFXQUEUE table	117
10.2	The DIFXLOG table	118
10.3	The CONDITION table	118
<b>11</b>	<b>DiFX alert messages</b>	<b>119</b>
11.1	Fatal	119
11.2	Severe	122
11.3	Error	123
11.4	Warning	127
11.5	Info	129
11.6	Verbose	129
11.7	Debug	129
<b>12</b>	<b>Installation and upgrade guide</b>	<b>129</b>
12.1	Installation with difxbuild	130
12.1.1	Introduction	130
12.1.2	Assumptions	130
12.1.3	Installation Part 1	130

12.1.4	Set this version of DiFX as the default version	132
12.1.5	Installation Part 2	132
12.1.6	Installation Part 3	132
12.1.7	Installation Part 4	132
12.1.8	Upgrading the installation	133
12.2	Manual installation	133
12.2.1	expat	133
12.2.2	cx_Oracle	134
12.2.3	OpenMPI	134
12.2.4	Intel Performance Primitives	134
12.2.5	FFTW	135
12.2.6	difxio	135
12.2.7	difxmessage (optional)	136
12.2.8	mark5access	136
12.2.9	mpifxcorr	137
12.2.10	calcservice	138
12.2.11	calcif2	138
12.2.12	difx2fits	139
12.2.13	mk5daemon (optional)	140
12.2.14	vex2difx	141
12.2.15	difx_db (NRAO only)	141

**13 Acknowledgements** **142**

# 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 DiFX, coexistence issues with the VLBA hardware correlator, explanation of various file/document types, and detailed installation instructions. This manual covers DiFX version 2.3 and periodically will be updated to current with DiFX improvements. Please report any errors that are found in this manual to [wbrisken@nrao.edu](mailto: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 DiFX correlator

This document is centered around the NRAO installation of the DiFX [2] software correlator and its supporting software. 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. Fig. 1 shows the general data flow-path within the DiFX software correlator system.

Past, present, and future versions of DiFX as packaged and used by VLBA operations are described in the following subsections.

## 2.1 NRAO-DiFX 1.0

Versions 1.0 and 1.1 based correlation on the VLBA hardware correlator job scripts – the `.fx` files. 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 (Mark4 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. 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.

### 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 Mark4 format decoder had a 1 day off bug.
5. The Mark4 format decoder had a  $64\times$  *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.

## 2.3 DiFX 1.5.0

With DiFX 1.5.0 comes a name change. Past releases of this series have been known as “NRAO-DiFX”. The DiFX community has been largely receptive to the NRAO additions in support of `mpifxcorr` and it was decided that dropping the “NRAO” was appropriate. In some cases the term “VLBA DiFX” or “VLBA DiFX 1.5” may be mentioned. These are simply the deployment of DiFX 1.5.0 for the VLBA correlator with some VLBA specific features. Note that the name given to the VLBA deployment of DiFX is formally “VLBA DiFX”.

Version 1.5.0 will start allowing correlation of experiments that cannot be represented by `.fx` files and will be based on `vex` files. Version 1.5.0 builds on version 1.1 and adds the following features:

1. Support for using a wide variety of `vex` files as the basis for correlation.
2. Native ephemeris-based object trajectories are supported.
3. Pulsar gating is supported.
4. Pulsar binning is supported, but not cleanly yet.
5. A graphical user interface is available for correlator operators.
6. The multicast system is fully implemented and is used monitor and control correlation and other operations.
7. Mark5B formatted data, including its 2048 Mbps extension, is supported.



8. The VLBA DiFX Operations Plan [1] is implemented, including interface to the VLBA archive.

Non-NRAO users of DiFX 1.5.0 will still be able to use the tools provided but may not be able to take full advantage of the database back-end without some customization; it is the aim of this document to point out cases where the database is required. Many of the programs described in previous versions of this document will be upgraded or overtaken by more capable replacements.

Release of DiFX 1.5.0 was announced on June 25, 2009.

### 2.3.1 Bugs fixed

Here are listed some of the more important bug fixes:

1. A rounding issue in `mpifxcorr` occasionally caused the wrong source's UVWs to be assigned.
2. Lower side band data would come out of the sniffer portion of `difx2fits` with the wrong sign for phase, rate, and delay.
3. Different rounding was used to generate start times for `.input` and `.calc` files. There are no severe consequences of this issue.
4. Scaling in pulsar gating has been made more sane.

## 2.4 DiFX 1.5.1

DiFX 1.5.1 is mostly a bug fix update to version 1.5.0, but with a few new features. The new features include:

1. Option to force job breaks (with the `break` parameter) has been added to `vex2difx`
2. Time/date formats other than decimal MJD are now accepted by `vex2difx`
3. Specification of data files to correlate (rather than Mark5 units) is supported in `vex2difx`
4. Specification of network parameters in `vex2difx` to allow correlation of eVLBI projects
5. `difx2fits` produces a new output file with suffix `.jobmatrix` provides the user with a better idea of the mapping of jobs into `.FITS` files
6. A `vex2difx` mode for generating DiFX files useful for determining pulsar phase has been added
7. EOP values can now be provided within the `.v2d` file
8. Upcoming FITS-IDI keyword `WEIGHTTYP` populated
9. Zero-weight data is not written from `mpifxcorr`
10. New utility `checkdir` to look for oddities in Mark5 module directory files

### 2.4.1 Bugs fixed

Here are listed some of the more important bug fixes:

1. Concatenation of jobs in the creation of `.FITS` files does the right thing for cases where the antenna subsets change and where antenna reordering is done.
2. The Pulsar Gate Model (GM) `.FITS` file table is now correctly populated for pulsar observations.
3. Autocorrelations are written for each pulsar bin

4. The FXCORR simulator mode of `vex2difx` now selects the correct reference time for antenna clock offsets.
5. A work-around for a Streamstor problem has been added that should improve reliability in Mark5 module correlation when a change in bank is needed.
6. The sign of clock offsets in vex files has been reversed to follow the vex standard
7. Jobs are split at leap seconds
8. LBA data formats are handled more correctly in `vex2difx`
9. The model generator (`calcif2`) now respects polynomial parameters interval and order given on the command line.

DiFX 1.5.1 was made available via subversion on Sep 8, 2009.

## 2.5 DiFX 1.5.2

DiFX 1.5.2 is mostly a bug fix update to version 1.5.1, but with a few new features. This version of DiFX comes with the following components (and versions): `calcif2` (1.1), `calserver` (1.2), `difx_db` (1.12; NRAO-only), `difx2fits` (2.6.1), `difx2profile` (0.1), `difxio` (2.12.1), `difxmessage` (0.7), `mark5access` (1.3.3), `mk5daemon` (1.2), `mpifxcorr` (1.5.2), `vex2difx` (1.0.2), and `vis2screen` (0.1). The new features include:

1. Support unmodulated VLBA format data with new pseudo-format “VLBN”
2. `mpifxcorr` now warns when `difxmessage` is in use so the user knows why no messages appear on the screen
3. New utility `difxcalculator` in the `difxio` package
4. eVLBI support within `vex2difx`
5. Vastly improved real-time correlation monitoring
6. New utility `diffDiFX.py` to compare two DiFX output files
7. Improved and more consistent error messages (and some of them are now documented!)
8. `vex2difx` now operates in strict parsing mode by default
9. Additional user feedback to indicate suspicious or bad `.polyco` and `.v2d` files
10. `calcif2` warns if any NaNs or Infs are produced
11. Clock adjustments are easier now with `deltaClock` and `deltaClockRate` parameters in the `vex2difx` antenna settings

### 2.5.1 Bugs fixed

1. Improve timestamp precision (thanks to John Morgan)
2. The `vlog` program (used at NRAO only, I think) misparsed the pulse cal information in some cases
3. Fixed memory leak in `difx2fits` when combining a large number of jobs
4. Improved FXCORR simulation mode in `vex2difx`
5. Mark5 directory reading systematically generates unique names for all scans even when two scans have the same name

6. Improve reporting of Mark5 errors during playback and change alert severity to be more appropriate
7. Don't overblank certain Mark4 modes (thanks to Sergei Pogrebenko for the bug report)
8. Vex 'data valid' period now properly respected
9. Vex clock table tolerance issue corrected
10. Changes in Mark5 mode should be safer (note that currently `vex2difx` never exercises multiple modes in a single job)
11. When making the cross spectrum sniffer plots, respect the reference antenna
12. Improved pulsar polynomial file error checking is performed
13. Amplitude-phase-delay (APD) sniffer plots always have refant first when multiple refants are supplied
14. Project name should now appear on sniffer APD plots
15. Mark5 units now send status information even when no playback is occurring (eliminating the incorrect LOST state issue as displayed in the DOI)

### 2.5.2 Known problems

1. Extensive use in VLBA operations has shown that occasional data dropouts of one or more antenna, sometimes in a quasi-repeatable manner, affect completeness of some jobs. It is not clear exactly what the cause is at this point, however its cure is a high priority.
2. Loss of a few FFTs of data will occur in rare circumstances.
3. Clock accountability is poor when jobs containing multiple clock models for antennas are combined.

DiFX 1.5.2 was made available via subversion on Jan 20, 2010.

## 2.6 DiFX 1.5.3

DiFX 1.5.3 is mainly intended as a bug fix update to version 1.5.2, though some new features have made their way into the codebase. This version of DiFX comes with the following components (and versions): `calcif2` (1.3), `calcserver` (1.3), `difx_db` (1.13; NRAO-only), `difx2fits` (2.6.2), `difx2profile` (0.2), `difxio` (2.12.2), `difxmessage` (7.2), `mark5access` (1.3.4), `mk5daemon` (1.3), `mpifxcorr` (1.5.3), `vex2difx` (1.0.3), and `vis2screen` (0.2). Many changes are motivated by issues found running DiFX full time in Socorro.

The new features include:

1. Mark5 directory (`.dir`) files can contain RT on the top line to indicate the need to play back using *Real-Time* mode.
2. `difxqueue` (NRAO only) now takes an optional parameter specifying the staging area to use.
3. New Mark5 diagnostic programs (`vsn` and `testmod`) introduced to wean off the use of the `Mark5A` program.
4. `mk5daemon` can now mount and dismount USB and eSATA disks through `mk5commands`.
5. `mk5cp` now makes the destination directory if it doesn't exist.
6. `mk5daemon` will now warn if free disk space is getting low.
7. `db2vex` (NRAO only) now allows field station logs to be provided. As of now, only media VSNs are extracted.

8. Playback off Mark5 units has been made more robust with better error reporting.
9. New utility `m5fold` that can be used to look at repeating signals in baseband data total power (e.g., switched power)
10. `vex2difx` now supports job generation in cases where upper side band was observed at one antenna and lower sideband at another.

### 2.6.1 Bugs fixed

1. Don't unnecessarily drop any FFTs of data.
2. Sub-integrations longer than one second could cause integer overflows.
3. Fix bug in `vex2difx` where jobs were not split at clock breaks.
4. `difx2fits` was guilty of incorrect clock accountability after a clock change at a station when merging multiple jobs. Worked around by not allowing such jobs to merge.
5. `db2vex` (NRAO only) warns when more than one clock value is found for an antenna.
6. Mark5 unit bank switches now routinely call `XLRGetDirectory()` to work around a newly discovered bug in the StreamStor software.
7. A couple possible memory leaks in the `mark5access` library were fixed (thanks Alexander Neidhardt and Martin Ettl).
8. Lots of compiler warnings quashed (mostly of the "unused return value" kind).
9. Olaf Wuchnitz found two FITS file writing problems in `difx2fits` dating back to code inherited from FXCORR!
10. Two more digits are retained for the time and one more digit is retained for amplitude information in the `.apd` and `.apc` sniffer files.
11. Some bugs related to replacement of special characters by "entities" in XML messages are fixed.
12. New traps are in place in many places to catch string overruns.
13. Fix for writing `.calc` files with more than one ephemeris driven object.
14. `vex2difx` would get *very* slow due to constantly sorting a list of events. Now this list is only sorted when necessary, drastically speeding it up.
15. The `RCfreqId` parameter in the `difxdatastream` structure (in `difxio`) was used with two different meanings that are normally the same. Cases where they differed caused exceptions. Fixed in `difxio` and `difx2fits`. (Thanks to Randall Wayth for leading to the discovery)
16. `difx2fits` would assign a bogus `.jobmatrix` filename when not running the sniffer.
17. `vex2difx` could get caught in an infinite loop when making jobs where two disk modules had zero time gap.
18. `difx2fits` used a bad config index when making the pulsar GM table when multiple configs were present.
19. Within `mpifxcorr` an extra second was added to the validity period for polycos to ensure no gap in coverage.

20. `mk5dir` would add correct the date improperly for Mark4 formats after beginning of 2010.
21. Lots of fixes for building FITS files out of a subset of baseband recorded channels.
22. FITS files now support antennas with differing numbers of quantization bits.
23. Lots of Mac OS/X build issues fixed.

DiFX 1.5.3 was released on April 16, 2010.

## 2.7 DiFX 1.5.4

DiFX 1.5.4 is likely the last 1.5 series formal release of DiFX, though an additional release could be made if demand is there.

The new features include:

1. `difx2fits` can now produce FITS files with only a subset of the correlated sub-bands.
2. `difx2fits` can be instructed to sniff on an arbitrary timescale.
3. The `makefits` wrapper for `difx2fits` now respects a `-B` option for phase bin selection.
4. `difxio` has improved checking that prevents merging of jobs with incompatible clocks.
5. `difxio` now maintains a separate `clockEpoch` parameter for each antenna.
6. `difxStartMessage` now contains DiFX version to run, allowing queued jobs to be run under different DiFX versions.
7. The curses utilities `mk5mon` and `cpumon` now catch exceptions and can be resized without infecting the terminals they are run in.
8. New sub-library called `mark5ipc` added that provides a semaphore lock for Mark5 units.
9. The `testdifxmessagereceive` utility can now filter on message types.
10. Support for SDK9 throughout (e.g., in `tt mpifxcorr`, `mk5daemon`, and other utilities).
11. Support for new Mark5 module directory formats (Haystack Mark5 memo 81).
12. Several new Mark5 utilities to make up for Mark5A functionality that will not longer be available: `vsn`, `testmod`, `recover`, `m5erase`.
13. `mk5cp` can now copy data based on byte range.
14. Many programs directly talking to the StreamStor card of Mark5 units use `WATCHDOG` macros for improved diagnostics when problems occur.
15. More protection against incomplete polyco files added to `mpifxcorr` (Note: should add this to `vex2difx` as well).
16. The GUI can now spawn different DiFX versions at will through the use of `difxVersion` parameter in the `DifxStartMessage` and wrapper scripts.
17. `difx2fits` can now convert LSB to USB for matching purposed. When used, all LSB sub-bands must have corresponding USB sub-bands on one or more other antenna.
18. `mark5access`-based utilities (e.g., `mp5spec`) can now read from `stdin`.
19. New utility `mk5cat` can send data on a Mark5 module to `stdout`.

### 2.7.1 Bugs fixed

1. Only alt-az telescopes received the correct model. Fixed. Note that CALC and FITS-IDI don't have a good match between their sets of allowed mount types.
2. `difx2fits` now properly propagates quantization bits on a per antenna basis.
3. Logic errors in `difxio` would confuse `difx2fits` in cases where different antennas use different frequency setups. Fixed.
4. Weights are blanked in `difx2fits` prior to populating each record, preventing screwy weights for unused sub-bands.
5. `vex2difx` would sometimes hang or not converge on job generation. Fixed.
6. `vex2difx` now doesn't assume source name is same as vex source def identifier.
7. `mpifxcorr` generated corrupted weights and amplitudes when post-FFT fringe rotation was done. Fixed.

## 2.8 Known bugs

1. Tweak Integration Time feature of `vex2difx` often does the wrong thing.

DiFX 1.5.4 was released on October 12, 2010.

## 2.9 DiFX 2.0.0

DiFX 2.0.0 is based on an upgraded `mpifxcorr` that breaks `.input` file compatibility with the 1.0 series. This new version will allow more flexible correlation of mis-matched bands and correlation at multiple phase centers along with general performance improvements. Development of the 2.0 capabilities will occur in parallel with the 1.0 series features.

### 2.9.1 New features

1. Pulse cal extraction in `mpifxcorr`.
2. Massive multi-phase center capability.
3. New utility `zerocorr` added.
4. External pulse cal extraction utility `m5pca1` added.
5. DiFX output format is all-binary, meaning speed and disk savings

## 2.10 Known bugs

1. Zoom band support has multiple problems.

DiFX 2.0.0 was released on October 12, 2010.

### 2.11 DiFX 2.0.1

DiFX 2.0.1 is a bug fix and clean-up version in response to numerous improvements to DiFX 2.0.0. There are a number of new features as well.

### 2.11.1 New features

1. New utility `checkmpifxcorr` to validate DiFX input files
2. Switched power detection in `mpifxcorr`
3. Early multi-thread VDIF format support
4. RedHat RPM file generation for some packages (can extend to others on request)
5. Improvements to method of selecting which pulse cal tones get propagated to FITS
6. Initial complex sampling support
7. Improved locking mechanism for direct mark5 access (using IPC semaphores; `difxmessage`)

### 2.11.2 Bug fixes

1. Fix model accountability bug in `difx2fits` when combining jobs
2. Numerous fixes for zoom bands (in `mpifxcorr`, `vex2difx` & `difx2fits`)
3. Native Mark5 has improved stability for cranky modules
4. Numerous fixes for DiFX-based phase cal extraction (mostly in `difx2fits`, mostly for multi-job)
5. Fractional bit correction for a portion of lower sideband data got broken in `difx 2.0.0`. Fixed.
6. Migrate `difxcalculator` to DiFX 2; was not complete for DiFX 2.0.0

DiFX 2.0.1 was released on June 24, 2011.

## 2.12 DiFX 2.1

### 2.12.1 New features

1. Mark5-based correlation: easy access to S.M.A.R.T. data (can be viewed with `getsmart`)
2. Mark5-based correlation: emit multicast message containing drive statistics after each scan
3. VSIS interface added to `mk5daemon`
4. Support for non power-of-2 FFT lengths
5. New utilities: `mk5map` (limited functionality), `fileto5c`, `record5c`
6. Remote running of `vex2difx` from `mk5daemon`
7. Multithread VDIF support enabled for the data sources FILE and MODULE, including stripping of non-VDIF packets
8. New features added to existing utilities:
  - `mk5cp`: copy without reference to a module directory
  - `mk5cp`: ability to send data over ssh connection
  - `vsn`: get SMART data from disk drives
9. e-Control source code analysis (Martin Ettl, Wettzell)
10. Restart of correlation is now possible

11. `difx2fits`: -0 option to write minimal number of visibilities to FITS
12. `difx2fits`: write new RAOBS, DECOBS columns in source table
13. `tweakIntTime` option to `vex2difx` has been re-enabled
14. `difDiFX.py` can now cope with two files that don't have exactly the same visibilities (i.e., some visibilities are missing from one file)
15. `plotDiFX.py` and `plotDynamicSpectrum.py` now have better plotting and more options
16. New FAKE datastream type for performance testing
17. Espresso, a lightweight system for managing disk-based correlation, has been added to the DiFX repository.
18. Option to correlate only one polarization has been added.
19. `mk5dir` can now produce `.dir` file information for VDIF formatted data.
20. Add NRAO's sniffer plotters to the repository.

### 2.12.2 Bug fixes

1. LBA format data now scaled roughly correctly (removing the need for large ACCOR corrections).
2. There was a bug when `xmaclength` was  $>$  `nfftchan` for pulsar processing. This has been corrected.
3. `guardns` was incorrectly (overzealously) calculated in `mpifxcorr`.
4. `Mk5DataStream::calculateControlParams: bufferindex>=bufferbytes` bug fixed.
5. Low weight reads could result in uninitialized memory; fixed.
6. Streamstor `XLRRead()` bug work-around installed several places (read at position 0 before reading at position  $>$  0). This is thought not to be needed with Conduant SDK 9.2 but the work-around has no performance impact.
7. Fix to pulse calibration data ordering for LSB or reordered channels.
8. Pulse cal amplitude now divided by pulse cal averaging time in seconds.
9. Pulse cal system would cause crash if no tones in narrow channel. Fixed.
10. Zoom band support across mixed bandwidths (see caveat below).
11. Fix for spurious weights at end of jobs (untested...)
12. Mixed 1 and 2 bit data are handled more cleanly
13. `mpifxcorr` terminates correctly for all short jobs. Previously it hung for jobs with a number of subints between `nCores` and  $4 \times$  `nCores`
14. Correctly scale cross-correlation amplitudes for pulsar binning when using `TSYS > 0` (accounts for varying number of samples per bin c.f. nominal)
15. Lower side-band pulse cal tones had sign error. Fixed.

DiFX 2.1 was released on May 25, 2012.



## 2.13 DiFX 2.1.1

DiFX 2.1.1 was a minor patch release to fix a scaling issue with autocorrelations of LBA-format data in `mpifxcorr`.

DiFX 2.1.1 was committed as a patch to DiFX 2.1 on June 7, 2012.

## 2.14 DiFX 2.2

### 2.14.1 New features

1. `calcif2`: ability to estimate delay polynomial interpolation errors
2. Support for a “label” identifier for a local version of DiFX that will help discriminate exact version used.
3. Faster Mark5 directory reading
4. Faster VDIF corner turning through customized bit shifting functions
5. `mpifxcorr` can now be built without Intel Integrated Performance Primitives, though resulting in a slower correlator.
6. `vdifio`: several new VDIF manipulation and processing utilities added: `vmux`, `vsum`, `vdifd`, `vdifspec`, `vdiffold`, `vdifbstate`
7. `difxbuild`: a new installation program
8. `difxspeed`: a program to benchmark and help optimize DiFX

### 2.14.2 Bug fixes

1. Mutex locking bugfix for very short jobs
2. Prevent MODE errors when a datastream runs out of data well before the end of a job
3. `calcif2`: fix azimuth polynomial generation in case of wrap
4. Fix for FITS file generation for mixed sideband correlation
5. `difx2fits` now uses appropriate gain tables for S and X band in S/X experiments (Thanks to James Miller-Jones for reporting)
6. `difx2fits`: correct `pcal`, `weather`, `tsys` and `flag` data for observations crossing new year
7. Fixed scaling of autocorrelations for LBA format data
8. 0.5 ns wobble in delays for 2 Gbps Mark5B data fixed
9. Fix bug preventing subintegrations longer than 1 second. Now 2 seconds is allowed (this limit comes from signed integer number of nanoseconds).
10. Weights corrected in cases where two setups differencing only by `pcal` setup were correlated against each other
11. Quashed data and weight echos that would occur for about 1 integration at the beginning of each scan for datastreams that ran out of data before end of job.
12. The multicast (diagnostic) weights were low or zero in case of frequency selection (zoom band or `freqId` selection). Fixed.
13. `mark5access`: fix (non)blocking issue when receiving data from `stdin`

DiFX 2.2 was released on June 12, 2013.

## 2.15 DiFX 2.3

### 2.15.1 New features

1. mpifxcorr: LO offsets are now corrected in the time domain when fringe rotation is also done in the time domain (the usual mode), allowing considerably larger LO offsets without decorrelation
2. mpifxcorr: Working polarization dependent delay and phase offsets
3. mpifxcorr: Experimental linear2circular conversion
4. mpifxcorr: Complex Double sideband (RDBE/Xcube) sampling support (Note: things are not perfect here; wait for 2.4 for real use)
5. mpifxcorr: new file/Mark5 based VDIF/Mark5b datastream (faster and more robust)
6. mpifxcorr: implement work-around for buggy kernel-driver combinations; Mark5 read sizes  $\geq 20$  MB now allowed
7. utilities: some new command line tools for Mark5B and VDIF files (vsum, mk5bsum, vmux, mk5bfix)
8. new options for passing calibration (Walter B: memo forthcoming)
9. Hops updated to version 3.9

### 2.15.2 Bug fixes

1. mpifxcorr: Datastream buffer send size now calculated correctly for complex sampled data
2. mpifxcorr: Avoid very rare bug where combination of geometric delay and data commencing mid-subint meant one invalid FFT might be computed
3. mpifxcorr: multicast weights are now computed correctly for mixed-sideband correlation
4. mpifxcorr: fixed bug where some autocorrelations were not saved in a mixed-sideband correlation
5. mpifxcorr: fixed bug where send size could be computed incorrectly by 1-2 bytes for
6. Mark4/VLBA/Mark5B/VDIF formats, potentially resulting in very small amounts of data loss

DiFX 2.3 was released on January 18, 2014.

## 2.16 Features left to implement

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

1. Full VDIF support
2. Support for K5 format
3. Pulsar bins with proper output format
4. Fast-forwarding over unneeded data in the native Mark5 module datastream
5. Space VLBI support
6. Playback of two independent modules in one Mark5 unit
7. Non-bank-mode Mark5 support
8. Support for antenna local oscillator offsets

## 2.17 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 114 [4], which will be generally available shortly. It has been modified for better support of DiFX FITS output. In general, these changes make FITLD less telescope specific so the resulting FITS-IDI files from any 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 [5] 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.

With the following exceptions, data reduction of DiFX correlated data should be identical to that of VLBA hardware correlator data. This includes the continued use of DIGICOR=1 in FITLD and the use of ACCOR as you would have for the hardware correlator. The exceptions are:

1. Use of FXPOL to correct data ordering in the case of *half* polar (e.g., RR and LL products) is no longer needed.
2. Use of VBGLU to concatenate data sets in the case of 512 Mbps observations is no longer needed.
3. Data is usually combined into a single FITS-IDI file with proper calibration data attached, usually implying that TBMRG is not needed to properly concatenate calibration data. This makes DiFX FITS-IDI data similar to the *pipeline-processed* VLBA data that was made available to users of the hardware correlator with the difference being that the original FITS-IDI format is retained, keeping file sizes typically 25% smaller.

These changes should make data processing easier in almost all circumstances.

## 3 Hardware and operating system configuration

The sections below may be useful in setting up your cluster and operating system (including the user account that will be running DiFX).

### 3.1 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.

## 3.2 Environment variables

In addition to environment variables needed at build-time (§12), some others are needed at run time. These are:

1. `CALC_SERVER` contains the name of the computer running `calcServer` (§12.2.10). This is accessed only by program `calcif`.
2. `DIFX_ARCHIVE_ROOT` points to the base directory of the archive staging area.
3. `DIFX_ARCHIVE_USERNAME` specifies the username of the archiving process (NRAO use only).
4. `DIFX_GROUP_ID` the Unix group to use. If set, `umask` is changed to 002 and all new files/directories become group writable.
5. `DIFX_HEAD_NODE` contains the name of the cluster head node.
6. `DIFX_MACHINES` points to a file containing a list of cluster members and their capabilities. See §8.12
7. `DIFX_MESSAGE_GROUP` (optional) specifies, with `DIFX_MESSAGE_PORT`, the multicast group and port to be used for GUI and monitoring.
8. `DIFX_MESSAGE_PORT` (optional, *see above*)
9. `DIFX_QUEUE_BASE` points to the base of the correlator staging area for jobs to be run.
10. `DIFX_VERSION` (optional, but recommended) the version of `difx` being used, e.g., `DIFX-2.2`.
11. `EVLAMPTS_DB` contains the connection information for the Postgres E VLA monitor point database (NRAO only, needed only if VLA is included in array).
12. `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.
13. `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.
14. `DIFX_LOG_DIR` points to the directory where logs shall be written.
15. `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.
16. `PGPLOT_FONT` points to a font file for PGPLOT. The font file is usually called `grfont.dat`.

17. `TCAL_PATH` points to a directory containing  $T_{\text{cal}}$  values for receivers.
18. `TESTS` points to a path containing test data projects.
19. `VLBA_DB` contains the connection information for the Oracle legacy VLBA database (NRAO only).
20. `VLBAMPTS_DB` contains the connection information for the Postgress EVLA-style VLBA monitor point database (NRAO only).

Like the environment variables described in §12, 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 §3.3). 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.

### 3.3 Directory structure and versioning

The directory structure of the NRAO deployment of 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., 1.5) 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 DiFX version has its own root directory, such as `/home/swc/DiFX-1.5`. All files associated with this version are under this directory. No data or files associated with any other DiFX version shall be placed within.

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

```
. /home/swc/DiFX-1.5/setup_difx
```

This script contains, among other lines, the following:

```
export DIFX_PREFIX=/home/swc/NRAO-DiFX-trunk
export DIFX_BASE=/home/swc/difx
export DIFX_ARCHIVE_ROOT=/home/ngas_staging/difx
export JOB_ROOT=${DIFX_BASE}/projects
export TESTS=${DIFX_BASE}/tests
export MARK5_DIR_PATH=${DIFX_BASE}/directories
export CALC_SERVER=swc000
export GAIN_CURVE_PATH=${DIFX_BASE}/gaincurves
export DIFX_MACHINES=${DIFX_BASE}/machines.difx
export DIFX_QUEUE_BASE=${DIFX_BASE}/queue
export DIFX_HEAD_NODE=swc000
export DIFX_VERSION=DIFX-1.5
export DIFX_GROUP_ID=vlba_difx
export DIFX_MESSAGE_GROUP=224.2.2.1
export DIFX_MESSAGE_PORT=50200
export IPPROOT=/home/swc/difx/intel/ipp/6.0.2.076/ia32
export PATH=${DIFX_PREFIX}/bin:${ORACLE_HOME}/bin:/users/difx/bin:/bin:/usr/bin
export LD_LIBRARY_PATH=${DIFX_PREFIX}/lib:${IPPROOT}/sharedlib:${ORACLE_HOME}/lib
echo "DIFX version 1.5 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 DiFX components as well as using them. Each installed version of DiFX will have its own setup file like this. Selecting which version is to be used is as simple as running the correct setup file. To change to the development version:

```
. /home/swc/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 DiFX this is set differently. It may also be desirable to customize this for your correlator. For example, one may set it to `USNO-DIFX-1.5`. 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.

## 4 Running DiFX at the VLBA correlator (VLBA Specific)

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.

Note that this section will see considerable enhancements as experience in running the correlator is gained.

### 4.1 Software correlation based on vex files

With VLBA DiFX version 1.5 comes correlation based on the `.vex` files rather than the hardware correlator jobs scripts. This new path frees operations from a host of difficult to maintain software, including `cjobgen` and its associated software. The vex-based correlation was first documented a memo titled “VLBA-DIFX Operations Plan” [1]. Step-by-step instructions describing the process is repeated here. The particular case being exemplified here is based on the complicated pulsar astrometry project. Most real-life examples will be simpler, but some may be more complex. Note that these instructions represent the expected way to proceed, but changes to the software architecture may introduces changes to some of these steps.

It should be kept in mind that all actions performed by the analysts will be *pass based* which means one or more jobs at a time. Rarely will analysts have to worry about individual jobs or FITS files. The correlator operators on the other hand work entirely on the job basis. Commands to be issued by the analysts are preceded by an arrow ( `→` ). In general, all files written in the processes that follow are readable and writable by everyone in the `vlba_difx` group. The exception is data sent to the archive staging area, which is readable and writable only by `e2emgr`.

1. First change to the project directory. Assume that the project is called `BX123` and that it was observed in December 2009.  
`→ cd /home/vlbiobs/astronomy/dec09/bx123`
2. Extract from the monitor database the Mark5 module logs, clock offsets and rates, and EOPs making a new vex file called `bx123.skd.obs` and a file called `bx123.skd.shelf`. The original (schedule) vex file that was used during observation is never to be modified. In extreme cases, the new vex file being created in this step `bx123.skd.obs` can be hand edited to reflect what actually happened during observation, but doing this should be extremely rare. This step locks in the EOP values that will be used for each job made for this project.  
`→ db2vex bx123.skd`
3. Next form the template input file for `vex2difx` from the `.oms` file written by `sched`. This creates `bx123.v2d`.  
`→ oms2v2d bx123.oms`
4. For simple experiments it is likely that the `.v2d` file created in the previous step can be used unmodified. For this complicated experiment changes will need to be made. Since this project requires four correlator *passes*, this `.v2d` file will need to be copied four times and each one edited to reflect the

purpose of the correlator pass. Sophisticated VLBA users may provide their own set of `.v2d` files that might need light editing before use.

```
→ cp bx123.v2d clock.v2d
```

```
→ emacs clock.v2d
```

5. VLBA-DiFX `.input` files are generated at this point using `vex2difx`. By design, `vex2difx` has no options associated with it – it is entirely configured through the `.v2d` files. In the case below, the files `clock_1.input`, `clock_1.calc`, and `clock_1.flag` will be created. This command will also make a file called `clock.joblist` that lists each job created for this correlator pass with a summary of the job properties, such as start and stop times and number of stations.

```
→ vex2difx clock.v2d
```

6. If the correlator jobs created above are deemed ready to run, they are sent to the correlator queue. In this process three things will occur: 1. `CALC` will be run to generate the correlator delay models needed for correlation, 2. the `.input` files generated by `vex2difx` will be copied to the software correlator run directory, and 3. the VLBA database will be told that the jobs are ready. At this time, a priority can be set to the jobs being sent to the correlator, making them appear at the top of the queue. Otherwise the jobs in the queue will appear in observe time order. In the example below, the option `-p 1` indicates that this job should run with elevated priority. Supplying `clock` with no prefix implies queuing all the jobs in the clock pass. Individual jobs could be queued by specifying a list of `.input` files.

```
→ difxqueue -p 1 add clock
```

7. When the jobs are complete, which can be determined with `difxqueue` using the `list` action, the correlator output is converted to FITS format. Data “sniffing” happens automatically during this step. The command to do this will ensure that all of the jobs in the pass have been successfully correlated. Note that the number of FITS files created is not necessarily the same as the number of correlator jobs. A file called `clock.fitslist` will be generated in this step that lists all of the fits files that are part of this correlator pass including for each FITS file a list of the jobs that contributed to that FITS file. The program `makefits` will use program `difx2fits` to do the actual conversion.

```
→ makefits clock.joblist
```

8. The sniffer output files are at this point inspected. Program `difxsniff` is run to produce plots which are identical to those produced by sniffer today. Multiple reference antennas (in this example, Los Alamos and Kitt Peak) can be provided at the same time. Sniffer plots and the data that is used to generate them will be placed in a sub-directory of the project directory called `sniffer/clock` for a pass called “clock”.

```
→ difxsniff LA KP clock.fitslist
```

```
→ gv sniffer/clock/apdfile.ps
```

9. If the FITS files are deemed acceptable, they are entered into the VLBA data archive.

```
→ difxarch clock.fitslist
```

10. Finally, once data has been archived, the intermediate files should be removed from the head node:

```
→ difxclean bx123
```

Note that this final step is needed only after the entire project is ready for releasing and should not be done every time between completion of job passes.

#### 4.1.1 Directory contains undecoded scans

Occasionally a message of the form `Module module directory contains undecoded scans!` will appear. This means that one or more scans for the module named *module* was not properly read or decoded. This should first be verified by examining the module directory file, called `$MARK5_DIR_PATH/module.dir`. This examination is best done with program `checkdir` which looks for a number of possible abnormalities. Rows where the eleventh column contains a negative number are scans that were not decoded properly. A known problem occasionally causes many consecutive scans at the end to be improperly read and thus undecoded. If this is the case, rename the existing directory file and try reacquiring the directory. Usually it will start working immediately.

If one or a small number scans repeatedly cannot be decoded, the scan may be corrupted for some reason. In this case, simply delete the row(s) from the directory file and then decrement the number following the module name on the first line of the file by the number of scans deleted; this count of the number of scans listed in the file must remain accurate. This operation will cause the correlator to skip over these affected scans and data will be lost, so use appropriate judgement in these cases.

#### 4.1.2 Directory read fails on partial module

Modules containing less than 8 working disks can be problematic. It is suggested that modules of this type have their directories read preemptively using a special command:

→ `mk5control safedirA 12`

which is the command to *safely* read the directory of the module in bank A of `mark5fx12`.

#### 4.1.3 Mark5 unit hangs while reading directory

Typically the first thing one should do if a hang occurs is to try again. For directory reading this can be attempted with the `mk5control` program. For instance, if the module in bank A of `mark5fx12` hangs during the directory read, stop the correlation process with the DOI or via `stopmpifxcorr` and then issue:

→ `mk5control getdirA 12`

If this also fails, or never starts, reboot the unit via the DOI or

→ `mk5control reboot 12`

or

→ `ssh 12 /sbin/reboot` if it really refuses to reboot.

Once the unit comes back, try retrieving the directory again.

#### 4.1.4 Mark5 directory reading fails partway through

When the GUI button GetDir fails, the program `mk5dir` can be used directly to read a module directory.

Things to try first:

1. Log into the fx unit and run `vsN` to look for obvious module problems
2. Move the module
3. Erase (or move/rename) the preexisting directory file
4. Reboot the correlator Mark5 unit

When GetDir fails or crashes the Mark5 unit, it is likely because there are one or more spots on the module that can't be read. Using `mk5dir`, you can read most of the directory while skipping any problematic scans.

The `mk5dir` program will work on both Mark5A and Mark5C modules. It is best to put them, respectively, in SDK 8 and 9 units. As with most utilities, typing `mk5dir` by itself will print help information.

The output directory will be named the usual `vsN.dir` and will overwrite any existing file of that name. It will be written to `$MARK5_DIR_PATH`, which is the same place to which GetDir writes directories.

The relevant options in this case are:



- `-f` (force a directory read even if a file already exists)
- `-v` (be verbose)
- `-e scan number` (stop reading the directory at a certain scan number)
- `-b scan number` (begin reading the directory at a certain scan number)

The scan numbering is worth noting. The command line options `-e` and `-b` number the scans starting at 1 (the first scan is 1). But the on-screen output of `mk5dir` will begin with a scan numbered 0.

The first step is to read as much of the directory before the first problem scan.

1. Log in to the fx unit
2. Run `vsn bank` to get an overview of the health of the module
3. Run `mk5dir -f -v bank`

As it reads each scan, it will print a line indicating its progress.

```
0/228 -> 3 Decoded
1/228 -> 3 Decoded
etc ...
```

The first number is the scan it just decoded and the second number is the total number of scans on the module. The “3” is related to the data format and should be 3 for Mark5C at all VLBA sites. VLA modules will show “4”. Legacy VLBA and foreign stations may show other numbers. The “Decoded” indicates success, as opposed to something like “XLR Read Error”.

Presumably, it will fail at some point. When it fails it probably won’t write an output directory file. Note which scan it failed to read. Remember that scan  $x/228$  is actually the  $x+1^{th}$  scan because it started counting at zero. You may have to reboot the fx unit at this point.

Now run `mk5dir` again, this time stopping one scan previous to the one where it died last time.

1. Run `vsn bank` (whether you had to reboot or not, checking the module with `vsn` is probably a good idea)
2. Run `mk5dir -f -v -e x bank` (`mk5dir` will stop once it has read  $x$  scans)
3. Rename the output file so it doesn’t get overwritten.

Now we can skip past the bad parts and read the rest of the directory.

1. Run `mk5dir -f -v -b x+2 bank` (`mk5dir` will start with the scan after the one it originally failed on)

If it fails, reboot as necessary, run `vsn` again, and try starting with scan  $x+3$  instead. Keep incrementing the start scan until it works; sometimes it might be faster to try a bigger jump, and on success work backwards to find where failing starts. Eventually, you will have skipped past the bad scans and read the rest of the directory. There are likely to only be one or two bad scans, so this step should actually be fairly simple.

Now you will have two directory files, the renamed file with the first  $x$  scans, and the second file with all the scans after the bad scan(s). Each file will also have entries for the scans that weren’t read by that particular invocation of `mk5dir`, and these lines need to be deleted. Then using your preference of linux commands and/or text editor, combine the two files in the appropriate order. Make sure there is a single header line at the top which lists the correct number of scans.

If there are nonsequential bad scans you will have to concatenate three or more files, but the steps remain the same.

#### 4.1.5 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 watching the output of `cpumon` and `mk5mon` (or the equivalent from the DOI). If no updates come from `cpumon` then it is likely that the computer has seized and requires a hard reboot. Otherwise if `mk5mon` shows no updates, the problem is likely with the Streamstor card and/or disk module. If logging into the Mark5 unit works, try resetting the Streamstor 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` (§7.49) in a terminal and issue the following command:

```
mk5control getvsn mark5
```

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

#### 4.1.6 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 the Difx Operator Interface or `startdifx` (§7.59) is used to run the correlator.

## 4.2 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 future versions of DiFX. 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 §8.25 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.

## 4.3 The VLBA database

Many of the existing VLBA tools (such as the Observation Management System (OMS), `mon2db`, `cjobgen`, and others) make use of an Oracle database for persistent storage of various information related to projects that use either the VLBA antennas or correlator. Many aspects of VLBA-DiFX are not a good match for the existing database tables; adapting the existing tables to work nicely with VLBA-DiFX will be disruptive and have implications for much existing code, including software that will not be needed once FXCORR is

shut down. The proposed solution to this dilemma is to use a parallel set of database tables for correlation and archiving when using VLBA-DiFX. The use of existing software for generation of FXCORR jobs will continue unchanged. For projects to be correlated using VLBA-DiFX, OMS will still be used for observation preparation tasks, but will not be used in preparation of correlation or anything that occurs beyond that in the project’s life cycle. Instead, `vex2difx` will be used to generate jobs, `difxqueue` will be used in lieu of OMS to stage correlator jobs, and `difxarch` will be used in the archiving of data. The queuing tool `difxqueue` will be used to display the state of the VLBA-DiFX job queue as well as populate it. The new tools will access three new database tables: DIFXQUEUE and DIFXLOG; the contents of these tables is shown in Tables 1 & 2.

Column	Type	Comments
PROPOSAL	VARCHAR2(10)	The proposal code
SEGMENT	VARCHAR2(2)	Segment (epoch) of proposal, or blank
JOB_PASS	VARCHAR2(32)	Name of correlator pass (e.g. “geodesy”)
JOB_NUMBER	INT	Number of job in the pass
PRIORITY	INT	Number indicating the priority of the job in the queue 1 is highest
JOB_START	DATE	Observe time of job start
JOB_STOP	DATE	Observe time of job stop
SPEEDUP	FLOAT	Estimated speed-up factor for job
INPUT_FILE	VARCHAR2(512)	Full path of the VLBA-DiFX input file
STATUS	VARCHAR2(32)	Status of the job, perhaps “QUEUED”, “KILLED”, “RUNNING”, “FAILED”, “UNKNOWN” or “COMPLETE”
NUM_ANT	INT	Number of antennas in the job

Table 1: The DIFXQUEUE database table. This table is based on the FXQUEUE table currently used by OMS. Entries to this table will be initially made by `difxqueue`. The STATUS field will be automatically updated as appropriate during correlation.

Column	Type	Comments
PROPOSAL	VARCHAR2(10)	The proposal code
SEGMENT	VARCHAR2(2)	Segment (epoch) of proposal, or blank
JOB_PASS	VARCHAR2(32)	Name of correlator pass (e.g. “geodesy”)
JOB_NUMBER	INT	Number of job in the pass
CORR_START	DATE	Start time/date of correlation
CORR_STOP	DATE	Stop time/date of correlation
SPEEDUP	FLOAT	Measured speed-up factor
INPUT_FILE	VARCHAR2(512)	File name of .input file
OUTPUT_FILE	VARCHAR2(512)	File name of correlator output
OUTPUT_SIZE	INT	Size (in $10^6$ bytes) of correlator output
CORR_STATUS	VARCHAR2(32)	Status of correlation, typically “COMPLETED”

Table 2: The DIFXLOG database table. This table is based on the FXLOG table currently used by OMS. A row will be written to this table after each successful correlation by the DiFX Operator Interface.

## 4.4 Archiving

Archiving of VLBA-DiFX data will be done on a per-pass basis. All `.FITS` files associated with a single correlator pass will be archived together. A particular staging directory for VLBA-DiFX data has been set up. Populating the archive amounts to first copying the files to be archived to this directory making sure that the first character of the file name is “.”. Once the entire file is transferred this file is renamed without the leading period. This system is the standard way to populate the Next Generation Archive System (NGAS) [8] without potential for an incompletely copied file to be archived. The file names will be composed only of alpha-numeric characters and “.” and “\_”. These characters have no special meaning in any relevant software, including http, XML, bash/Linux command lines, the Oracle database parser, etc. File names will have the following format:

*VLBA\_projectCode\_passName\_fileNum\_corrDateTcorrTime.idifits*

where the italicized fields, which themselves will be limited to alphanumeric characters, are as follows:

Field	Type	Comment
<i>projectCode</i>	string	Project code, including segment if appropriate
<i>passName</i>	string	Name of the pass, as set in the <code>.v2d</code> file
<i>fileNum</i>	integer	FITS file sequence number within pass
<i>corrDate</i>	date ( <i>yymmdd</i> )	Date corresponding to correlation completion
<i>corrTime</i>	time ( <i>hhmmss</i> )	Time corresponding to correlation completion

Parameter *fileNum* is the sequence number of the created `.FITS` file which may or may not have a direct correspondence with the job sequence number within the correlator pass. An example archive file name relevant to the sample project used in this memo may be:

*VLBA\_BX123\_clock.1\_091223T032133.idifits*

All files produced for a given pass will be placed in a single directory,

`$DIFX_ARCHIVE_ROOT/projectCode/passName`

where `DIFX_ARCHIVE_ROOT/` is an environment variable pointing to the head of the archive staging area for VLBA-DiFX. During the transfer to the archive, the *projectCode* portion of the directory tree will begin with a period that is to be renamed once all files are completely copied. This will allow the archive loader to logically group together all the files of the pass. If needed, an index file listing the association of archive `.FITS` files and correlator jobs can also be placed in this directory. In order to ensure the atomic nature of correlator passes in the archive, the renaming of the copied files from the temporary versions starting with “.” will not occur until all archive files are transferred. The `.fitslist` file produced by `difx2fits` would serve this purpose. An archive loader will periodically (initially about every 30 minutes, but perhaps later with much shorter intervals) look for new files in the archive staging area to store. The archive data will be available moments later for users wanting to download the data.

## 4.5 Ownership and permissions of files

This section describes how the different user accounts interact within the DiFX correlation process. This is VLBA operations specific. No fewer than 4 user accounts are used in the life cycle of a DiFX project:

- *analysts*: The *analysts* account is used to prepare the `.input` and other files used to run DiFX. These files retain *analysts* ownership in the `/home/vlbiobs/astrometry` area and while queued.
- *root*: The *root* account owns the `mk5daemon` processes that run on each of the Mark5 and software correlator computers.
- *difx*: User account *difx* is used to actually run `mpifxcorr` (and `difxlog`) during correlation. This is important, as the *difx* account has all of the proper environment variables set up. The DiFX Operator Interface may also be run as user *difx*.

- *e2emgr*: The data archive requires that the files staged for entry into the archive be given *e2emgr* ownership. This is accomplished with the program `e2ecopy` which runs with root privileges and hence can change its userid to *e2emgr* as needed.

By default, all files will have owner and group read/write permission and global read permission.

## 4.6 Copying baseband data

Some users wish to perform their own analysis on baseband VLBI data. This section describes the procedure for copying data and ends with guidelines that should be sent to each user requesting data to be copied. The guidelines are there to streamline the process and to minimize the change of problems.

### 4.6.1 Performing the data copy

*FIXME : write me!*

### 4.6.2 Guidelines to users

Users wishing to retain a copy of the baseband data should make sure to conform to the following guidelines. Please note that many of the instructions below are there to ensure that root access is not required in the copying of the data. If root access is needed as a result of failure to comply, a delay in the copying will be incurred.

1. Arrangement for data copy should be made prior to observing.
2. The external disk must have a working USB2 connector for data transfer. We specifically do not support Firewire at this time. We do not support power-over-USB drives.
3. Each disk should have a sticker label attached with the owner's name, institution, phone number, shipping address and email. The project code should be clearly marked as well. If multiple disks are shipped, each should have a unique serial number clearly labeled on the disk.
4. The external disk must be preformatted with a standard Linux filesystem (ext2 or ext3). It is preferred that the options `-m 0 -T largefile4` are used with `mke2fs` and it is also suggested that the `-L` option is used to specify a volume name/number that matches the labelled serial number.
5. It is the user's responsibility to ensure that sufficient post-formatted capacity is available on the disks.
6. A world-writable root-level directory with the name of each project to be copied should be made. The directory name should contain only uppercase letters and numbers.
7. The disk should be empty upon delivery. NRAO will not be responsible for data that is deleted. The root-level directory(s) mentioned above should be the only exception(s).
8. Include the power transformer/cable and USB cable with the disk. It is recommended that owner labels be attached to each of these.
9. Note to foreign users: please ensure that the power supply has either a NEMA 1-15 / Type A ungrounded power connector or a NEMA 5-15 / Type B grounded power connector and that the power supply works on 110V/60Hz. If this is accomplished using an adapter, it must be included in the box with the disk.
10. Foreign users will be responsible for customs charges and are encouraged to contact NRAO well in advance of any shipment to minimize cost.

## 4.7 Some comments on channels

This section discusses the accountability of channel identification through the entire 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.

## 5 DiFX and pulsars

DiFX supports three pulsar processing modes:

- **Binary Gating** A simple on-off pulse accumulation window can be specified with an “on” phase and an “off” phase. This can be used to boost the signal to noise ratio of pulsar observations by a factor of typically 3 to 6 and can also be used to search for off-pulse emission.
- **Matched-filter Gating** If the pulse profile at the observation frequency is well understood and the pulse phase is very well predicted by the provided pulse ephemeris, additional signal to noise over binary gating can be attained by appropriately scaling correlation coefficients as a function of pulse phase. Depending on the pulse shape, addition gains by a factor of up to 1.5 in sensitivity over binary gating are realizable.
- **Pulsar Binning** Pulsar *binning* is supported within DiFX. This entails generating a separate visibility spectrum for each requested range of pulse phase. There are no explicit limits to the number of pulse phase bins that are supported, however, data rates can become increasingly large. Currently AIPS does not support databases with multiple phase bins. Until there is proper post-processing support for pulsar binning, a separate FITS file will be produced for each pulsar phase bin.

In all cases the observer will be responsible for providing a pulsar spin ephemeris. Except for certain applications of mode 5 the ephemeris must be capable of predicting the absolute rotation phase of the pulsar. Enabling pulsar modes incurs a minimum correlation-time penalty of about 50%. High output data rates (computed from time resolution, number of spectral channels, and number of pulsar bins) may require greater correlator resource allocations. The details of pulsar observing, including practical details of using the pulsar modes and limitations imposed by operations, are documented at *FIXME*.

### 5.1 Pulse ephemeris

The use of any pulsar mode requires a pulse ephemeris to be provided by the astronomer. This is a table of one or more polynomial entries, each of which evaluates the pulsar’s rotation phase over an interval of

typically a few hours. The classic pulsar program `Tempo` can be used to produce the polynomials required [6]. The pulse phase must be evaluated at the Earth center which is usually specified in `tempo` by station code 0 (zero). Many pulsars exhibit a great degree of timing noise and hence the prediction of absolute pulse phase may require updated timing observations. When submitting the polynomial for use at the correlator, please adhere to the following naming convention: *experiment-pulsar.polyco* , e.g., `BB118A-B0950+08.polyco` . Instructions for generating the polynomial file are beyond the scope of this document.

## 5.2 Bin configuration file

All three pulsar modes also require the preparation of a `.binconfig` file by the astronomer. The contents of this file determine which of the three pulsar modes is being used. Three pieces of information are contained within this file: the pulsar ephemeris (`polyco`) files to apply, definitions of the pulsar bins, and a boolean flag that determines whether the bins are weighted and added within the correlator. The file consists of a set of keywords (including a colon at the end) that must be space padded to fill the first 20 columns of the file and the values to assign to these keywords that start at column 21. The file is case sensitive. The pulsar bins each consist of a ending phase and a weight; each bin is implicitly assumed to start when the previous ends and the first bin starts at the end phase of the last. The phases are represented by a value between 0 and 1 and each successive bin must have a larger ending phase than the previous. Examples for each of the three pulsar modes are shown below:

### 5.2.1 Binary gating

```
NUM POLYCO FILES: 1
POLYCO FILE 0: BB118A-B0950+08.polyco
NUM PULSAR BINS: 2
SCRUNCH OUTPUT: TRUE
BIN PHASE END 0: 0.030000
BIN WEIGHT 0: 1.0
BIN PHASE END 1: 0.990000
BIN WEIGHT 1: 0.0
```

### 5.2.2 Matched-filter gating

```
NUM POLYCO FILES: 1
POLYCO FILE 0: BB118A-B0950+08.polyco
NUM PULSAR BINS: 6
SCRUNCH OUTPUT: TRUE
BIN PHASE END 0: 0.010000
BIN WEIGHT 0: 1.0
BIN PHASE END 1: 0.030000
BIN WEIGHT 1: 0.62
BIN PHASE END 2: 0.050000
BIN WEIGHT 2: 0.21
BIN PHASE END 3: 0.950000
BIN WEIGHT 3: 0.0
BIN PHASE END 4: 0.970000
BIN WEIGHT 4: 0.12
BIN PHASE END 5: 0.990000
BIN WEIGHT 5: 0.34
```

Note here that there is zero weight given to pulse phases ranging between 0.05 and 0.95.

### 5.2.3 Pulsar binning

```
NUM POLYCO FILES: 1
POLYCO FILE 0: BB118A-B0950+08.polyco
NUM PULSAR BINS: 20
SCRUNCH OUTPUT: FALSE
BIN PHASE END 0: 0.025000
BIN WEIGHT 0: 1.0
BIN PHASE END 1: 0.075000
BIN WEIGHT 1: 1.0
BIN PHASE END 2: 0.125000
BIN WEIGHT 2: 1.0
BIN PHASE END 3: 0.175000
BIN WEIGHT 3: 1.0
.
.
.
BIN PHASE END 18: 0.925000
BIN WEIGHT 18: 1.0
BIN PHASE END 19: 0.975000
BIN WEIGHT 19: 1.0
```

The primary difference is `SCRUNCH OUTPUT: FALSE` which causes each pulsar bin to be written to disk.

### 5.3 Preparing correlator jobs

When using `vex2difx` to prepare correlator jobs, one must associate the pulsar with a setup of its own that includes reference to the `.binconfig` file. An excerpt from a `.v2d` file is below:

```
SETUP gateB0950+08
{
    tInt = 2.000
    nChan = 32
    doPolar = True
    binConfig = BB118A-B0950+08.binconfig
}

RULE B0950+08
{
    source = B0950+08
    setup = gateB0950+08
}
```

The `.binconfig` file should be in the same path as the `.v2d` file when running `vex2difx`.

### 5.4 Making FITS files

For the two gating modes, preparing FITS files with `difx2fits` is no different than for any other DiFX output. FITS-IDI does not support multiple phase bins so the pulsar binning case is different and the situation is non-optimal. Each pulsar bin must be made into its own FITS file with a separate execution of `difx2fits`. The `-B` (or `--bin`) command line option takes the bin number (starting at zero as above) and writes a FITS file containing data only associated with that bin number. Be sure to systematically name output files such that the bin number is understood.



## 6 Conventions

### 6.1 Clock offsets and rates

The clock offset (and its first derivative with respect to time, the clock rate) are stored in a number of places through the correlator toolchain. The convention used by vex is that the clock offset is positive if the Data Acquisition System (DAS) time tick is early (i.e., the station clock is running fast) and accordingly the full name in the vex file is *clock\_early*. In all other parts of the system the opposite sign convention is used, that is the clock offset is how late (slow) the DAS clock is. In particular, all VLBA correlator job files, the VLBA database, the DiFX `.input` file and FITS formatted output files all use the “late” clock convention. Note in particular that the *clockOffset* and *clockRate* parameters of the ANTENNA sections in the `.v2d` files use the “late” convention, not vex’s “early” convention.

### 6.2 Geometric delays and rates

The delays used to align datastreams before correlation are nominally produced by the Goddard CALC package. The calculations are done on an antenna basis using the Earth center for antenna A and the requested station for antenna B and thus results in a negative delay for sources above the horizon. The core of DiFX uses the opposite convention and thus the delays and their time derivatives (rates) as stored in the `.delay`, `.rate` and `.im` files use the “positive delay above horizon” convention. The FITS-IDI files (as produced by `difx2fits` and the VLBA hardware correlator) use the same convention as CALC, that is “negative delay above horizon”.

### 6.3 Antenna coordinates

Geocentric ( $X, Y, Z$ ) coordinates are used universally within the software correlator and its associated files. This usage pattern extends to cover `sched`, CALC, the VLBA database and the existing hardware correlator as well. The values are everywhere reported in meters. The  $Z$ -axis points from the Earth center to the geographic North pole. The  $X$ -axis points from the Earth center to the intersection of the Greenwich meridian and the equator (geographic longitude  $0^\circ$ , latitude  $0^\circ$ ). The  $Y$ -axis is orthogonal to both, forming a right handed coordinate system; The  $Y$ -axis thus points from the Earth center to geographic longitude  $90^\circ\text{E}$ , latitude  $0^\circ$ . The unit-length basis vectors for this coordinate system are called  $\hat{x}$ ,  $\hat{y}$ , and  $\hat{z}$ . The only exception to the above stated rules is within the AIPS software package. Task FITLD flips the sign of the  $Y$  coordinate, apparently to maintain consistency with software behavior established in the early days of VLBI.

### 6.4 Baseline coordinates

The baseline vectors are traditionally put in a coordinate system that is fixed to the celestial sphere rather than the Earth; see the discussion in §7.1 for a discussion of coordinates that is mathematically more precise. The unit-length basis vectors for this coordinate system are called  $\hat{u}$ ,  $\hat{v}$ , and  $\hat{w}$ . The axes are defined so that  $\hat{w}$  points in the direction of the observed source tangent point, the  $\hat{u}$  is orthogonal to both the vector pointing to the celestial north pole  $\hat{N}$  ( $\delta_{J2000} = +90^\circ$ ) and  $\hat{w}$ , and  $\hat{v}$  orthogonal to  $\hat{u}$  and  $\hat{w}$ . Note that the sign of the  $\hat{v}$  is such that  $\hat{v} \cdot \hat{N} > 0$  and  $\{\hat{u}, \hat{v}, \hat{w}\}$  form a right-handed coordinate system. A baseline vector  $\vec{B}_{ij}$  is defined for an ordered pair of antennas, indexed by  $i$  and  $j$  at geocentric coordinates  $\vec{x}_i$  and  $\vec{x}_j$ . For convenience, the Earth center will be denoted by  $\vec{x}_0$  which has coordinate value  $\vec{0}$ . The  $(u, v, w)$  coordinates for baseline vector  $\vec{B}_{ij}$  is given by  $(\vec{B}_{ij} \cdot \hat{u}, \vec{B}_{ij} \cdot \hat{v}, \vec{B}_{ij} \cdot \hat{w})$ . There are two natural conventions for defining baseline vectors:  $\vec{B}_{ij} = \vec{x}_i - \vec{x}_j$  and  $\vec{B}_{ij} = -\vec{x}_i + \vec{x}_j$ , which will be referred to as first-plus and second-plus respectively; both conventions are used within the correlator system. Antenna-based baseline vectors are stored in polynomial form in the `.im` file and tabulated in the `.uvw` file. In all cases antenna-based baseline vectors are baseline vectors defined as  $\vec{B}_i \equiv \vec{B}_{0i} = \pm \vec{x}_0 \mp \vec{x}_i$ . The values stored in the `.im` and `.uvw` files adopt the first-plus convention. For antennas on the Earth surface this implies that the  $w$  baseline component is always negative

for antennas that see the target source above the horizon. The `difx` format output from `mpifxcorr` contains a baseline vector for each visibility spectrum computed from the locations of the antenna pair using the first-plus convention. Note that in this file output, the reported baseline number is  $256 * i + j$  and  $i$  and  $j$  are antenna indices starting at 1. The FITS-IDI format written by `difx2fits` adheres to the second-plus convention.

## 6.5 Visibility phase

# 7 Reference guide to programs and utilities

This section has usage information for the numerous programs and scripts used in the 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, depending on the program. 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 `arg1 [ ... argN]`, 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. The package that includes each program is included in its section heading.

Note that several VLBA specific programs are discussed in this manual that are not documented here, such as `tsm`. These are preexisting programs that may be documented elsewhere and are less likely to be useful outside VLBA operations. Also note that programs from the `difx_db` package are internal to NRAO and in general are not applicable outside VLBA operations. The code for these programs can be made available upon request.

## 7.1 `calcif2`

Program `calcif2` evaluates the delay model, producing a delay model file (ending with `.im`) from a file containing the source, antenna and scan timing information (ending with `.calc`). The detailed calculations are performed by the Goddard CALC program. Prior to this writing (May 4, 2013), the only option for the calculation back-end was CALC version 9 with NRAO additions which add ocean loading and near-field corrections (accurate as close as a few  $\times 10^5$  km). Now new options are being introduced, including CALC 9 with the Sekido-Fukushima near-field model (using the `--sekido` option). Note that this option requires an installation of a special version of CALC that is not covered in this document.

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:

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

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

Normally `calcif2` will be called by `difxqueue`, `startdifx`, or another higher-level program if needed.

`calcif2` connects via Remote Procedure Call (RPC) to an instance of `CalcServer` which must be running on a computer identified by environment variable `$CALC_SERVER`, or by 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.

In addition to calculating the delay model, this program computes the baseline vectors,  $u, v, w$  (relative to Earth center on a per-antenna basis) and source elevation vs. time.

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  $u, v, w$  corrections
- F or --fit : Instead of producing an  $n$  term polynomial from  $n$  samples, calculate more samples and perform a fit. This is not of general use as tests have shown that the improvement is negligible.
- z or --allow-neg-delay : don't zero delays that are negative (i.e., shadowed by Earth)
- A or --noatmos : don't include atmosphere in calculation of  $u, v, w$
- 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
- sekido use a version of CALC with the Sekido-Fukushima near-field delay model (use of this is not documented here)

*calcFile* is a `.calc` file (§8.20), such as one generated by `vex2difx` (§7.71)

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

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

Example 3: `calcif2 -a -i 60`

The `--sekido` option is intended as a short term stop gap to provide near-field functionality before `calc11` (with native near-field support) is fully implemented. To this end, documentation of this rather complicated option will likely not get written.

## 7.2 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 `calcif2`, 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.

### 7.3 **checkmpifxcorr** (package : mpifxcorr)

Program **checkmpifxcorr** reads the `.input` and other associated files for a DiFX job and parses them with the same logic used by **mpifxcorr** in order to determine their validity.

Usage: **checkmpifxcorr** *configFile*

All of the files referenced from the provided `.input` file are read as well (excepting any baseband files or the `.vex` file). This check has proven especially useful for pulsar processing.

### 7.4 **checkdir** (package : mk5daemon)

Program **checkdir** can be used to check the integrity of one or more `.dir` files that are stored at a location pointed by environment variable `MARK5_DIR_PATH`. Even after many years of use, the Mark5 units tend to be a weak point in the reliability of correlation. Since reading the module directory and examining a bit of data from each scan are the first actions done to a module, many of the possible problems show up at this time. This utility looks for a number of possible problems, including scans that could not be decoded, overlapping or out-of-order scans, scans with illegal format parameters and others. This program makes no attempt to fix problems. It is up to the operator to determine if a problem is real or not and if further action should be taken. In cases where many scans are not properly decoded it is worthwhile to rename (or remove) the `.dir` file in question and regenerate the directory. A second directory read often succeeds when a first one does not.

Usage: **checkdir** [ *options* ] [ *module list* ]

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : be more verbose in execution (-v -v for more)
- q or --quite : be less verbose in execution
- a or --all : run on all files in `$MARK5_DIR_PATH`
- s or --show : print the entire directory file to screen
- H or --histogram : print a histogram of record rates

Example 1: **checkdir -a**

Example 2: **checkdir NRA0-123 NRA0+266**

Example 3: **checkdir -s NRA0+233**

Either `-a` or a list of module names can be provided (but not both simultaneously). If the former, a less verbose output will be generated by default. Except in the lowest verbosity mode (the default for `-a`), module directories without any detected problems will show a one line summary consisting of the number of scans and the time range of the module.

### 7.5 **condition** (package : difx\_db)

This is an NRAO-only program owing to its ties to the VLBA database.

Program **condition** is mainly used to extract Mark5 module conditioning reports from the database but also has the means to manually import data into the database. When querying (with the `find` action), one or more “identifiers” can be supplied which can be either the names of the Mark5 modules or serial number of individual disks (or a mix of the two!). Environment variable `VLBA_DB` must be set to point to the correct database.

Usage: `condition` [ *options* ] *action* + *args*

*options* can be:

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

`-v` or `--verbose` : be verbose in execution

*action* can be one of:

add *report1* [*report2* ... ]

find *identifier1* [*identifier2* ... ]

*report* is the name of a file containing one or more condition reports from `SSErase`

*identifier* is either a Mark5 module VSN or a hard disk serial number

Example 1: `condition add NRA0-040`

Example 2: `condition find NRA0-042`

Example 3: `condition find NRA0+342 NRA0+270`

Example 4: `condition find Y66M3BQE`

## 7.6 `condition_watch` (package : difx\_db)

This is an NRAO-only program owing to its ties to the VLBA database.

Program `condition_watch` is meant to run as a background process on the correlator head node. Its function is to receive `Mark5ConditionMessages` emitted by a special version of `SSErase` (the module conditioning program) and stuff this data into the database. This program is automatically started by `mk5daemon` when it is supplied with the `-w` or `--condition-watch` arguments. When restarting `mk5daemon` by hand, make sure that a duplicate copy of `condition_watch` is not left running. Environment variable `VLBA_DB` must be set to point to the correct database.

Usage: `condition_watch` [ *options* ]

*options* can be:

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

Example: `condition_watch`

## 7.7 `cpumon` (package : difxmessage)

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 60 characters wide and is at least as tall as there are computers that may transmit information. To quit, use `ctrl-C`. 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)
6. Number of CPU cores

## 7.8 db2vex (package : difx\_db)

This is an NRAO-only program owing to its ties to the VLBA database.

Python program `db2vex` adds three tables to a `vex` file based on data found within the VLBA database and VLA databases. For legacy VLBA/HSA projects, the Oracle database is accessed using the `cx_Oracle` library to retrieve information from the `TAPELOG_OBS`, `CLOCK`, and `EOP` tables. For RDBE VLBA/GBT projects, a new Postgres VLBA database (`vlbampts`) using the EVLA monitor data format is accessed. For projects containing the (modern) phased-VLA, the EVLA database (`evlampts`) is accessed for similar information.

This program should be run after the Earth Orientation Parameters (EOPs) for a project have reached a suitable maturity and after the clocks for the antennas have been inserted into the (legacy) database. If a preliminary run of this program is made for the purposes of doing a clock search correlation pass it is recommended that the output file (a `vex` format file with the new tables attached) get renamed to indicate that this is not to be used to derive jobs for production correlator passes.

Usage: `db2vex [ options ] vexFile [ fsLog ... ]`

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : be verbose in execution
- `-f` or `--force` : force program to overwrite existing `db2vex` output files
- `-p` or `--preobs` : don't look for VSNs, but get clock and EOP values (used for USNO observations by `goUSNO`)
- `-a` or `--mark5a` : force to run in Mark5A mode
- `-c` or `--mark5c` : force to run in Mark5C mode
- `-u` or `--usno` : force to run in USNO Mark5C mode
- `--vsnProjectCode projectCode` : use *projectCode* when getting VSNs from the database

*vexFile* is the name of the `vex` file produced by `sched`

*fsLog* is a field system log file; multiple such files can be provided

Example 1: `db2vex bx123.skd`

Example 2: `db2vex -v gb070.vex gb070nt.log gb070mc.log`

The output of `db2vex` will be another `vex` format file called *vexFile.obs* . This output file is to be used by `vex2difx` for conversion into jobs to run by `mpifxcorr` . The `--vsnProjectCode` was intended for a pathological situation where a project code had to be renamed.

Notes:

1. Environment variable `VLBA_DB` must be set to point to the Oracle database in question.
2. Mark5C observations require environment variable `VLBAMPTS_DB` to point to the new EVLA-style VLBA database
3. Observations using the phased VLA require environment variable `EVLAMPTS_DB` to point to the EVLA monitor database

## 7.9 diffDiFX.py (package : vis2screen)

Program `diffDiFX.py` generates a context-sensitive difference of two DiFX output files for detailed version testing. Corresponding visibility records are differenced and statistics on the differences are accumulated and printed at the end of the processing.

Usage: `diffDiFX.py [ options ] { difxfile1 difxfile2 }`

*options* can be:

- h or --help : print usage information and exit
- f *FREQ* or --freq=*FREQ* : Only look at visibilities from this *FREQ* index
- b *BASELINE* or --baseline=*BASELINE* : Only look at visibilities from this *BASELINE* num
- t *THRESHOLD* or --threshold=*THRESHOLD* : Display any difference that exceeds *THRESHOLD*
- e *EPSILON* or --epsilon=*EPSILON* : Display any differences that exceeds allowed numerical error *EPSILON*
- s *SKIPRECORDS* or --skiprecords=*SKIPRECORDS* : Skip *SKIPRECORDS* records before starting comparison
- m *MAXRECORDS* or --maxrecords=*MAXRECORDS* : Stop after comparing *MAXRECORDS* (if  $\neq 0$ ) records
- p *PRINTINTERVAL* or --printinterval=*PRINTINTERVAL* : Print a summary every *PRINTINTERVAL* records
- c *MAXCHANNELS* or --maxchannels=*MAXCHANNELS* : The length of the array that will be allocated to hold vis results
- v or --verbose : Turn verbose printing on
- i *INPUTFILE* or --inputfile=*INPUTFILE* : Parse *INPUTFILE* for the correlation setup
- matchheaders : On seeing a header mismatch, skip through file 2 looking for next match

*difxfile1* is the first difx file to compare

*difxfile2* is the second difx file to compare

Example: `diffDiFX.py -i example_1.input example_1.difx/DIFX_55523_025239.s0000.b0000 comparison_1.difx/DIFX_55523_025239.s0000.b0000`

If the error for any record exceeds the specified threshold a verbose error message is printed. Summary statistics are printed at the end of the file. Warnings are printed if the headers do not match between the two files.

## 7.10 difx2fits

Program `difx2fits` creates a FITS output file from the native output format created by `mpifxcorr` and several other files carrying information about the observation. Multiple input file sets can be specified. A separate output FITS file is created for each unique frequency setup encountered. When run, `difx2fits` requires the following files to be present for each DiFX file set being converted:

1. *baseFilename.difx/*
2. *baseFilename.input*
3. *baseFilename.calc*

4. *baseFilename.im*

Several other files are optional and are typically used to populate calibration and ancillary tables:

1. *baseFilename.flag*
2. *flags*
3. *pcal*
4. *tsys*
5. *weather*
6. *\$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 or in the place indicated by the `.input` file. As the visibility file (`.difx`) is read, any records that are all zero are omitted.

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

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-n` or `--no-model` : don't write model (ML) table
- `-s` *scale* or `--scale` *scale* : scale visibility data by *scale*
- `-t` *interval* or `--deltat` *interval* : generate `.jobmatrix` file with time intervals of length *interval* seconds
- `-S` or `--sniff-all` : sniff all bins and phase centers, not just the first
- `-T` *interval* or `--sniff-time` *interval* : use *interval* as the sniffer time resolution
- `-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
- `-k` or `--keep-order` : don't sort the antennas by name
- `-1` or `--dont-combine` : make a separate FITS file for each input job
- `-x` or `--dont-sniff` : don't generate sniffer output files
- `--phasecentre` *p* : (U.S. spelling okay too) Create a FITS file for all the *p*<sup>th</sup> phase centers (default 0)
- `--override-version` : ignore difx version clashes

*baseFilename* is the prefix of the jobfile to convert; it is okay 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 dq109_1 DQ109.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.

Unless disabled by setting `interval` to a non-positive number with the `-t` or `--deltat` option, an output file with suffix `.jobmatrix` will be produced. This file contains an ASCII art diagram of which jobs contributed to each `.FITS` file produced as a function of both time and antenna.

If submitting a bug report for `difx2fits`, please include in it the full output of `difx2fits -v -v` and the `.input` and `.calc` files.

`difx2fits` displays several diagnostics during the conversion process, separately for each output FITS file. The size of each FITS table is printed; a zero size indicates that table is not produced. For the visibility table, input files contributing to the output are printed. Scan information is printed at increased verbosity levels. Not all DiFX output visibilities are written to the FITS file. Accounting of the disposition of the visibilities is provided. Invalid records are those containing infinite or NaN values and indicate a likely bug in the software. Flagged records are those identified by `vex2script` (or other DiFX file set generation programs) in the `.flag` file as being illogical, such as cases where a particular baseline during a job belongs to a different subarray. When using integration times longer than 1 second it is possible for one visibility to span two scans. Such records are dropped. Finally any visibilities, produced outside a normal scan start/stop time are dropped; this should not occur unless the `.calc` file is modified between correlation and FITS creation.

## 7.11 difx2mark4

Program `difx2mark4` creates a Mark4 output file set from `mpifxcorr` input and output files. When run, `difx2mark4` requires the following files to be present for each file set being converted:

1. `baseFilename.difx/`
2. `baseFilename.input`
3. `baseFilename.im`

as well as the `.vex` file referenced in the `.input` file, which may be common to many DiFX file sets.

Usage: `difx2mark4 [ options ] baseFilename1 [...baseFilenameN]`

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-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
- `-k` or `--keep-order` : don't sort the antennas by name
- `--override-version` : ignore difx version clashes
- `-r` or `--raw` : suppresses normalization of amplitudes
- `-p` or `--pretend` : do a dry run
- `-e` or `--experiment-number n` : set the experiment number to *n* which must be a 4 digit number (default is 1234)
- `-b code flo fhi` : Override freq band codes. Frequencies are in MHz. Multiple parameters of this kind can be specified.

*baseFilename* is the prefix of the jobfile to convert *without* the underscore and job number

Example `difx2mark4 dq109`

## 7.12 difxarch (package : difx-db)

Program `difxarch` is a simple script that moves FITS files produced by `makefits` from the correlation queue staging area (defined by the `DIFX_QUEUE_BASE` environment variable) to the archive staging area (defined by environment variable `DIFX_ARCHIVE_ROOT`). A process running on the archive computer will periodically monitor new files in this staging area and will then copy them to the actual archive. In order to prevent premature pick-up of these files, they are first moved into a directory with a name beginning with a period (`.`). This directory is renamed without the period once all files to be archived are copied.

Usage: `difxarch` [ *options* ] *passName1* [ *...passNameN* ]

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : increase verbosity of output
- `-p` or `--pretend` : generate SQL and bash commands but don't execute them
- `--override-version` ignore DiFX version clashes

*passName* is the name of a correlator pass; a file called *passName.fitslist* is expected to be present

Example: `difxarch -v clock`

## 7.13 difxbuild

Program `difxbuild` aids in the installation of DiFX onto a cluster. Full documentation on the install process can be found in §12.1, so details will not be shown here. Command syntax is as follows:

Usage: `difxbuild` [ *options* ] *command* [ *command arguments* ]

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-d` or `--documentation` : print full in-line documentation to screen
- `-t` or `--todo` : print developer's to-do list
- `-v` or `--verbose` : increase verbosity of output
- `-q` or `--quiet` : decrease output verbosity
- `-p` or `--pretend` : generate SQL and bash commands but don't execute them
- `-V` or `--version` : print version and quit

*command* is one of the `difxbuild` commands, such as `build` or `svn`; the program help information will list all options

*command arguments* are options for some commands

## 7.14 difxcalculator (package : difxio)

Program `difxcalculator` looks at a set of DiFX input files (`.input`, `.calc`, etc.) and reports/calculates key operating parameters. This program is inspired by the `difx_calculator.xls` spread sheet available at <http://cira.ivec.org/dokuwiki/doku.php/difx/calculator>.

Usage: `difxcalculator` [ *options* ] *baseName* [*speedUp*]

*options* can be:

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

*baseName* is the base name of a correlator job

*speedUp* is the expected processing rate relative to real-time

Example: `difxcalculator mt933_01`

Known bugs:

1. Does not take into consideration multiple phase centers or zoom bands.

### 7.15 `difxclean` (package : difx\_db)

Program `difxclean` simply deletes all data from `$DIFX_QUEUE/` for a particular project. It also removes all jobs with status not equal to `COMPLETE` for this project from the `DIFXQUEUE` table of the database. It is intended that it be run at the same time the project is released, meaning data has been correlated and archived. The user does not have to be in any particular directory when running this program.

Usage: `difxclean [ options ] project`

*options* can be:

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

`-p` or `--pretend` : don't actually do the erasure

*project* is the name of the project to be "cleaned" out

Example: `difxclean mt917`

### 7.16 `difxcopy` (package : difx\_db)

Python program `difxcopy` is used to copy DiFX input (and other) files to a different directory. In the process, explicit references to other files that are being copied are changed to reflect their new file system path. For a given file prefix, *prefix*, the following files are copied if they exist: *prefix.input*, *prefix.calc*, *prefix.flag*, *prefix.delay*, *prefix.uvw*, *prefix.rate*, and *prefix.im*. The *.vex* file referenced within the *.calc* file is also copied.

Usage: `difxcopy [ options ] jobPrefix1 [...jobPrefixN] destDir`

*options* can be:

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

`-v` or `--verbose` : possibly increase verbosity of output

*jobPrefixN* is the file prefix of a job, e.g., `mt911_04` would be the prefix corresponding to input file `mt911_04.input`

*destDir* is the directory in which the copied and modified files will be placed

Example: `difxcopy mt911_02 mt911_03 mt911_04 /home/difx/queue/MT911`

## 7.17 difxlog (package : difxmessage)

Program `difxlog` can be used to collect DiFX multicast messages for a particular correlator job and write them to a file. A new instance of `difxlog` must be started for each job being run. Normally this will be done automatically if implemented in the particular deployment of DiFX. Both `startdifx` and `mk5daemon` can start DiFX and will instantiate a `difxlog` process as needed. Only messages of type `DifxAlertMessage` and `DifxStatusMessage` are collected and written to the output file.

Usage: `difxlog jobIdentity outFile [ logLevel pidWatch ]`

*jobIdentity* : the name of the job being run (specifically, it should match the `identifier` field in the `DifxMessage` being sent).

*outFile* : the name of the output file containing log information.

*logLevel* : the minimum message severity to retain (see §9.1).

*pidWatch* : the program id (in the Unix sense) of the `mpifxcorr` process running.

Example: `difxlog mt911.04 mt911.04.difxlog 4 1243`

Unless a *pidWatch* value is specified, `difxlog` will run until killed. If a *pidWatch* value is provided, `difxlog` will quit as soon as that process stops running. The *loglevel* parameter can be used to select the maximum severity level to write to the log. The possible values and their meanings are:

0	Fatal	<code>mpifxcorr</code> cannot continue because of the noted problem
1	Severe	an internal error that should never happen happened (likely bug)
2	Error	a problem was encountered in the data processing
3	Warning	something suboptimal was noted
4	Informative	a note containing progress information
5	Verbose	more detailed progress information
6	Debug	values probably of use only to software developers

## 7.18 difxqueue (package : difx\_db)

This is an NRAO-only program owing to its ties to the VLBA database.

Python program `difxqueue` is a program used to maintain the DiFX correlator queue. There are two main responsibilities in doing so: copying or deleting files in the correlator queue directory (which is project specific: `$DIFX_QUEUE_BASE/projectName`) and maintaining the database entries for each queued job. In the VLBA context, this program is the main interface between the analysts and the correlator operators. This program is mainly intended to work on one *job pass* at a time rather than single jobs or whole projects. In some cases one job pass could be one job, or it could be a whole project (or both), but in many cases there will be multiple passes per project with possibly multiple jobs per pass. It is possible for `difxqueue` to operate on individual jobs when a list of job numbers is provided. The first command line argument describes the action to perform. Each subsequent argument is then context dependent; see the examples or run with the `-h` command line parameter to get a feel for the variety of options allowed. Once a job has been correlated successfully, its status will be `COMPLETE`. There is no need to `delete` a job from the queue once it is complete. Doing so will require recorrelation if the results of that job are still needed. Each job in the queue has a priority. The smaller the priority, the lower the number. By default a queued job will have priority 2.

Usage: `difxqueue [ options ] action [ args ]`

*options* can be:

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

`-p priority` or `--priority priority` : set the priority of jobs to *priority*  
`-q queuedir` or `--queuedir queuedir` : manually set the staging directory  
`-v` or `--verbose` : increase verbosity of output  
`-d` or `--db-only` : do not copy/delete/move files; operate only on database  
`--override-version` : ignore DiFX version clashes

*action* : the action to perform

**add** : add job(s) to the queue, usually a whole pass at a time. Default priority is 2; use the `-p` option to set the priority if a different priority is required.

Example 1: `difxqueue add clock`

Example 2: `difxqueue add mt911 1 2 3 4`

Example 3: `difxqueue -p 3 add geodesy`

**bump** : increase the priority of queued job(s)

Example: `difxqueue bump clock`

**del** : remove job(s) from the queue

Example 1: `difxqueue del clock`

Example 2: `difxqueue del mt911 2 3`

**list** : list all jobs within a pass

Example: `difxqueue list mt911`

**listall** : list all incomplete jobs in the queue; note that this is not restricted even to any particular project. If one or more projects is specified, all jobs, complete or not, for those projects will be listed. If no segment code is appended to a project name, then all matching proposal codes will be listed.

Example 1: `difxqueue listall`

Example 2: `difxqueue listall BX123 BY321`

Example 3: `difxqueue listall BR138A`

**log** : list all correlations that have happened for a given project. This simply searches the DIFXLOG database table and dumps it to the screen in a readable fashion.

Example 1: `difxqueue log BX123`

**prod** : print production queue list, possibly sending to a file

Example 1: `difxqueue prod`

Example 2: `difxqueue prod queue.txt`

**set** : set the status of queued job(s)

Example 1: `difxqueue set tc015d COMPLETE`

Example 2: `difxqueue set tc015d QUEUED 3 4`

**slide** : decrease the priority of queued job(s)

Example: `difxqueue slide mt911 6`

*args* : *action* dependent arguments, usually a pass name and possible list of job numbers

Note that except for the `listall` and `prod` actions, the current working directory must contain the `.joblist` file for a project.

## 7.19 difxsniff (package : calcif2)

Program `difxsniff` is a reimplementaion of the VLBA analysts' 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* *FITS1* [ ... *FITSN* ]

*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 2: `difxsniff NL FD *.FITS`

## 7.20 difxspeed (package : vex2difx)

Program `difxspeed` does processing benchmarking, possibly over a range of parameters, of DiFX. To ensure that data playback (reading from files, Mark5 modules or network) are not limiting performance, the FAKE mode of DiFX (see §??) is used; thus the output data are meaningless. `difxspeed` takes a `.speed` (§8.10) file as input. This file contains various parameters, many of which are identical to those in the `.v2d` (§8.36) files. An important difference with the parameters specified in `.speed` files is that multiple values can be provided for many of the parameters. In the benchmarking process, a separate run of DiFX for each combination of the supplied parameters is performed. The first combination is run twice, with the first being labeled a *dummy* run. This is because the timing of the first execution can vary depending on recent usage of the correlator.

Usage: `difxspeed` [ *options* ] *inputFile* [ *numIterations* ]

*options* can be:

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

*inputFile* is the `.speed` file describing the series of benchmarks to run

*numIterations* is the number of times to execute all test combinations

Each run of `difxspeed` will append a new column of data to a file called `inputFile.out` ; if the file does not exist, a new file will be created. Documentation of this output file format can be found in §8.11.

## 7.21 difxusage (package : difx.db)

This is an NRAO-only program owing to its ties to the VLBA database.

Program `difxusage` mines the VLBA database for correlator usage statistics. Usage is as follows:

Usage: `difxusage` [ *options* ] *mjdStart* *mjdStop*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : be more verbose in execution
- l or --list : print all matching jobs
- a or --all : select jobs in all states
- c or --complete : select only complete jobs (default)
- k or --killed : select only killed jobs
- u or --unknown : select only unknown jobs

*mjdStart* is the start time (in Modified Julian Days) to look for jobs

*mjdStop* is the stop time (in Modified Julian Days) to look for jobs

Note that environment variable `VLBA_DB` must be set to point to the Oracle database in question.

## 7.22 DiFX Operator Interface

The DiFX Operator Interface (DOI) is a java-based application to monitor and control the correlation of DiFX jobs. Jobs to be correlated can be selected with a file browser or retrieved with a database request. A separate manual [9] will be made available with instructions for its use. The only specific detail that will be mentioned here is the contorted path the starting of a job takes:

1. The job to run is selected.
2. The DOI determines which resources (Mark5 units and processor nodes) are required.
3. If the intended output file already exists, a dialog will ask the operator whether to overwrite this file or not.
4. The DOI allocates resources.
5. The DOI assembles a `DifxStartMessage` XML document and multicasts it with the correlator head node as the recipient.
6. The `mk5daemon` process running on the head node captures this message.
7. `mk5daemon` fork(s); the child process changes its `userId` to `difx` and spawns an `mpirun` process via `ssh` to ensure the proper environment variables are set.
8. The `mpirun` process starts a copy of `mpifxcorr` on each of the Mark5 units and processing nodes that is requested.
9. `mk5daemon` fork(s) again; the child process changes its `userId` to `difx` and spawns a `difxlog` process via `ssh` to ensure the proper environment variables are set.
10. All processes continue until job end is reached or the job is killed.
11. When the first fork(ed) `mk5daemon` process ends, the `difxlog` process stops automatically, causing the second fork(ed) process also to stop.
12. The DOI receives messages suggesting the job has ended and frees the allocated resources.

### 7.23 **e2ecopy** (package : difx\_db)

Program **e2ecopy** copies files one directory to another, changing the ownership to `$DIFX_ARCH_USERNAME` in the process. This program must be setuid root; the person installing the program must run `chmod +s e2ecopy` after installation if `make install` is not run by root. Normally this program is run by `difxarch` (see §7.12). This is a VLBA-centric program, but could be used by others.

Usage: **e2ecopy** [ *options* ] *fromDir toDir file1* [ *...* *fileN* ]

*options* can be:

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

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

*fromDir* : the source directory of the file(s) to copy

*toDir* : the destination directory

*file* : a file to copy (multiple files may be provided)

Note: “e2e” is NRAO terminology for “End to End”, a philosophy of providing user software covering the full project lifecycle from proposal handling to archive access. In this particular case the name arose due to the location of the archive staging area at NRAO.

### 7.24 **errormon** (package : difxmessage)

Program **errormon** listens for multicast messages of the `difxError` variety and simply prints their contents to the terminal. It is effectively the same as `difxlog` except that log data is sent to `stdout` rather than a systematically named file.

Usage: **errormon** [ *options* ] [*maxSeverity*]

*options* can be:

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

*maxSeverity* : maximum severity level to display (default = 8)

See § 9.1 for a list of severity codes. If no *maxSeverity* is provided, the default level of 8 will cause no selection to occur; all messages will be printed.

A similar program, **errormon2**, does nearly the same thing, but defaults to a less verbose output, and sends output to `stderr` rather than `stdout` (so use with `grep` or other \*nix tools is more cumbersome. It also writes its output to a log file.

See documentation for `difxlog` to see the list of alert levels.

### 7.25 **genmachines** (package : calcif2)

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 (§8.25) 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 ] input1 [ ... inputN ]`

*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
- `-m file` or `--machinesfile file` : use *file* instead of `$DIFX_MACHINES`
- `-n` or `--no-threads` : don't write a `.threads` file.
- `-d` or `--difxdb` : lookup module locations in a database.

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

## 7.26 `getshelf` (package : difx\_db)

This is an NRAO-only program owing to its ties to the VLBA database.

Program `getshelf` retrieves the shelf location of specified modules from the legacy VLBA database and prints them to the screen. While possibly useful, this program is not required for the software correlation process.

Usage 1: `getshelf [ options ] module1 [ module2 [ ... ]]`

Usage 2: `getshelf [ options ] shelfFile`

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : print the database query string as well

*moduleN* is the volume serial number (VSN) of a module; multiple modules can be specified

*shelfFile* is a `.shelf` file (§8.31), as may be written by `db2vex` Default is the current working directory if none is provided.

Example 1: `getshelf NRAO+267`

Example 2: `getshelf bx123a.skd.shelf`

## 7.27 `jobdisks` (package : calcif2)

Program `jobdisks` looks through job files to see which modules (disks) are needed for correlation. It reads through `.input` files, as used by `mpifxcorr`, to get the needed information. 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 ] [file1]...[fileN]`

*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`

Known bugs:

1. Program should make sure datastream type is MODULE.

## 7.28 `joblist` (package : calcif2)

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 (§8.7)
m .machines file (§8.25)
t .threads file (§8.32)
i .im file (§8.19)
v .difx file (§8.8)
```

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 ] [ dir1 ]...[ dirN ]`

*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/*`

## 7.29 `jobstatus` (package : calcif2)

**Warning:** As of DiFX 2.0.1, this utility has not yet been updated to work with DiFX 2 output format.

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* ] [ *dir1* ] ... [ *dirN* ]

*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/*`

Known bugs:

1. This program has not been updated to work with DiFX 2.0 output

### 7.30 `listcpus` (package : calcif2)

Python program `listcpus` uses `ssh` to connect to each machine listed in a file (usually `$DIFX_MACHINES`) and peaks at the list of CPUs on that machine and prints to `stdout`. Only the first column of this file is used and any content after a `#` is ignored. For each CPU on the machine, the model name, which usually also contains the CPU speed, is listed. For multi-core CPUs, each core will appear as its own CPU.

Usage: `listcpus` [ *options* ]

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : increase output verbosity
- `-m file` or `--machines file` : use *file* instead of `$DIFX_MACHINES` for list of machines to probe

Multiple directories can be specified.

Example 1: `listcpus`

Example 2: `listcpus -m myCPUs.list`

### 7.31 `makefits` (package : difx2fits)

This program is tuned for NRAO use; some modifications may be required for use at other sites.

Program `makefits` is basically a wrapper for `difx2fits` (§7.10) that does some sanity checking and ensures that files end up in the proper places with the proper names. This program is intrinsically *pass-based* and it bases its functionality on the `.joblist` (§8.21) file that is written by `vex2difx` (§7.71). One must run this program on the software correlator head node (`swc000` in the current VLBA DiFX implementation). Upon successful completion, FITS-IDI files are created in the same directory in the correlator job staging area (`$DIFX_QUEUE_BASE/projectName`) and the sniffer output files are left in a subdirectory of the current working directory. An additional output file is left in the current working directory called `passName.fitslist`. This file has a list of the FITS files that are to be archived once the data for this pass are deemed valid.

The checks that `makefits` performs will by default not allow an incomplete set of FITS files to be produced. This can be overridden with a special command line argument (below). This is part of an accountability chain that aims to ensure that nothing gets omitted.

Usage: `makefits` [ *options* ] *passName*

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : increase output verbosity
- `--override-version` : ignore potential difx version conflicts
- `--allow-partial` : bypass check for complete set of correlated output and proceed

Multiple directories can be specified.

Example: `makefits clock`

### 7.32 **makemark4** (package : difxdb)

This program is tuned for NRAO use; some modifications may be required for use at other sites.

Program **makemark4** is essentially a wrapper for **difx2mark4** (§7.11) that does some sanity checking and ensures that files end up in the proper places with the proper names. This program is intrinsically *pass-based* and it bases its functionality on the **.joblist** (§8.21) file that is written by **vex2difx** (§7.71). One must run this program on the software correlator head node (**swc000** in the current VLBA DiFX implementation). Upon successful completion, Mark4 file sets are created in the same directory in the correlator job staging area (**\$DIFX\_QUEUE\_BASE/projectName**). An additional output file is left in the current working directory called *passName.mark4list*. This file has a list of the Mark4 file sets that are to be archived once the data for this pass are deemed valid.

Usage: **makemark4** [ *options* ] *passName*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : increase output verbosity

Example: **makemark4** rdv95

### 7.33 **m5bstate** (package : mark5access)

Program **m5bstate** will perform state counts on a baseband data file.

Usage: **m5bstate** *file format nFrames* [ *offset* ]

*file* is the file to decode

*format* is the format of the data

*nFrames* is the number of data frames (typically a few kB in size) to decode

*offset* is the number of bytes into the file to start decoding

Example: **m5bstate** sample.vlba VLBA1\_2-256-8-2 100

Notes:

1. See documentation for **m5b** for details on specifying the data format.
2. Only real-sampled data with 1 or 2 bits per sample is supported at this time.
3. In the case of VDIF data, only single thread data with  $2^n$  channels is supported. For equivalent functionality in the multi-thread VDIF case see **vdifd**.

### 7.34 **m5d** (package : mark5access)

Program **m5d** is a very simple example program using the mark5access decoding library. It turns out to be useful enough as a stand-alone program to be separately documented. This program takes as command line input the name of a file containing (or thought to be containing) VLBI baseband data, the expected format of the data, and the number of samples per baseband to decode. Optionally a starting file offset can be supplied. If the data can be decoded correctly, information about the data will be printed to the screen along with a table of decoded data. The output values, -3, -1, 1, or 3 for valid data, are printed in *nchan* columns. Data that cannot be decoded (either due to data replacement headers, data fill pattern replacing the actual data after unloading from a Mark5 module, or identified via the VDIF invalid bit) will show as 0. It should be invoked with the following parameters:

Usage: `m5d file format n [ offset ]`

*file* is the file to decode

*format* is the format of the data

*n* is the number of samples to decode

*offset* is the number of bytes into the file to start decoding

Example 1: `m5d sample.vlba VLBA1_2-256-8-2 24`

Example 2: `m5d sample.mk4 MKIV1_4-128-2-1 600 200`

Example 3: `m5d sample.5b Mark5B-512-16-2 1200`

The format parameter is constructed from four parts as *type-rate-nchan-nbit* where:

*type* is the type of format and should be one of VLBA1\_1, VLBA1\_2, VLBA1\_4, MKIV1\_1, MKIV1\_2, MKIV1\_4, Mark5B, or VDIF

*rate* is the data rate in Mbps

*nchan* is the number of baseband channels

*nbit* is the number of bits per recorded sample

See the usage examples above for some explicit values. Note for the VLBA and MKIV format types the fanout is appended as this affects the decodability of the files.

Notes:

1. In the case of VDIF data, only single thread data with  $2^n$  channels is supported. For equivalent functionality in the multi-thread VDIF case see `vdifd`.

### 7.35 `m5findformats` (package : mark5access)

Program `m5findformats` attempts to determine which format a baseband data file may be. Currently it searches over 16 to 2048 Mbps data rates in factors of 2 and checks only for MKIV, VLBA and Mark5B types.

Usage: `m5findformats filename`

- *filename* is the name of the baseband data file.

Run with no command line arguments to get help information.

### 7.36 `m5fold` (package : mark5access)

Program `m5fold` takes a baseband data stream and integrates the power (formed by squaring the voltage) in a number of time bins that equally divide a given period. This is a simplified version of “folding” such as is used in pulsar processing. A typical use of such functionality would be to investigate the waveform of the switched power injected into the receiver for calibration. This program has found considerable utility in determining time offsets between the sample clock and formatter time (modulo the period of the calibration cycle). In the case of 2-bit sampling a non-linear correction is applied before results are written to a file. This correction takes the form

$$P = \frac{1}{\left(\operatorname{erf}^{-1}\left(\frac{P-v_{\text{high}}^2}{1-v_{\text{high}}^2}\right)\right)^2}, \quad (2)$$

where  $P$  is a value proportional to true power and  $\hat{P}$  is the value obtained by calculating  $\langle \hat{v}^2 \rangle$  when the bitstream is reproduced with values  $\hat{v} \in (-v_{\text{high}}, -1, 1, v_{\text{high}})$ . This non-linear correction can be turned off by setting  $nbin$  to a negative value. Note that this program is not useful for 1-bit quantized data. The program should be used as follows:

Usage: `m5fold infile format nbin nchunk freq outfile [ offset ]`

*infile* is the file to decode

*format* is the format of the data

*nbin* is the number of bins to calculate per period; if negative, power correction is not performed and the absolute value of *nbin* is used

*nchunk* is the number of 10000 sample chunks to operate on

*freq* is the reciprocal of the period to be observed (Hz)

*outfile* is the name of the output file

*offset* (optional) is the number of bytes into the file to start decoding

Example: `m5fold sample.vlba VLBA1.2-256-8-2 128 10000 80 sample.fold`

See the documentation for `m5d` for information on specifying the data format.

The output file will contain  $nchan+1$  columns where  $nchan$  is the number of baseband channels in the data stream. The first column contains the time (seconds) within the period. Each remaining column is folded power for one baseband channel. If  $nbin$  is positive and the data is 2-bit quantized, the scaling is such that  $\langle v^2 \rangle = \sigma^2$  yields a power reading of 1.0, for sampler threshold  $\sigma$ . Optimal signal to noise ratio occurs for a value of about 1.03. For non 2-bit quantization, the power will be in units of reconstituted counts<sup>2</sup>.

In the case of VDIF data, only single thread data with  $2^n$  channels is supported. For equivalent functionality in the multi-thread VDIF case see `vdiffold`.

### 7.37 `m5pcal` (package : mark5access)

Program `m5pcal` can be used to extract pulse cal tones from baseband data in Mark4, VLBA, Mark5B and single-thread VDIF formats.

Usage: `m5pcal [ options ] infile format freq1 [ freq2 [...] ] outfile`

*options* can be:

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

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

`-q` or `--quiet` : decrease output verbosity

`-n n` : loop over  $n$  chunks of data (default is 1000)

`-N N` : perform  $N$  outer loops, each yielding a result set

`-o o` or `--offset o` : jump  $o$  bytes into file

`-i i` or `--interval i` : use pulse cal comb interval of  $i$  MHz (default is 1)

`-e e` or `--edge e` : don't use channels closer than  $e$  MHz from band edges when computing delay (default is 1/8 of bandwidth)

*infile* is the file to decode

*format* is the format of the data‘

*freq1* . . . is/are the frequencies (MHz) relative to baseband of the first tone to detect; there should be one *freq* specified per baseband channel

*outfile* is the name of the output file

See the documentation for `m5d` for information on specifying the data format.

### 7.38 `m5spec` (package : mark5access)

Program `m5spec` is an example program using the `mark5access` decoding library that is a bit more advanced than the `m5d` program is. It forms total power spectra for each baseband channel in the data, including cross spectra for polarization pairs, assuming data is in alternating polarization pairs (if not, the cross spectra should make no sense, but they are formed anyway). The results are written to a text file with the following columns: Column 1 is the frequency offset from baseband for each channel; Columns 2 to  $nchan+1$  are the total power spectra for each baseband channel; Columns  $nchan+2$  to  $4\times nchan+1$  contain, in pairs, the amplitude and phase of the cross spectra for each pair of channels. It should be invoked with the following parameters:

Usage: `m5spec infile format npoint n outfile [ offset ]`

*infile* is the file to decode

*format* is the format of the data

*npoint* is the number of points to calculate for each spectrum

*n* is the number of FFT frames to include in the calculation

*outfile* is the name of the output file

*offset* (optional) is the number of bytes into the file to start decoding

Example: `m5spec sample.vlba VLBA1.2-256-8-2 256 1000 vlba.spec`

See the documentation for `m5d` for information on specifying the data format.

In the case of VDIF data, only single thread data with  $2^n$  channels is supported. For equivalent functionality in the multi-thread VDIF case see `vdifspec`.

### 7.39 `m5test` (package : mark5access)

Program `m5test` is an example program using the `mark5access` decoding library that works its way through a VLBI baseband data stream attempting to decode data and header information to look for problems. Every million samples (per baseband channel) a summary line containing frame number, decoded date and time, and counts of valid and invalid frames are shown. After 20 invalid frames are encountered the program will stop. Otherwise the program will run until end of file or until interrupted by the user. Usage is as follows:

Usage: `m5test infile format [ offset ]`

*infile* is the file to decode

*format* is the format of the data

*offset* (optional) is the number of bytes into the file to start decoding

Example: `m5test sample.vlba VLBA1.2-256-8-2`

See the documentation for `m5d` for information on specifying the data format.



#### 7.40 **m5time**(package : mark5access)

Program **m5time** decodes the time of the beginning of a Mark4, VLBA, or Mark5B datastream and prints the result in integer MJD and UT hours, minutes, seconds to the screen.

Usage: **m5time** *infile* *format*

*infile* is the file to decode

*format* is the format of the data‘

#### 7.41 **m5timeseries** (package : mark5access)

Program **m5timeseries** produces a power measurements for each of channel of a baseband data file, averaging over a specified time interval.

Usage: **m5timeseries** *infile* *format* *tint* *ntime* *outfile* [ *offset* ]

*infile* is the file to decode

*format* is the format of the data‘

*tint* is the integration time per sample in milliseconds

*ntime* is the number of samples to generate

*outfile* is the name of the output file

*offset* (optional) is the number of bytes into the file to start decoding

Example: **m5timeseries** *sample.vlba* VLBA1.2-256-8-2 6.25 8000 *sample.series*

The output file contains *nchan*+2 columns of data where *nchan* is the number of channels in the data file. The first column is sample number. The second column is time since beginning of series, in seconds. The remaining columns are power measurements for the channels.

#### 7.42 **m5tsys** (package : mark5access)

#### 7.43 **mk5cat** (package : mk5daemon)

This program sends data on a module to standard out. See additional documentation under **mk5cp** which operates on similar principles (**mk5cat** is **mk5cp** writing to *stdout* Note that the executable for **mk5cat** is identical to that for **mk5cp** and only the name of the program actually differs.

Usage: **mk5cat** [ *options* ] { *bank* | *VSN* } *scans*

*options* can be:

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

-v or --verbose : increase verbosity, e.g., print directory to screen

*bank* is either A or B

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

*scans* is one or more scan numbers (starting at 1) with scan numbers separated by commas.

Many of the other baseband data utilities documented here such as **m5d**, **m5spec** and **vmux** can take input from *stdin* and thus can be mated with **mk5cat**. Usually a single hyphen (-) as the name of the input file indicates this to these programs.

Example: **mk5cat** B PT\_BB241\_No0111 | **m5spec** - Mark5B-2048-16-2 128 10000 *methanol.spec*

## 7.44 **mk5control** (package : mk5daemon)

**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 unit1 ... unitN`

*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`; this cures many/most mark5 hangs
- **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
- **StopDir** Stop a directory read that is in progress
- **KillMpifxcorr** Send `sigkill` (like `kill -9`) to all processes on machine called `mpifxcorr`
- **Copy** Copy data from a module to files in a provided directory. At least three parameters must be provided that match the parameters of `mk5cp`. Because of the way `mk5control` parses the command line, the word `copy` and the parameters must all be enclosed in quotes.
- **StopCopy** Stop a data copy process
- **GetVer** Request send of a `DifxMessageMk5Version` XML message
- **mountXX** Cause Linux device `/dev/sdXX` to be mounted on `/mnt/usb`
- **umount** Cause `/mnt/usb` to be unmounted
- **Test** Used in debugging — for developers only

## 7.45 **mk5cp** (package : mk5daemon)

Program **mk5cp** copies baseband VLBI data from a module to a file somewhere on the operating system filesystem, perhaps an external USB disk. This program is often started using **mk5control** to tell the instance of **mk5daemon** running on the desired Mark5 unit to start the copy. Status information is multicast via a **Mark5StatusMessage** document.

Usage: **mk5cp** [ *options* ] { *bank* | *VSN* } *scans* *outputDirectory*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : increase verbosity: print directory to screen

*bank* is either A or B

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

*scans* is one or more scan numbers (starting at 1) with scan numbers separated by commas. No spaces are allowed in the list. A range can be specified with a hyphen (see examples). Alternatively, a scan name, or portion thereof, can be specified. When a partial scan is provided, any scan name matching that partial scan name will be copied. The *scans* parameter can also take either a time range (two floating point modified Julian Days connected with an underscore, or a byte range (must be a multiple of 4) via two integers separated by an underscore, or a byte start and a length can be specified with two integers separated by a plus sign.

*outputDirectory* is the directory to which files will be copied. Make sure the destination directory exists before running this program and make sure sufficient free space remains on that filesystem. If the *outputDirectory* parameter is set to - (the hyphen), data will go to stdout. Another utility called **mk5cat** is derived from this behavior.

Example 1: **mk5cp** NRA0-123 4 /mnt/usb/WaltersProject

Example 2: **mk5cp** A 1,2,3 /tmp

Example 3: **mk5cp** B BC120A /mnt/usb/BC120A/PT

Example 4: **mk5cp** NRA0+255 6-12 /tmp

Example 5: **mk5cp** NRA0+255 123123100.124123100 /tmp

Example 6: **mk5cp** NRA0+255 54327.13124.54327.15124 /tmp

Example 7: **mk5cp** NRA0+322 123123100+1000000 /tmp

## 7.46 **mk5daemon** (package : mk5daemon)

**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**.

- **High level control of Mark5 units**

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 (§7.44).

- **Low level control of Mark5C units**

This program exposes a VLBI Standard Interface (VSI) over TCP port 2620 that very closely emulates equivalent functionality of the Mark5C Data Recording System (DRS) program provided by Haystack observatory. The implementation of the DRS command set is not complete but is sufficient for monitor and control at record time. At the time of writing this program is used in lieu of DRS at the two Mark5C units provided by USNO.

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

Every 10 seconds, `mk5daemon` extracts data from 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 §9 for details on these XML messages.

- **Starting of mpifxcorr**

If `mk5daemon` is started with the `-H` or `--head-node` option, it will be allowed to start new correlations. A correlation will be started when a `difxStart` message is received if it passes some minor sanity checks. Since `mk5daemon` runs as root, it has the capability of changing file ownerships. By default, the output files from `mpifxcorr` and `difxlog` will have their ownership and permissions changed to match those of the `.input` file.

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
- q or `--quiet` : be less verbose and don't multicast state
- H or `--head-node` : give head-node permissions
- m or `--isMk5` : force `mk5daemon` on this host to act as Mark5 regardless of hostname (default is `mark5fx??`)
- u *userID* or `--user userID` : use *userID* when executing remote commands (default is 'difx')

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

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

### 7.47 `mk5dir` (package : mark5daemon)

Program `mk5dir` extracts the directory from a module. Normally one would not call this program directly but would use the `getdir` option of `mk5control`. By default this program will change the disk module state to `played`. There is a danger that if this is done with an SDK 9 unit and the disk is later needed in an SDK 8 unit that it will no longer be readable in the later without a full reset of its VSN.

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
- `-f` or `--force` : force directory read even if current
- `-F` or `--fast` : get format details from new-style directory (Mark5B format only)
- `-n` or `--nodms` : get directory but don't change disk module state
- `-s` or `--safedms` : only change disk module state if SDK 8 unit or new dir type
- `-d` or `--dmsforce` : proceed with change of module state even if this makes module unreadable in SDK 8 units
- `-b b` or `--begin b` : begin with scan number *b*
- `-e e` or `--end e` : end with scan number *b*
- `-w file` or `--write file` : write directory file *file* to the module

*bank* is one of A, B or AB

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

The resultant directory file will be saved in a file called `VSN.dir` in the directory pointed to by environment variable `MARK5_DIR_PATH` .

This program responds to the value of environment variable `DEFAULT_DMS_MASK`. This variable should be an integer representing the state of three bits. `mk5dir` only responds to the setting of bit 1 (value 2); if this bit is set, the disk module state will not be updated on directory reading. It is recommended to set this environment variable at recording stations so auto-erasure of modules does not occur.

In the mode where a specified `.dir` file is to be written to a Mark5 module directory the VSN must be provided explicitly (i.e., selecting by bank is not allowed)

### 7.48 `mk5erase` (package : mark5daemon)

Program `mk5erase` replaces the functionality of `SSErase`. It is used to either erase or condition a Mark5 module. It supports SDK9 and earlier revisions of the Conduant API and either legacy or new (see Mark5 memop 81) module directories. Conditioning results are multicast upon conclusion of conditioning, to be received by `condition_watch`. Conditioning (which is started with the `-c` option) causes an entire read/write cycle of the entire module to be performed. This can require a good fraction of 24 hours to complete. Status updates and progress are sent every 10 seconds as well. By default the original directory version will be restored on the module, with zero scans. The version of directory to use can be forced with either the `-1` or `-n` options.

Usage: `mk5erase [ options ] VSN`

*options* can be:

- `-h` or `--help` : print usage information and exit
- `-v` or `--verbose` : increase verbosity: print directory to screen
- `-f` or `--force` : force directory read even if current
- `-c` or `--condition` : Do full conditioning, not just erasing
- `-r` or `--readonly` : Perform read-only conditioning
- `-w` or `--writeonly` : Perform write-only conditioning
- `-d` or `--getdata` : Save the performance data to a file called `VSN.timedata`
- `-l` or `--legacydir` : Put an empty legacy directory on the module when complete
- `-n` or `--newdir` : Put an empty new style directory on the module when complete
- `-0` or `--nodir` : Put a zero-size directory on the module when complete

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

Note that this program will not run without specifying a legal mounted module VSN. If you wish to erase a module that has no VSN set, use the `vsu` program first.

Control-C can be used to safely abort conditioning early. The directory will be left in an indeterminate state, so use caution when doing this; if conditioning is stopped before completion use the `vsu` program to assess and possibly modify the current module state.

## 7.49 `mk5mon` (package : difxmessage)

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 in characters as there are Mark5 units that may transmit information. To quit, use ctrl-C. 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

## 7.50 mpifxcorr

The core of the DiFX software correlator is the program called `mpifxcorr`. This program uses Message Passing Interface (MPI) to exploit 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 (§12.2.3) and within the 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 mpifxcorr inputFile [ options ]`

`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 ^udapl,openib --mca mpi_yield_when_idle 1` to suppress some warning messages and be less aggressive on networking

`options` are additional options that `mpifxcorr` can take which include:

`-MmonHostname:[monSkip]` : hostname of a machine serving as a monitor data server, with optional value indicating how many records to skip between sends.

`-rstartSec` : start `startSec` seconds into the job, writing a new set of files into the visibility directory (`.difx/`)

Within the 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 in conjunction with `mk5daemon`.

## 7.51 oms2v2d (package : vex2difx)

The VLBI scheduling program `sched` generates a file with extension `.oms` which is used to populate some fields in the VLBA database. These fields are usually used to feed the dynamic scheduler but can also be used to reduce the tedium of transferring information from the `sched` input file (`.key`) to the `vex2difx` input file `.v2d`. For simple experiments this resulting `.v2d` file can be used unedited, but for more typical experiments editing will be required. The resulting file will have the same file prefix as the input file and will end with `.v2d`.

Usage: `oms2v2d [ options ] file.oms`

`file.oms` is an `.oms` file written by `sched`

`options` can be:

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

`-f` or `--force` : allow overwrite of output file

Example: `oms2v2d bx123.oms`

Note: `sched` now produces a `.tv2d` file (template `vex2difx`) that can contain useful information for some projects (especially multi-phase-center projects), however, this file is not tied to any particular version of DiFX (or more importantly, version of `vex2difx`) and thus cannot be guaranteed to be legal. It is thus suggested to use `oms2v2d` and transfer over needed information by hand after the fact.

### 7.52 padVDIF (package : vdifo)

Program `padVDIF` takes an input VDIF file and inserts additional packets as needed to create a contiguous without gaps. Newly inserted frames will have the invalid bit set.

Usage: `padVDIF infile outfile Mbps [ newStartMJD ]`

*infile* is the input VDIF file

*outfile* is the output VDIF file

*Mbps* is the data rate in megabits/second

*newStartMJD* is the MJD (with fractional component) to overwrite the times with

Example: `padVDIF raw.vdif smooth.vdif 2048`

### 7.53 plotapd (package : SniffPlots)

Program `plotapd` takes a text file containing *sniffer* fringe fit results and makes plot files. Separate plots for Amplitude, Phase and Delay (hence the name suffix “apd” are made for each baseline in the resultant file. A plot of delay rate is also produced.

This is an interactive command line program; running `plotapd` will prompt the user for inputs. The program `difxsniff` will run `plotapd` and its sister programs automatically, so usually it won’t be necessary to run by hand.

The `PGPLOT_FONT` environment variable must be set, otherwise all plot text will be missing.

### 7.54 plotbp (package : SniffPlots)

Program `plotbp` takes a text file containing *sniffer* bandpass output files and creates plots. This program can produce both auto- and cross-correlation data plots.

This is an interactive command line program; running `plotbp` will prompt the user for inputs. The program `difxsniff` will run `plotbp` and its sister programs automatically, so usually it won’t be necessary to run by hand.

The `PGPLOT_FONT` environment variable must be set, otherwise all plot text will be missing.

### 7.55 plotwt (package : SniffPlots)

Program `plotwt` takes a text file containing *sniffer* data weights. In this context, a data weight of 0 indicates complete loss of data and a weight of 1 indicates complete data. Each plotted data point will usually span many correlator integration periods. A solid dot will be plotted for the mean of these points, and “error bars” will indicate the range of weights over the averaging period. Note that in some cases where awkward integration times are used it may be possible for the weight to occasionally exceed 1, as long as the long running average never does.

This is an interactive command line program; running `plotwt` will prompt the user for inputs. The program `difxsniff` will run `plotwt` and its sister programs automatically, so usually it won’t be necessary to run by hand.

The `PGPLOT_FONT` environment variable must be set, otherwise all plot text will be missing.

### 7.56 printDiFX.py (package : vis2screen)

Program `printDiFX` prints a summary of the visibility information in a DiFX output file. It loops through all the records printing some basic info about frequencies, baselines, polarizations, times etc., plus a couple of selected visibility values from the start and middle of the record, to the screen.



Usage: `printDiFX difxfile inputfile`

*difxfile* is the full name of the visibility file in the `.difx` directory

*inputfile* is the path to the input file used to generate this difx output

Example: `printDiFX example_1.difx/DIFX_55523_025239.s0000.b0000 example_1.input`

### 7.57 `printVDIF` (package : vdfio)

Program `printVDIF` loops through a VDIF file inspecting each packet header and printing some basic summary info (time etc.) to the screen.

Usage: `printVDIF vdiffile Mbps`

*vdiffile* is the recorded VDIF file to inspect

*Mbps* is the data rate in megabits/second

Example: `printVDIF example.vdif 256`

### 7.58 `recover` (package : mk5daemon)

`recover` is a program that wraps the `XLRRecover` call for convenient use. This replaces the functionality of the `recover=` command of the `Mark5A` program.

Usage: `recover [ options ] type bank`

*type* is the type of recovery to attempt. See below.

*bank* should be either A or B and is the bank containing the module to address.

*options* can be:

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

`-f` or `--force` : allow overwrite of output file

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

Example: `recover -v 2 A`

There are three possible modes of operation that are selected with the *type* argument:

- 0 Fix directory if power failed during recording
- 1 Allow access to data that might have been overwritten
- 2 Unerase the module

These recovery attempts will not always be successful.

## 7.59 **startdifx** (package : calcif2)

Starting `mpifxcorr` generally requires a lengthy command. This inspired 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 `calcif2` and `genmachines`) and optionally run `difx2fits` to create a `.FITS` file for each job. This program is meant to work within the DiFX environment and would probably require modification to be useful in other situations.

Usage: `startdifx` [ *options* ] [ *startDelay* ] *input1* [ *input2* ... ]

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : be more verbose
- q or --quiet : be quieter
- 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 exists
- 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
- m or --message : start `mpifxcorr` by sending `difxStart` message to `mk5daemon`
- F or --fits : run `difx2fits` on output of each job separately
- l or --localhead : use the current host as the headnode, not `$DIFX_JEAD_NODE`
- override-version : ignore potential difx version conflicts

*startDelay* is an optional number of seconds to jump into the job upon start

*inputN* 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`

If the option to send a message to `mk5daemon` to start the correlation is used, the default multicast group and port will be `224.2.2.1` and `50200` respectively. These can be overridden with environment variables `DIFX_MESSAGE_GROUP` and `DIFX_MESSAGE_PORT`.

## 7.60 **statemon** (package : difxmessage)

Program `statemon` listens for multicast messages of the `difxStatus` variety and simply prints their contents to the terminal. This is mainly useful for debugging `mpifxcorr` and any programs responsible for launching it.

Usage: `statemon` [ *options* ]

*options* can be:

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

### 7.61 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 (usually this is `swc000` for the VLBA). 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 `--quiet` : don't produce much output

### 7.62 stripVDIF (package : vdifio)

Program `stripVDIF` strips network headers from a VDIF format baseband data file (e.g., captured from Wireshark) and dumps a pure VDIF stream.

Usage: `stripVDIF infile outfile [ skipbytesfront [ skipbytesback [ skipbytesinitial ]]`

*infile* is the input VDIF file

*outfile* is the output VDIF file

*skipbytesfront* is the number of bytes to skip over before each frame (default is 54)

*skipbytesback* is the number of bytes to skip over after each frame (default is 4)

*skipbytesinitial* is the number of bytes to skip over only once after opening the file (default is 28)

Example: `stripVDIF vdif.wireshark vdif.pure 54 4 28`

### 7.63 testdifxinput (package : difxio)

This program was intended mainly for helping debug parsing of `.input` files and associated other files. It turned out to be useful as a general tool to investigate the contents of such files. When multiple input files are provided on the command line merging of the resultant data structures is attempted. Two output files are created when run: `input.out` and `calc.out`. These files should closely resemble the input files if the parsing was done properly.

Usage: `testdifxinput [ options ] inputFilePrefix1 [ inputFilePrefix2 ...]`

*options* can be:

- `-v` or `--verbose` : be a bit more verbose
- `-h` or `--help` : print help information and quit

*inputFilePrefixn* is the base name of an input file

## 7.64 `testdifxmessagereceive`(package : difxmessage)

Test program `testdifxmessagereceive` captures multicast DiFX messages and prints them to the screen. Both the raw XML is shown and the decoded values. It is mostly useful as a tool for examining the correctness of the multicast messages that are broadcast and is not intended to be part of an operational system. There is a special binary mode which instead listens for the multicast high time resolution autocorrelations. In this mode, only a terse summary of what is received is printed (see the source code for more information).

Usage: `testdifxmessagereceive` [ *options* ] [ *type* ]

*options* can be:

- h or --help : print help information
- b or --binary : generate output based on binary records

*type* is the kind of message to capture (not for use with binary records):

1. DifxLoadMessage
2. DifxAlertMessage
3. Mark5StatusMessage
4. DifxStatusMessage
5. DifxInfoMessage
6. DifxDatastreamMessage
7. DifxCommand
8. DifxParameter
9. DifxStart
10. DifxStop
11. Mark5VersionMessage
12. Mark5ConditionMessage
13. DifxTransientMessage

If *type* is not provided, all message types will be captured.

## 7.65 `testmod` (package : mk5daemon)

`testmod` is a program that is used to perform read and write tests on Mark5 modules. It is meant as a replacement of the `ResetModule` program that relies on the `Mark5A` program which is being phased out of VLBA operations. Read-only tests can be performed without risk of erasing astronomical data recorded on the disks. The more invasive write-read tests (which are the default) will erase all preexisting data! A matrix of numbers similar to what is produced by `ResetModule` or `SSerase` in condition mode, but with statistics from a much smaller volume of reading/writing is produced. Usually badly performing disks will occur in pairs with both bad disks belonging to the same bus (e.g., disks 0 and 1, 2 and 3, 4 and 5, or 6 and 7). Badly performing drives should have their directory files `.dir` (see § 8.13) updated by hand to include RT at the end of the top line.

Usage: `testmod` [ *options* ] *bank*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : produce more informative/diagnostic output; -v -v for even more

- f or --force : continue to produce files despite warnings
- r or --readonly : Perform read-only test
- R or --realtime : Switches to real-time mode (see below)
- d or --skipdircheck : Disable directory checking (see below)
- S or --speed : Disable correctness testing to better test throughput
- n *n* or --nrep *n* : Repeat the test *n* times (default is 2)
- s *s* or --blocksize *s* : Read and write *bytes* at a time (default is 10 MB)
- b *b* or --nblock *b* : Perform *b* reads per test (default is 50)
- p *p* or --pointer *p* : Start read-only tests at byte position *p*
- o *file* or --dirfile *file* : Write the module directory to file *file*

*bank* is the Mark5 bank containing the disk to be studied (A or B)

Many modules being tested are perhaps damaged in some way and may require the -R and/or -d options above. It is generally safe to use these options, but the diagnostic power of this program may be reduced in cases where some drives are intrinsically slow, but still produce valid data.

### 7.66 testseqnumbers (package : difxmessage)

Program `testseqnumbers` is a utility to listen for DiFX multicast messages and identify any that come with a sequence number that is not sequential. This is a good way to identify possible packet loss or duplication on a DiFX cluster network.

Usage: `testseqnumbers [ options ]`

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : produce more output; -v -v for even more

If run without the -v option, only unexpected packets will be noted. If run with one -v flag, each received packet will be identified with a period being written to the screen. If run with 2 -v flags, each packet received will have its source and sequence number printed.

### 7.67 vdifbstate (package : vdifo)

Program `vdifbstate` will perform state counts on a multi-thread VDIF baseband data file.

Usage: `vdifbstate file frameSize dataRate threadlist nFrames [ offset ]`

*file* is the file to decode

*frameSize* is the size of the VDIF frames including frame headers (5032 for VLBA or VLA VDIF data)

*dataRate* is the file data rate, measured in Mbps, not including frame headers

*threadList* is a comma-separated list of thread ids to decode

*nFrames* is the number of data frames (typically a few kB in size) to decode

*offset* is the number of bytes into the file to start decoding

Example: `vdifbstate sample.vdif 5032 1024 1,2,3,4 100`

Notes:

1. See documentation for `m5b` for details on specifying the data format.
2. Only real-sampled data with 1 or 2 bits per sample is supported at this time.
3. If a non-power-of-two number of threads is requested, extra channels will be invented to pad out the next power of two. Data in these extra channels has undefined qualities.

### 7.68 `vdifd` (package : `vdifio`)

Program `vdifd` takes a multi-thread VDIF file and decodes some samples. It is implemented as a python script that makes use of `vmux` and `m5d` to do most of the work. It is thus a good program to study to understand how `vmux` can be used. This program takes as command line input the name of a file containing (or thought to be containing) multi-thread VDIF baseband data, information about the VDIF stream, and the number of samples per baseband to decode. Optionally a starting file offset can be supplied. If the data can be decoded correctly, information about the data will be printed to the screen along with a table of decoded data. The output values, -3, -1, 1, or 3 for valid data, are printed in *nchan* columns. Data that cannot be decoded (either due to data replacement headers, data fill pattern replacing the actual data after unloading from a Mark5 module, or identified via the VDIF invalid bit) will show as 0. It should be invoked with the following parameters:

Usage: `vdifd file frameSize dataRate threadlist n [ offset ]`

*file* is the file to decode

*frameSize* is the size of the VDIF frames including frame headers (5032 for VLBA or VLA VDIF data)

*dataRate* is the file data rate, measured in Mbps, not including frame headers

*threadList* is a comma-separated list of thread ids to decode

*n* is the number of samples to decode

*offset* is the number of bytes into the file to start decoding (default is 0)

Example: `vdifd sample.vdif 5032 1024 1,2,3,4 24`

In the above example, a stream consisting of 4 channels, each with 64 MHz bandwidth and 2 bits per sample and with thread ids 1, 2, 3 and 4 are to be decoded. If the thread ids were to be permuted, the decoded output data would be permuted in the same manner.

Notes:

1. At this time only 2-bit real-valued data can be properly decoded.
2. If a non-power-of-two number of threads is requested, extra channels will be invented to pad out the next power of two. Data in these extra channels has undefined qualities.

### 7.69 `vdiffold` (package : `vdifio`)

Program `vdiffold` takes a baseband data stream and integrates the power (formed by squaring the voltage) in a number of time bins that equally divide a given period. This is a simplified version of “folding” such as is used in pulsar processing. A typical use of such functionality would be to investigate the waveform of the switched power injected into the receiver for calibration. This program has found considerable utility in determining time offsets between the sample clock and formatter time (modulo the period of the calibration cycle).

In the case of 2-bit sampling a non-linear correction is applied before results are written to a file. This correction takes the form

$$P = \frac{1}{\left(\operatorname{erf}^{-1}\left(\frac{\hat{P}-v_{\text{high}}^2}{1-v_{\text{high}}^2}\right)\right)^2}, \quad (3)$$

where  $P$  is a value proportional to true power and  $\hat{P}$  is the value obtained by calculating  $\langle \hat{v}^2 \rangle$  when the bitstream is reproduced with values  $\hat{v} \in (-v_{\text{high}}, -1, 1, v_{\text{high}})$ . This non-linear correction can be turned off by setting  $nbin$  to a negative value. Note that this program is not useful for 1-bit quantized data.

The program should be used as follows:

Usage: `vdifold infile frameSize dataRate threadlist nbin nchunk freq outfile [ offset ]`

*infile* is the file to decode

*frameSize* is the size of the VDIF frames including frame headers (5032 for VLBA or VLA VDIF data)

*dataRate* is the file data rate, measured in Mbps, not including frame headers

*threadList* is a comma-separated list of thread ids to decode

*nbin* is the number of bins to calculate per period; if negative, power correction is not performed and the absolute value of *nbin* is used

*nchunk* is the number of 10000 sample chunks to operate on

*freq* is the reciprocal of the period to be observed (Hz)

*outfile* is the name of the output file

*offset* (optional) is the number of bytes into the file to start decoding

Example: `vdifold sample.vdif 5032 1024 1,2,3,4 128 10000 80 sample.fold`

The output file will contain  $nchan+1$  columns where  $nchan$  is the number of baseband channels in the data stream. The first column contains the time (seconds) within the period. Each remaining column is folded power for one baseband channel. If  $nbin$  is positive and the data is 2-bit quantized, the scaling is such that  $\langle v^2 \rangle = \sigma^2$  yields a power reading of 1.0, for sampler threshold  $\sigma$ . Optimal signal to noise ratio occurs for a value of about 1.03. For non 2-bit quantization, the power will be in units of reconstituted counts<sup>2</sup>.

Notes:

1. At this time only 2-bit real-valued data can be properly decoded.
2. If a non-power-of-two number of threads is requested, extra channels will be invented to pad out the next power of two. Data in these extra channels has undefined qualities.
3. The output columns are in the same order as the thread id list. Thus, you can rearrange the output order by changing the order of the thread list. This enables reordering of data so that polarization pairs occur consecutively, allowing more sensible cross-correlation columns.

## 7.70 `vdifspec` (package : vdifio)

Program `vdifspec` forms total power spectra for each baseband channel in the data, including cross spectra for polarization pairs, assuming data is in alternating polarization pairs (if not, the cross spectra should make no sense, but they are formed anyway). The results are written to a text file with the following columns: Column 1 is the frequency offset from baseband for each channel; Columns 2 to  $nchan+1$  are the total power spectra for each baseband channel; Columns  $nchan+2$  to  $4 \times nchan+1$  contain, in pairs, the amplitude and phase of the cross spectra for each pair of channels. It should be invoked with the following parameters:

Usage: `vdifspec infile frameSize dataRate threadlist npoint n outfile [ offset ]`

*infile* is the file to decode

*frameSize* is the size of the VDIF frames including frame headers (5032 for VLBA or VLA VDIF data)

*dataRate* is the file data rate, measured in Mbps, not including frame headers

*threadList* is a comma-separated list of thread ids to decode

*npoint* is the number of points to calculate for each spectrum

*n* is the number of FFT frames to include in the calculation

*outfile* is the name of the output file

*offset* (optional) is the number of bytes into the file to start decoding

Example: `vdifspec sample.vdif 5032 1024 1,2,3,4 256 1000 vlba.spec`

Notes:

1. At this time only 2-bit real-valued data can be properly decoded.
2. If a non-power-of-two number of threads is requested, extra channels will be invented to pad out the next power of two. Data in these extra channels has undefined qualities.
3. The output columns are in the same order as the thread id list. Thus, you can rearrange the output order by changing the order of the thread list. This enables reordering of data so that polarization pairs occur consecutively, allowing more sensible cross-correlation columns.

## 7.71 `vex2difx`

`vex2difx` is a program that takes a `.vex` files (such as one produced by `sched` with various tables based on observe-time data appended, probably by `db2vex` in the case of VLBA operations) and a `.v2d` configuration file (see §8.36) and generates one or more `.input` and `.calc` file pairs for use with the DiFX correlator. Note specifically that `.ovex` files, as used at many/most Mark4 correlators, are not supported. `vex2difx`, along with `calcif2`, supercedes the functionality of `vex2config` and `vex2model`, two programs that were widely used but never fully integrated into the VLBA's software chain. Don't forget that `oms2v2d` can be used to create a valid baseline `.v2d` file from the `.oms` file made by `sched`, perhaps saving some time.

The following guiding principles drove the design of `vex2difx`:

1. The output files should never need to be hand edited
2. Simple experiments should not require complicated configuration
3. All features implemented by `mpifxcorr` should be accessible
4. All experiments expressible by `vex` should be supported
5. The configuration file should be human and machine friendly
6. Command line arguments should not influence the processing of the `vex` file

Note that not all of these ideals have been completely reached as of now. It is not the intention of the developer to guess all possible future needs of this program. Most new features will be easy to implement so send a message to the `difx-users` mailing list or file a JIRA [7] bug tracking ticket for requests.

Usage: `vex2difx [ options ] v2dFile`



*options* can be:

- h or --help : print usage information and exit
- o or --output : write a configuration file called *v2dFile.params* (see §8.28) as output
- v or --verbose : produce more informative/diagnostic output; -v -v for even more
- d or --delete-old : delete old output from same .v2d file
- f or --force : continue to produce files despite warnings
- s or --strict : treat some warnings as errors and quit (default)

*v2dFile* is a .v2d file (see §8.36) that controls the operation of this program; the filename cannot contain underscore characters

Example: `vex2difx bx123.v2d`

### 7.71.1 VDIF issues

### 7.71.2 Mark5B issues

The Mark5B format, including its 2048 Mbps extension, is now supported by `vex2difx`. The .vex file track assignments for Mark5B format has never been formally documented. `vex2difx` has adopted the track assignment convention used by Haystack. Formally speaking, Mark5B has no tracks. Instead it stores up to 32 bitstreams in 32 bit words. The concept of fanout is no longer used with Mark5B. Instead, the equivalent operation of spreading one bitstream among 1 or more bits in each 32 bit word is performed automatically. Thus to specify a Mark5B mode, only three numbers are needed: Total data bit rate (excluding frame headers), number of channels, and number of bits per sample (1 or 2). The number of bitstreams is the product of channels and bits.

The \$TRACKS section of the vex file is used to convey the bitstream assignments. Individually, the sign and magnitude bits for each channel are specified with `fanout_def` statements. In unfortunate correspondence with existing practice, 2 is the first numbered bitstream and 33 is the highest. In 2-bit mode, all sign bits must be assigned to even numbered bitstreams and the corresponding magnitude bit must be assigned to the next highest bitstream. To indicate that the data is in Mark5B format, one must either ensure that a statement of the form

```
track_frame_format = MARK5B;
```

must be present in the appropriate \$TRACKS section or

```
format = MARK5B
```

must be present in each appropriate ANTENNA section of the .v2d file. As a concrete example, a \$TRACKS section may resemble:

```
$TRACKS;
def Mk34112-XX01_full;
  fanout_def = A : &Ch01 : sign : 1 : 02;
  fanout_def = A : &Ch01 : mag  : 1 : 03;
  fanout_def = A : &Ch02 : sign : 1 : 04;
  fanout_def = A : &Ch02 : mag  : 1 : 05;
  fanout_def = A : &Ch03 : sign : 1 : 06;
  .
  .
  .
  fanout_def = A : &Ch15 : mag  : 1 : 31;
  fanout_def = A : &Ch16 : sign : 1 : 32;
  fanout_def = A : &Ch16 : mag  : 1 : 33;
  track_frame_format = MARK5B;
```

```
enddef;
```

### 7.71.3 Media specification

`vex2difx` allows `.input` file generation for two types of media. A single `.input` file can have different media types for different stations. Ensuring that media has been specified is important as antennas with no media will be dropped from correlation. The default media choice is Mark5 modules. The `TAPELOG_OBS` table in the input `vex` file should list the time ranges valid for each module. Jobs will be split at Mark5 module boundaries; that is, a single job can only support a single Mark5 unit per station. All stations using Mark5 modules will have `DATA SOURCE` set to `MODULE` in `.input` files. If file-based correlation is to be performed, the `TAPELOG_OBS` table is not needed and the burden of specifying media is moved to the `.v2d` file. The files to correlate are specified separately for each antenna in an `ANTENNA` block. Note when specifying filenames, it is up to the user to ensure that full and proper paths to each file are provided and that the computer running the datastream for each antenna can see that file. Two keywords are used to specify data files. They are not mutually exclusive but it is not recommended to use both for the same antenna. The first is `file`. The value assigned to `file` is one or more (comma separated) file names. It is okay to have multiple file keywords per antenna; all files supplied will be stored in the same order internally. The second keyword is `filelist` which takes a single argument, which is a file containing the list of files to read. The file pointed to by `filelist` only needs to be visible to `vex2difx`, not the software correlator nodes. This file contains a list of file names and optionally start and stop MJD times. Comments can be started with a `#` and are ended by the end-of-line character. Like for the `file` keyword, the file names listed must be in time order, even if start and stop MJD values are supplied. An example file as supplied to `filelist` is below:

```
# This is a comment. File list for MK for project BX123
/data/mk/bx123.001.m5a 54322.452112 54322.511304
/data/mk/bx123.002.m5a 54322.512012 54322.514121 # a short scan
/data/mk/bx123.003.m5a 54322.766323 54322.812311
```

If times for a file are supplied, the file will be included in the `.input` file `DATA TABLE` only if the file time range overlaps with the `.input` file time range. If not supplied, the file will be included regardless of the `.input` file time range, which could incur a large performance problem.

A few sample `ANTENNA` blocks are shown below:

```
ANTENNA MK
{
  filelist=bx123.filelist.mk
}

ANTENNA OV { file=/data/ov/bx123.001.m5a,
             /data/ov/bx123.002.m5a,
             /data/ov/bx123.003.m5a }

ANTENNA PT { file=/data/pt/bx123.003.m5a } # recording started late here
```

### 7.71.4 Pulsars

Some information, including example `.v2d` sections, on setting up pulsar correlation can be found in §5.

You may find additional information at <http://cira.ivec.org/dokuwiki/doku.php/difx/vex2difx>.

### 7.72 **vexpeek** (package : vex2difx)

Program **vexpeek** takes a vex file as input and sends to *stdout* the experiment name, segment, and a list of antennas and the MJD times that they were included in the observation. This program is mainly intended to be called from python program **db2vex** which needs to know a little about the file before appending the **CLOCK** and **TAPELOG\_OBS** tables. The VLBA operations system relies on such functionality but there is no reason other operations couldn't use this. This program uses the same parsing infrastructure as **vex2difx** so the warnings that may be produced and sent to **stderr** in running **vex2difx** will also do so with **vexpeek**. Thus, when **db2vex** is run some of these error messages may be seen.

Usage: **vexpeek** [*options*] *vexFile*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : print decoded version of *vexFile* to screen
- f or --format : add per-antenna format description to output
- b or --bands : print list of bands used by *vexFile*
- u or --diskusage : add per-antenna disk usage to output

*vexFile* is the vex format file to be inspected.

### 7.73 **vlog** (package : calcif2)

Program **vlog** takes as input a calibration file (**cal.vlba**, §8.6). 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** (§8.18, §8.29, §8.33 and §8.34). 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.

### 7.74 **vmux** (package : vdifio)

Program **vmux** takes a VDIF file with multiple threads of one channel each and multiplexes the data into a new VDIF file consisting of a single thread containing all the channels. In the case that a non-power-of-two number of channels are contained in the input file (or equivalently, a non-power-of-two number of threads are specified on the command line), the next power of two will be selected as the number of channels in the output thread and any unused channel slots will contain random data.

Usage: **vmux** *inputFile* *inputFrameSize* *framesPerSecond* *threadList* *outputFile* [*offset* [*chunkSize*]]

*inputFile* is the input multi-thread VDIF file, or - for *stdin*

*inputFrameSize* is the size of one thread's data frame, including header (for RDBE VDIF data this is 5032)

*framesPerSecond* is the number of frames per second in the input file for each thread (and is thus the number of output frames per second as well)

*threadList* is a comma-separated list of integers in range 0 to 1023; the order of the numbers is significant and dictates the order of channels in the output data

*outputFile* is the name of the output, single-thread VDIF file, or - for *stdout*

*offset* is an optional offset into the input file (in bytes)

*chunkSize* is (roughly) how many bytes to operate on at a time (default is 2000000)

### 7.75 **vsn** (package : mk5daemon)

Program **vsn** is used to check or set the Volume Serial Number (VSN), the write protect state, and the Disk Module State (DMS) of a module. This program can not be used when the Mark5 unit is being used for something else. You must be logged into the Mark5 unit that contains the module to inspect/change. In addition to displaying the VSN of the module, this utility will list information about each disk in the module. The columns displayed are:

1. *Disk number*: in the range 0 to 7.
2. *Drive model*: the model number of the disk.
3. *Serial number*: the serial number of the disk (in parentheses).
4. *Drive model revision number*: addition model information.
5. *SMART capable*: 1 indicates SMART information is available; 0 otherwise.
6. *SMART state*: (only valid if SMART capable) 1 indicates good health.

Note that the drive model, serial number and revision number can have spaces making it hard to tell when one field start and the next begins. Thus the serial number is contained within parentheses to make this clear.

Usage: **vsn** [ *options* ] *bank* [ *newVSN* ]

*options* can be:

- h or --help : print usage information and exit
- f or --force : proceed without asking
- v or --verbose : be more verbose in operation
- p or --played : set DMS to played
- r or --recorded : set DMS to recorded
- e or --erased : set DMS to erased
- w or --writeprotect : set write protection
- u or --unwriteprotect : clear write protection
- s or --smart : Get S.M.A.R.T. data from disks and write to file *VSN.smart*)

*bank* is the Mark5 unit bank to look at (must be A or B)

*newVSN* is the new name to assign to the module and must be a legal VSN

Example 1: show VSN: **vsn A**

Example 2: set VSN: **vsn A NRA0+456**

If you get a message such as “Watchdog caught a hang-up executing . . .” that means access to the module failed. This could indicate a bad module. The module should be reinserted (perhaps in a different bank or unit) and the unit rebooted before coming to a firm conclusion.

### 7.76 **vsum** (package : vdifio)

Program **vsum** prints a summary of one or more VDIF files, printing such information as list of threads found, collectively, in the first and last few MB of the file, the VDIF epoch and other time information, and packet size. If the data is not recognized as VDIF, an error code will be printed; see below for the list of codes and their meanings. Legacy format VDIF data is not supported.

Usage: **vsum** *file1* [ *file2* [...]]

If the summary operation failed for a file, one of the following error codes will be returned:

- 1 The file size could not be determined
- 2 The file could not be opened
- 3 Memory allocation failure
- 4 File read failed
- 5 Frame size could not be determined
- 6 First frame could not be found
- 7 Seek to near-end-of-file failed
- 8 Read at end of file failed
- 9 A valid frame at the end of the file could not be found

### 7.77 **zerocorr** (package : mark5access)

Program **zerocorr** is intended to cross correlate data with zero time delay (and a window determined by the spectral resolution) between two recordings made at the same station. It is possible to correlate data observed in different formats and even mis-matched bandpasses, through the construction of a file describing the details of the single sub-band that is to be correlated.

Usage: **zerocorr** [ *options* ] *confFile*

*options* can be:

- h or --help : print usage information and exit
- v or --verbose : be more verbose in operation

*confFile* is a file describing the correlation parameters

Example: **zeroconf** td006.zc

See documentation on *confFile* (or .zc file) in section 8.40. Output data documentation can be found for .vis files in section 8.39 and for .lag files in section 8.23.

## 8 Description of various files

In the descriptions that follow, the locations of some files is given as `/home/vlbiobs`, meaning the directory `/home/vlbiobs/astrometry/mmmmyy/project` or one of its subdirectories (this is VLBA-centric). 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. 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`.

## 8.1 .acb

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

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

## 8.2 .apc

This file type is nearly identical to the better known .apd file; the name acronym refers to Amplitude Phase Channel. The amplitude, phase, and rate for the brightest channel is determined for each IF for each solution interval. When generation of sniffer output files is not disabled, each .FITS file written by `difx2fits` will be accompanied by a corresponding .apc file. This file contains *channel-based* 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
<i>MJD</i>	integer $\geq 1$	MJD day number corresponding to line
<i>hour</i>	$\geq 0.0, < 24.0$	hour within day
<i>source number</i>	integer $\geq 1$	source table index
<i>source name</i>	string	name of source; no spaces allowed
<i>ant1 number</i>	integer $\geq 1$	antenna table index for first antenna
<i>ant2 number</i>	integer $\geq 1$	antenna table index for second antenna
<i>ant1 name</i>	string	name of antenna 1; no spaces allowed
<i>ant2 name</i>	string	name of antenna 2; no spaces allowed
<i>nBBC</i>	integer $\geq 1$	number of baseband channels, $n_{\text{BBC}}$ The next four columns are repeated $n_{\text{BBC}}$ times
<i>channel</i>	$\geq 1, \leq n_{\text{chan}}$	the strongest channel
<i>amplitude</i>	$\geq 0.0$	the amplitude of the peak channel
<i>phase</i>	degrees	phase of the peak channel
<i>rate</i>	Hz	the channel phase rate

### 8.3 .apd

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 Amplitude, Phase, Delay (hence the name) and rate results from 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
<i>MJD</i>	integer $\geq 1$	MJD day number corresponding to line
<i>hour</i>	$\geq 0.0, < 24.0$	hour within day
<i>source number</i>	integer $\geq 1$	source table index
<i>source name</i>	string	name of source; no spaces allowed
<i>ant1 number</i>	integer $\geq 1$	antenna table index for first antenna
<i>ant2 number</i>	integer $\geq 1$	antenna table index for second antenna
<i>ant1 name</i>	string	name of antenna 1; no spaces allowed
<i>ant2 name</i>	string	name of antenna 2; no spaces allowed
<i>nBBC</i>	integer $\geq 1$	number of baseband channels, $n_{\text{BBC}}$ The next four columns are repeated $n_{\text{BBC}}$ times
<i>delay</i>	ns	the fringe fit delay
<i>amplitude</i>	$\geq 0.0$	the amplitude of fringe fit peak
<i>phase</i>	degrees	phase of fringe fit peak
<i>rate</i>	Hz	the fringe fit rate

### 8.4 .binconfig

The `.binconfig` file is a file created by the user of DiFX and referenced by the `.input` file to specify pulsar options. The file uses the standard DiFX input file format and has the following parameters:

Key	Units/allowed values	Comments
NUM POLYCO FILES	integer $\geq 1$	Number of polycy files to read ( $nPoly$ ) The next row is duplicated $nPoly$ times
POLYCO FILE $p$	string	Name of $p^{th}$ polynomial file
NUM PULSAR BINS	integer $\geq 1$	Number of pulse bins to create ( $nBin$ )
SCRUNCH OUTPUT	boolean	Sum weighted bins? If not, write all bins The next rows are duplicated $nBin$ times
BIN PHASE END $b$	float 0.0-1.0	Pulsar phase where bin ends
BIN WEIGHT $b$	float $\geq 0.0$	Weight to use when scrunching

The start of one bin is equal to the end of the previous bin; bins wrap around phase 1.0. The BIN PHASE END parameters must be listed in ascending phase order. See Sec. 5 for example usage of `.binconfig` files.

## 8.5 .bootstrap

The `difxbuild` installer program begins its process by building an environment based on the contents of a `.bootstrap` file. In the simplest case only three parameters are required (`version`, `headnode`, and `difxbase`), however installations can be customized through the use of several other parameters. The `.bootstrap` file is a text file containing zero or one `key = value` statements on each line. Comments begin with a `#`.

The parameters specified can include:

- **version:** which version of difx to install. Currently supported values are DIFX-DEVEL, DIFX-2.1, and DIFX-2.2. The `DIFX_VERSION` environment variable will reflect this value. This parameter is required.
- **headnode:** the computer that will be singled out as the head node. The `DIFX_HEAD_NODE` environment variable will reflect this value. This parameter is required.
- **difxbase:** the top level directory for DiFX software. DiFX-version-independent files will be placed directly beneath this directory. By default DiFX version specific files will be installed in a subdirectory of this (see information about the root parameter below). It is okay (and encouraged) to use the same difxbase for all installed versions as this allows common third-party software to be used. This parameter is required.
- **root:** the base directory for DiFX version/label specific files for the primary platform. Secondary platforms will use the same but with a provided extension (see `altplatformX` below). If not provided, this parameter will default to `difxbase` (or `label` if specified).
- **iproot:** path to the base of the Intel Performance Primitives library. This is IPP version dependent and may require a bit of trial and error to get right. If this is set to none then an IPP-free DiFX will be installed. This requires FFTW to be installed. Each architecture can have its own iproot value. `iproot` specifies the the default; architecture-dependent overrides are specified with a parameter such as `iprooti686` or `iprootx86_64`.
- **label:** a label used to identify an installation of DiFX. By default it is set equal to the specified version. Setting it to an alternate value allows multiple installations of the same DiFX version to be later identifiable. The `DIFX_LABEL` environment variable will reflect this value.
- **calcserver:** the computer to send RPC model requests to. If not specified, this will default to the value of the headnode parameter. The `CALC_SERVER` environment variable will reflect this value.
- **cflags:** default c and c++ compiler flags to use. If not specified, the default of `-O2 -Wall -march=core2` will be used.
- **pathextra:** extra binary search paths to add to the `PATH` environment variable that is set in the `setup_difx` script.



- **ldextra**: extra paths to be added to the LD\_LIBRARY\_PATH environment variable that is set in the setup\_difx script.
- **wrapper**: an optional wrapper program that can be used to spawn mpifxcorr. This value gets coded into the runmpifxcorr.label launcher script. For example, valgrind could be used as the wrapper program if memory leak checking is desired. Use this parameter with caution.
- **mca**: parameters to add to the /etc/openmpi-mca-params.conf file. If not provided, no such file will be created. This can be useful to include or exclude certain network interfaces. You can set this on a per platform basis. To do this, for example, set mcai686 and mcax86\_64 separately.
- **primaryarch**: Normally bootstrapping needs to be done on a machine running on the primary architecture. If primaryarch is set, the bootstrapping step can be run on any machine. This should be set to i686, x86\_64, or whatever "uname -m" returns on the primary architecture.
- **altplatformX**: Here X is a number from 1 to 9. This parameter gives a sub-label to each non-primary platform. Examples might be SDK8 and SDK9 for Mark5 units using two different Conduant library versions. For each specified alternate platform the following three additional parameters are needed ...
  - **altplatformXarch**: The CPU architecture, as determined by "uname -m", that this platform is based upon.
  - **altplatformXhost**: A representative computer making use of this platform. This is used when spawning a parallel build process.
  - **altplatformXtest**: A bash conditional expression used to determine if the computer running the setup\_bash script belongs to this platform. An example is: x'pkg-config --modversion streamstor' < "x9.0"

## 8.6 cal.vlba

Monitor data that gets attached to FITS files is extracted by tsm into a file called *projectcal.vlba* where *project* is the name of the project, i.e., *bg167* or *bc120a*. A single file contains the monitor data for all VLBA antennas, maybe also including GBT, Effelsberg and Arecibo, 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. A program called vlog (sec §7.73) reads this file and produces files called flag, pcal, tsys, and weather in the software correlator project directory.

## 8.7 .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 DiFX system, one .calc file is created by vex2difx (§7.71) for each .input file. This file is read by calcif2) (§7.1) 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 `*`. Deprecated keys that will likely be removed in an upcoming version are identified with an `×`.

Key	Units/allowed values	Comments
JOB ID	integer $\geq 1$	taken from <code>.fx</code> file
* JOB START TIME	MJD + fraction	start time of original <code>.fx</code> file
* JOB STOP TIME	MJD + fraction	end time of original <code>.fx</code> file
* DUTY CYCLE	float $\leq 1$	fraction of the job contained within scans
OBSCODE	string	observation code assigned to project
* SESSION	short string	session suffix to OBSCODE, e.g., A or BE
* DIFX VERSION	string	version of correlator, e.g. DIFX-1.5
* DIFX LABEL	string	name of correlator install, e.g. DIFX-WALTER
VEX FILE	string	dir/filename of vex file used to create the job
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
* SPECTRAL AVG	integer $\geq 1$	number of channels to average in FITS creation
* START CHANNEL	integer $\geq 0$	start channel number (before averaging)
* OUTPUT CHANNELS	integer $\geq 1$	total number of channels to write to FITS
	$> 0.0, < 1.0$	fraction of total channels to write to FITS
* TAPER FUNCTION	string	currently only UNIFORM is supported

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 $\geq 1$	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
* TELESCOPE $t$ SHELF	string	shelf location of module to correlate

The third section contains a table of sources. Sources are indexed from the following section describing the scans.

Key	Units/allowed values	Comments
NUM SOURCES	integer $\geq 1$	number of sources ( $nSource$ ) The rows below are duplicated $nSource$ times.
SOURCE $s$ NAME	string	name of source (possibly renamed from <code>.vex</code> )
SOURCE $s$ RA	radians	J2000 right ascension
SOURCE $s$ DEC	radians	J2000 declination
SOURCE $s$ CALCODE	string	usually upper case letters or blank
SOURCE $s$ QUAL	integer $\geq 0$	source qualifier

The fourth 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
NUM SCANS	integer $\geq 1$
SCAN $s$ IDENTIFIER	string
SCAN $s$ START (S)	seconds
SCAN $s$ DUR (S)	seconds
SCAN $s$ OBS MODE NAME	string
SCAN $s$ UVSHIFT INTERVAL (NS)	time to integrate before doing uv shifts (used mainly for multi-phase-center observations)
SCAN $s$ AC AVG INTERVAL (NS)	averaging interval for export of fast-dump spectra (used for VFASTR)
SCAN $s$ POINTING SRC	integer $\geq 1$
SCAN $s$ NUM PHS CTRS	integer $\geq 1$
SCAN $s$ PHS CTR $p$	integer $\geq 1$

The fifth 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 $\geq 1$	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 $e$ TAL_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 next (completely optional) section has a table for positions and velocities of spacecraft. Each spacecraft is indexed by  $s$  and each row thereof by  $r$ .

Key	Units/allowed values	Comments
* NUM SPACECRAFT	integer $\geq 0$	number of spacecraft ( $nSpacecraft$ ) Everything below is duplicated $nSpacecraft$ times.
SPACECRAFT $s$ NAME	string	name of spacecraft
SPACECRAFT $s$ ROWS	integer $\geq 1$	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.

The final section identifies the files to be produced.

Key	Units/allowed values	Comments
IM FILENAME	string	dir/filename of <code>.im</code> file to create
FLAG FILENAME	string	dir/filename of <code>.flag</code> file to create

## 8.8 `.difx/`

The SWIN format visibilities produced by `mpifxcorr` are written to a directory with extension `.difx`. Three kinds of files can be placed in this directory as described below.

Note that the formats and naming conventions of these files is not guaranteed to stay unchanged from version to version of DiFX, and hence it is not recommended to rely on these files for archival purposes.

### 8.8.1 Visibility files

The bulk of the output from `mpifxcorr` is usually in the form of a binary visibility file. Usually there will be a single visibility file in this directory, but there are three ways in which multiple files may be produced: 1. a restart of the correlation, 2. if there are multiple phase centers, and 3. if there are multiple pulsar bins.

The visibility files are systematically named in the form: `DIFX_day_sec.ssrc.bbin`, where `day` is the 5 digit integer MJD of the start of visibilities, `sec` is a zero-padded 6 digit number of seconds since the MJD midnight, `src` is a 4 digit zero-padded integer specifying the phase center number (starting at 0), and `bin` is a 4 digit zero-padded integer specifying the pulsar bin number (starting at 0).

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 binary header and is followed by binary data.

The format of the header is shown in the table below.

Key	data type	units	comments
baseline number	int		$= (a_1 + 1) * 256 + (a_2 + 1)$ for $a_1, a_2 \geq 1$
day number	int	MJD	date of visibility centroid
seconds	double	sec	vis. centroid seconds since beginning of MJD
config index	int	$\geq 0$	index to <code>.input</code> file configuration table
source index	int	$\geq 0$	index to <code>.calc</code> file scan number
freq index	int	$\geq 0$	index to <code>.input</code> frequency table
antenna 1 polarization	char	R, L, X, Y	
antenna 2 polarization	char	R, L, X, Y	
pulsar bin number	int	$\geq 0$	
visibility weight	double	$\geq 0.0$	data weight for spectrum; typically $\sim 1$
<i>u</i>	double	meter	<i>u</i> component of baseline vector
<i>v</i>	double	meter	<i>v</i> component of baseline vector
<i>w</i>	double	meter	<i>w</i> component of baseline vector

Note that for both the header and the data, the endianness is native to the machine running `mpifxcorr`, and there are currently no provisions for processing such files on a machine of different endianness.

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.

### 8.8.2 Pulse cal data files

Pulse calibration data can be extracted by `mpifxcorr`. Extraction is configured on a per-antenna basis. Data for each antenna is written to a separate file; if correlation is restarted, an additional pulse cal data file will be written.

The pulse cal data files are systematically named in the form: `PCAL_day_sec_ant`, where *day* is the 5 digit integer MJD of the start of visibilities, *sec* is a zero-padded 6 digit number of seconds since the MJD midnight, and *ant* is the 1 or 2 letter antenna name in capital letters. There is potential for these text files to have very long lines (more than 10,000 bytes) when many pulse cal tones are extracted.

The data format is exactly the same as documented in §8.29 so this will not be repeated here.

### 8.8.3 Switched power files

`mpifxcorr` can be used to extract switched power from individual antennas. Extraction is configured on a per-datastream (usually the same as per-antenna) basis. Data for each data stream is written to a separate file; if correlation is restarted, an additional set of switched power files will be started.

The switched power files are systematically named in the form: `SWITCHEDPOWER_day_sec_ds`, where *day* is the 5 digit integer MJD of the start of visibilities, *sec* is a zero-padded 6 digit number of seconds since the MJD midnight, and *ds* is the datastream index as set in the `.input` file. The test lines in these files can be long (more than 1000 bytes).

The format of these files is as follows. Each line of the file represents all measurements made on one datastream at over one integration period. The lines contain the following columns:

1. *mjdstart* : The start of the integration period in mjd+fraction
2. *mjdstop* : The end of the integration period in mjd+fraction

These are then followed by 4 numbers for each recorded channel:

1.  $P_{\text{on}}$  : power in the “on” state
2.  $\sigma_{P_{\text{on}}}$  : uncertainty of the power in the “on” state
3.  $P_{\text{off}}$  : power in the “off” state
4.  $\sigma_{P_{\text{off}}}$  : uncertainty of the power in the “off” state

The magnitudes of the numbers are meaningless but their ratios have meaning. Text after a comment character (`#`) are ignored.

## 8.9 .difxlog

The `difxlog` program (§7.17) captures `DifxAlertMessage` and `DifxStatusMessage` message types that are sent from an ongoing software correlation process and writes the information contained within to a human readable text file. One line of text is produced for each received message. The first five columns contain the date and time in `ddd MMM dd hh:mm:ss yyyy` format (e.g., `Wed Apr 22 12:48:41 2009`). The sixth column contains a word describing the contents of the remainder of the line: Options are:

`STATUS` : The status of the process is described

`WEIGHTS` : The playback weights for each antenna are listed

*other* : This word represents an alert severity level (one of `FATAL`, `SEVERE`, `ERROR`, `WARNING`, `INFO`, `VERBOSE` and `DEBUG`) and is followed by the alert message itself.

## 8.10 .speed

The program `difxspeed` (§7.20) runs a set of performance benchmarks described by a `.speed` file as documented here. This file is a text file containing various parameters. There are 5 required parameters:

Parameter	Comments
<code>datastreams</code>	comma separated list of nodes on which to run datastream processes
<code>cores</code>	comma separated list of nodes on which to run core processes
<code>antennas</code>	list of 1 or 2 letter antenna names to process
<code>nThread</code>	one or more values listing number of threads to use (see below)
<code>vex</code>	vex file to use as descriptor of observation

The value of `nThread` applies to all core processes; if multiple comma-separated values are specified, these will result in additional runs of benchmarking.

Other parameters that can be specified, either as single values or as arrays to be used in full combination with all other value arrays, include:

Parameter	Comments
<code>nAnt</code>	number of antennas (in order of listed antennas, starting with first listed)
<code>nCore</code>	number of core processes to start, using in order <code>cores</code>
<code>tInt</code>	integration time (seconds)
<code>specRes</code>	spectral resolution (MHz)
<code>fftSpecRes</code>	resolution of transform
<code>xmacLength</code>	cross-multiply stride size
<code>strideLength</code>	fringe rotation stride size
<code>numBufferedFFTs</code>	number of FFTs to process in one go
<code>toneSelection</code>	pulse cal tone selection

Notes:

1. `datastreams` and `cores` lists can repeat hostnames.
2. Some combinations of parameters is illegal; at the moment it is up to the user to ensure all combinations of values are allowed.
3. Additional parameters can be easily added to the program on demand.

## 8.11 .speed.out

The output of `difxspeed` (§7.20) is a file usually ending in `.speed.out`. The first many lines are comments describing to a human reader the fixed parameters of the benchmark trials and a table describing the meanings of the columns of the uncommented data lines that follow. In summary, the columns in the lines that follow are:

Column(s)	value(s)
1 to $N$	values of variable parameters as described by comments above
$N + 1$	The average execution time of all trials run with the combination of parameters
$N + 2$	The RMS scatter in execution time
$N + 3$ to $N + 2 + R$	List of execution times from all trials

In the above table,  $N$  is the number of parameters taking on multiple values and  $R$  is the number of times `difxspeed` was run.

## 8.12 \$DIFX\_MACHINES

This section describes the format of a file used through DiFX-2.2. For more recent versions please see documentation on the DiFX wiki <http://cira.ivec.org/dokuwiki/doku.php/difx/start/>.

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` (§7.25) 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.

## 8.13 .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, e.g., `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. At the end of this line the characters `RT` can be added (by hand) which will cause the modules to be accessed using *Real-Time* mode which is tolerant of missing or bad disks within the module. 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, 2 for Mark5B
Name	string	scan name, usually including the project code and station
Extra info	string(s)	see below

After the name of the scan additional free-form text can appear. These extra parameters can be machine parsable. The only use of this as of this writing is to indicate the thread ids for VDIF data. This will always be formatted as `Threads=` followed by a comma separated list of thread ids without any spaces. For example: `Threads=0,640,256,896`.

## 8.14 .filelist

When using the `filelist` parameter in an `ANTENNA` section of a `.v2d` (§8.36), the list of data files to correlate are stored in a text file. This is a text file containing data lines and optionally comments. Any text after the comment character (`#`) is ignored. A data line consists of a filename (must have complete path as can

be used to find the file on the datastream node for this antenna) and optionally a start time and stop time. Start and stop times can be expressed in any of the formats supported by `vex2difx`.

## 8.15 .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
SU	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
UV	The visibility data table
FG	The flag table
TS	The system temperature table
PH	The phase calibration table (pulse calcs and state counts)
WR	The weather table
GN	The gain curve table
GM	The pulsar gate model table

Not all of these tables will always be written.

## 8.16 .fitslist

A `.fitslist` file is written by `makefits` and contains the entire list of `.FITS` files for the correlator pass. Due to the different constraints of the correlation process and the FITS-IDI format, the number of resultant FITS files may be greater or less than the number of jobs. This file type is used by `difxarch` to ensure that all of the correlated output ends up in the archive. The file is composed of two parts: a header line and one line for each `.FITS` file. The header line consists of a series of `key=value` pairs. Each `key` and `value` must have no whitespace and no whitespace should separate these words from their connecting `=` sign. While any number of key-value pairs may be specified, the following ones (which are case sensitive) are expected to be present:

1. `exper` : the name of the experiment, including the segment code
2. `pass` : the name of the correlator pass
3. `jobs` : the name of the `.joblist` file used by `makefits`
4. `mjd` : the modified Julian day when `makefits` created this file
5. `DiFX` : the version name for the DiFX deployment (the value of `$DIFX_VERSION` when `vex2difx` was run)
6. `difx2fits` : the version of `difx2fits` that was run



Each additional line contains information for one `.FITS` file of the correlation pass. These lines contain three fields:

1. *archiveName* : the name of the file that will get injected into the archive (see §4.4)
2. *fileSize* : the size of the file in MB
3. *origName* : the name of the file as produced by `difx2fits` (via `makefits`)

### 8.17 `.flag`

The program `vex2difx` 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,  $n$ , which is the number of flag lines that 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.

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

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

### 8.19 `.im`

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 `*` are optional. Note that all polynomials are expanded about their `MJD`, `SEC` start time and use seconds as the unit of time.

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 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
INTERVAL (SECS)	integer	interval between new polynomial models
ABERRATION CORR	$\left\{ \begin{array}{l} \text{UNCORRECTED} \\ \text{APPROXIMATE} \\ \text{EXACT} \end{array} \right.$	level of $u, v, w$ aberration correction
NUM TELESCOPES	integer $\geq 1$	number of telescopes ( $nTelescope$ ) The row below is duplicated $nTelescope$ times
TELESCOPE $t$ NAME	string	upper case antenna name abbreviation
NUM SCANS	integer $\geq 1$	number of scans ( $nScan$ ). Everything below is duplicated $nScan$ times.
SCAN $s$ POINTING SRC	string	name of source used as pointing center
SCAN $s$ NUM PHS CTRS $\geq 1$	number of phase centers this scan ( $nPC_s$ )	Everything below is duplicated $nPC_s$ times.
SCAN $s$ PHS CTR $p$ SRC	string	name of source defining this phase center
SCAN $s$ NUM POLY	$\geq 1$	number of polynomials covering scan ( $nPoly_{s,p}$ ) Everything below is duplicated $nPoly_{s,p}$ times
SCAN $s$ POLY $p$ MJD	integer $\geq 0$	the start MJD of this polynomial
SCAN $s$ POLY $p$ SEC	integer $\geq 0$	the start sec of this polynomial
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 $a$ AZ	$order+1$ numbers	azimuth polynomial (deg)
* ANT $a$ EL GEOM	$order+1$ numbers	geometric (encoder) elevation (deg)
* ANT $a$ EL CORR	$order+1$ numbers	refraction corrected elevation (deg)
* ANT $a$ PAR ANGLE	$order+1$ numbers	parallactic angle (deg)
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 $w$

## 8.20 .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.

### 8.20.1 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
CALC FILENAME	string	name and full path to <code>.calc</code> file
CORE CONF FILENAME	string	name and full path to <code>.threads</code> file
EXECUTE TIME (SEC)	integer seconds	observe time covered by this <code>.input</code> file
START MJD	integer MJD	start date
START SECONDS	integer seconds	start time
ACTIVE DATASTREAMS	integer $\geq 2$	number of antennas ( $nAntenna$ )
ACTIVE BASELINES	integer $\geq 1$	number of baselines to correlate ( $nBaseline$ )
VIS BUFFER LENGTH	integer $\geq 1$	the number of concurrent integrations to allow
OUTPUT FORMAT	boolean	always <code>SWIN</code> here
OUTPUT FILENAME	string	name of output <code>.difx</code> directory

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

### 8.20.2 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 $\geq 1$	number of modes in file ( $nConfig$ )
CONFIG NAME	string	name of configuration
INT TIME (SEC)	seconds	integration time
SUBINT NANOSECONDS	nanosec	amount of time to process as one subintegration
GUARD NANOSECONDS	nanosec $\geq 0$	amount of extra data to send for overlap
FRINGE ROTN ORDER	int	0 is post-FFT, 1 is delay/rate, ...
ARRAY STRIDE LENGTH	int	used for optimized fringe rotation calculations
XMAC STRIDE LENGTH	int	number of channels to cross multiply in one batch (must evenly divide)
NUM BUFFERED FFTS	int	number of FFTs to cross-multiply in one batch
WRITE AUTOCORRS	boolean	enable auto-correlations; <code>TRUE</code> here
PULSAR BINNING	boolean	enable pulsar mode
PULSAR CONFIG FILE	string	( <i>only if BINNING is True</i> ) see § 8.4
PHASED ARRAY	boolean	set to <code>FALSE</code> (placeholder for now)
DATASTREAM $d$ INDEX	integer $\geq 0$	DATASTREAM table index, starting at 0
BASELINE $b$ INDEX	integer $\geq 0$	BASELINE table index, starting at 0

### 8.20.3 Rule table

The rule table describes which configuration will be applied at any given time. Usually this filters on scan attributes such as source, but can also be done in a time-based manner (start and stop times). An time for which no configuration matches will not be correlated. If more than one rule matches a given time, they must all refer to the same configuration.

This table begins with header

```
# RULES #####!
```

The table below uses  $r$  to represent the rule index, which ranges from 0 to  $nRule - 1$ .

Key	Units/allowed values	Comments
RULE $r$ CONFIG NAME	string	name to associate with this rule
★ SOURCE	string	source to match
★ SCAN ID	string	scan name to match
★ CALCODE	string	cal code to match
★ QUAL	string	source qualifier to match
★ MJD START	string	earliest time to match
★ MJD STOP	string	latest time to match

### 8.20.4 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 (§8.20.6). 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  $nFreq - 1$  and  $t$  to represent pulse cal tone index, which ranges from 0 to  $nTone_f$ .

Key	Units/allowed values	Comments
FREQ ENTRIES	integer $\geq 1$	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
NUM CHANNELS $f$	integer $\geq 1$	initial number of channels (FFT size, $nFFT$ , is twice this)
CHANS TO AVG $f$	integer $\geq 1$	average this many channels before generating output spectra)
OVERSAMPLE FAC. $f$	integer $\geq 1$	total oversampling factor of baseband data
DECIMATION FAC. $f$	integer $\geq 1$	portion of oversampling to handle by decimation
PHASE CALS $f$ OUT	integer $\geq 0$	number of phase calcs to produce ( $nTone_f$ )
PHASE CAL $f/t$ INDEX	integer	The row below is duplicated $nTone_f$ times. tone number of band

### 8.20.5 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 (§8.20.6). 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  $nAntenna - 1$  and  $c$  to represent clock coefficient, ranging from 0 to  $nCoeff_a$ .

Key	Units/allowed values	Comments
TELESCOPE ENTRIES	integer $\geq 1$	number of antennas ( $nAntenna$ )
TELESCOPE NAME $a$	string	abbreviation of antenna name
CLOCK REF MJD $a$	double	date around which the following polynomial is expanded
CLOCK POLY ORDER $a$	int $\geq 0$	polynomial order of telescope clock model ( $nCoeff_a$ )
CLOCK COEFF $a/c$	$\mu\text{sec}/\text{sec}^c$	clock model polynomial coefficient

### 8.20.6 Datastream table

The datastream table begins with header

# DATASTREAM TABLE #!

The table below uses  $f$  to represent recorded frequencies, which ranges from 0 to  $nFreq - 1$ . A second index,  $z$ , is used to iterate over zoom bands, ranging from 0 to  $nFreq - 1$ . A third 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 DiFX system, all sub-bands must have the same polarization structure, so  $nBB = nFreq \cdot nPol$ . This index is reused for the zoom bands in an analogous manner.

Key	Units/allowed values	Comments
DATASTREAM ENTRIES	integer $\geq 1$	number of antennas ( $nDatastream$ )
DATA BUFFER FACTOR	integer $\geq 1$	
NUM DATA SEGMENTS	integer $\geq 1$	
TELESCOPE INDEX	integer $\geq 0$	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 $\geq 1$	bits per sample
DATA FRAME SIZE	integer $\geq 1$	bytes in one frame(or file) of data
DATA SAMPLING	string	REAL or COMPLEX
DATA SOURCE	string	FILE (see §4.2), MODULE for Mark5 playback, or FAKE for benchmark
FILTERBANK USED	boolean	currently only FALSE
PHASE CAL INT (MHZ)	int	pulse cal comb frequency spacing, or 0 if no pulse cal tones
NUM RECORDED FREQS	integer $\geq 0$	number of different frequencies recorded for this datastream
REC FREQ INDEX $f$	integer $\geq 0$	index to frequency table
CLK OFFSET $f$ (us)	$\mu\text{sec}$	frequency-dependent clock offset
FREQ OFFSET $f$ (us)	$\mu\text{sec}$	frequency-dependent LO offset
NUM REC POLS $f$	1 or 2	for this recorded frequency, the number of polarizations
REC BAND $i$ POL	$R$ or $L$	polarization identity
REC BAND $i$ INDEX	integer $\geq 1$	index to frequency setting array above; $nBB$ per entry
NUM ZOOM FREQS	integer $\geq 0$	number of different zoom bands set for this datastream
ZOOM FREQ INDEX $z$	integer $\geq 0$	index to frequency table
NUM ZOOM POLS $z$	1 or 2	for this recorded frequency, the number of polarizations
ZOOM BAND $i$ POL	$R$ or $L$	polarization identity
ZOOM BAND $i$ INDEX	integer $\geq 1$	index to frequency setting array above; $nBB$ per entry

### 8.20.7 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 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 $\geq 1$	number of entries in table, $nBaseline$
D/STREAM A INDEX $b$	integer $\geq 0$	datastream table index of first antenna
D/STREAM B INDEX $b$	integer $\geq 0$	datastream table index of second antenna
NUM FREQS $b$	integer $\geq 1$	number of frequencies on this baseline, $nFreq_b$
POL PRODUCTS $b/f$	integer $\geq 1$	number of polarization products, $nProd_b$
D/STREAM A BAND $p$	integer $\geq 0$	index to frequency array in datastream table
D/STREAM B BAND $p$	integer $\geq 0$	same as above, but for antenna B, not A

### 8.20.8 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 $\geq 1$	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.

## 8.21 .joblist

A single `.joblist` file is written by `vex2difx` (§7.71) as it produces the DiFX `.input` (and other) files for a given correlator pass. This file contains the list of jobs to run and some versioning information that allows improved accountability of the software versions being used. This file is used by `difxqueue` and `makefits` to ensure that a complete set of jobs is accounted for. The file is composed of two parts: a header line and one line for each job. The header line consists of a series of `key=value` pairs. Each `key` and `value` must have no whitespace and no whitespace should separate these words from their connecting `=` sign. While any number of key-value pairs may be specified, the following ones (which are case sensitive) are expected to be present:

1. `exper` : the name of the experiment, including the segment code
2. `v2d` : the `vex2difx` input file used to produce the jobs of this pass
3. `pass` : the name of the correlator pass
4. `mjd` : the modified Julian day when `vex2difx` created this file
5. `DiFX` : the version name for the DiFX deployment (the value of `$DIFX_VERSION` when `vex2difx` was run)

6. `vex2difx` : the version of `vex2difx` that was run

Each additional line contains information for one job in the pass. The columns are:

1. `jobName` : the name/prefix of the job
2. `mjdStart` : the observe start time of the job
3. `mjdStop` : the observe stop time of the job
4. `nAnt` : the number of antennas in the job
5. `maxPulsarBin` : the maximum number of pulsar bins to come from any scan in this job (usually zero)
6. `nPhaseCenters` : the maximum number of phase centers to come from any scan in this job (usually one)
7. `tOps` : estimated number of trillion floating point operations required to run the job
8. `outSize` : estimated FITS file output size (MB)

Usually the comment character `#` followed by a list of station codes is appended to the end of each line.

## 8.22 `.jobmatrix`

As of version 2.6 of `difx2fits` a file with extension `.jobmatrix` will be written for each `.FITS` file created. This file is meant as a summary for human use and as such does not have a format that should be considered fixed. The file contains a 2-dimensional map (antenna number vs. time) of which jobs contributed to the `.FITS` file.

## 8.23 `.lag`

Program `zerocorr` (§7.77) produces lag output in a format documented here. There is one line of output per lag. Each line has 7 columns as per the following table:

Line	Contents
1	Channel (spectral point) number (counts from zero)
2	Time lag (sec)
3	Real value of the lag function
4	Imaginary value of the lag function
5	Amplitude
6	Phase
7	Window function (weight at this lag)

## 8.24 `.log`

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.

## 8.25 .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 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 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.

## 8.26 .mark4list

A `.mark4list` file is written by `makemark4` and contains the entire list of `gzip` compressed file sets (ending in `.mark4.tar.gz`) for the correlator pass. Due to the different constraints of the correlation process and the Mark4 format, the number of resultant compressed file sets may be greater or less than the number of jobs. This file type is used by `difxarch` to ensure that all of the correlated output ends up in the archive. The file is composed of two parts: a header line and one line for each compressed file set. The header line consists of a series of `key=value` pairs. Each `key` and `value` must have no whitespace and no whitespace should separate these words from their connecting `=` sign. While any number of key-value pairs may be specified, the following ones (which are case sensitive) are expected to be present:

1. `exper` : the name of the experiment, including the segment code
2. `pass` : the name of the correlator pass
3. `jobs` : the name of the `.joblist` file used by `makemark4`
4. `mjd` : the modified Julian day when `makemark4` created this file
5. `DiFX` : the version name for the DiFX deployment (the value of `$DIFX_VERSION` when `vex2difx` was run)
6. `difx2mark4` : the version of `difx2mark4` that was run

Each additional line contains information for one compressed file set of the correlation pass. These lines contain three fields:

1. `archiveName` : the name of the file that will get injected into the archive (see §4.4)
2. `fileSize` : the size of the file in MB
3. `origName` : the name of the file as produced by `difx2mark4` (via `makemark4`)



## 8.27 .oms

A `.oms` file is written by `sched` and contains machine (and human) readable information that is useful in setting up correlator jobs. In the case of the VLBA DiFX correlator, program `oms2v2d` (§7.51) uses this file to prepare a template `.v2d` file (§8.36) that contains some information not available in the `vex` file, such as intended integration time and number of channels.

## 8.28 .params

A file with extension `.params` is written by `vex2difx` (§7.71) when it is provided with the `-o` option. This file is a duplicate of the `.v2d` file that was supplied but with all unspecified parameters listed with the defaults that they assumed. The format is exactly the same as the `.v2d` files; see §8.36 for documentation of the format. The `.params` file can be used as a legal `.v2d` file if necessary.

## 8.29 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 9 fixed-size fields:

1. *antId* : Station name abbreviation, e.g., LA
2. *day* : Time centroid of measurement (day of year, including fractional portion)
3. *dur* : Duration of measurement (days)
4. *cableCal* : Cable calibration measurement (picoseconds)
5. *nPol* : Number of polarizations with measurements
6. *nBand* : Number of sub-bands with measurements
7. *nTone* : Number of pulse cal tones detected per band per polarization, possibly zero
8. *nState* : Number of state count states measured per band per polarization, possibly zero
9. *nRecChan* : 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  $[nPol][nBand]$ .

## 8.30 .polyco

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 [6].

### 8.31 .shelf

The `vex` file format (see §8.38 and references within) does not have a formal slot to record the shelf location of media, so `db2vex` stashes the shelf location in a separate file. This information is critical for the correlator operators to know where to find modules for a project and for analysts preparing correlator jobs to know if media have arrived. The `.shelf` file is used by `vex2difx` when writing `.calc` files. It can also be used as input to `getshelf`. The file format is very simple. One row is used for each module that was used in the observation. Typically rows are sorted in the same order as antennas in the `.input` file. The comment character is `#` – any text following this character on a line is ignored. Each line contains 3 white-space separated columns:

1. *antId* : The typically 2 letter station abbreviation
2. *vsn* : The volume serial number of the media (e.g., NRAO-123)
3. *shelf* : The shelf location, which can be any string without whitespace (e.g., BD89), or `none` if the media is not at the correlator

### 8.32 .threads

The `.threads` file tells `mpifxcorr` how many threads to start on each processing node. Within 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.

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

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

This format should not be confused with switched power files produced by `mpifxcorr` (see §8.8.3).

### 8.34 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. *antId* : Station name abbreviation, e.g., LA
2. *day* : Time of measurement (day of year, including fractional portion)
3. *T* : Ambient temperature (Centigrade)

4. *P* : Pressure (mbar)
5. *dewPoint* : Dew point (Centigrade)
6. *windSpeed* : Wind speed (m/s)
7. *windDir* : Wind direction (degrees E of N)
8. *precip* : Accumulated rain since UT midnight (cm)
9. *windGust* : Maximum wind gust over collection period (m/s)

### 8.35 .wts

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
<i>MJD</i>	integer $\geq 1$	MJD day number corresponding to line
<i>hour</i>	$\geq 0.0$ , $< 24.0$	hour within day
<i>antenna number</i>	$\geq 1$	antenna table index
<i>antenna name</i>	string	
<i>n<sub>BBC</sub></i>	$\geq 1$	Number of baseband channels
<i>mean weight</i>	$\geq 0.0$	This column repeated <i>n<sub>BBC</sub></i> times
<i>min weight</i>	$\geq 0.0$	This column repeated <i>n<sub>BBC</sub></i> times
<i>max weight</i>	$\geq 0.0$	This column repeated <i>n<sub>BBC</sub></i> times

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

### 8.36 .v2d

The `.v2d` file is used to specify correlation options to `vex2difx` and adjust the way in which it forms DiFX input files based on the `.vex` file. The `.v2d` file consists of a number of global parameters that affect the way that jobs are created and several sections that can customize correlation on a per-source, per mode, or per scan basis. All parameters (those that are global and those that reside inside sections) are specified by a parameter name, the equal sign, and one value, or a comma-separated list of values, that cannot contain whitespace. Whitespace is not required except to keep parameter names, values, and section names separate. All parameter names and values are case sensitive except for source names and antenna names. The `#` is a comment character; any text after this on a line is ignored.

Most parameters are one of the following types:

- **bool** : A boolean value that can be True or False. Any value starting with `0`, `f`, `F`, or `-` will be considered False and otherwise True.
- **float** : A floating point number. Can be of the forms: `1.23`, `1.2e-4`, `-12.6`, `4`
- **int** : An integer.
- **string** : Any sequence of printable(non-whitespace) characters. Certain fields require strings of a maximum length or certain form.

- **date** : A date field; see below.
- **array** : Array can be of any of the four above types and are indicated by enclosing brackets, e.g., [int]. The empty list is indicated with [] which is usually implied to be all-inclusive.

All times used in `vex2difx` are in Universal Time and are internally represented as a double precision value. The integer part of this value is the date corresponding to  $0^h$  UT. The fractional part, when multiplied by 86400, gives the number of seconds since  $0^h$  UT. Note that this format does not allow one to specify the actual leap second if one occurs on that day. Several date formats are supported:

- **Modified Julian Day** : A decimal MJD possibly including fractional day. E.g.: 54345.341944
- **Vex time format** : A string of the form: 2009y245d08h12m24s
- **VLBA-like format** : A string of the form: 2009SEP02-08:12:24
- **ISO 8601 format** : A string of the form: 2009-09-02T08:12:24

Global parameters can be specified one or many per line such as:

`maxGap = 2000 # seconds`

or

`mjdStart = 52342.522 mjdStop=52342.532`

The following parameter names are recognized:

Name	Type	Units	Defaults	Comments
vex	string			filename of the vex file to process; <b>this is required</b>
mjdStart	date		obs. start	discard any scans or partial scans before this time
mjdStop	date		obs. stop	discard any scans or partial scans after this time
break	date			list of MJD date/times where jobs are forced to be broken
minSubarray	int		2	don't make jobs for subarrays with fewer antennas than this
maxGap	float	sec	180	split an observation into multiple jobs if there are correlation gaps longer than this number
tweakIntTime	bool		False	adjust (up to 40%) int. time to ensure int. blocks per send
singleScan	bool		False	if True, split each scan into its own job
singleSetup	bool		True	if True, allow only one setup per job; True is required for FITS-IDI conversion
maxLength	float	sec	7200	don't allow individual jobs longer than this amount of time
minLength	float	sec	2	don't allow individual jobs shorter than this amount of time
maxSize	float	MB	2000	max FITS-IDI file size to allow
dataBufferFactor	int		32	the <code>mpifxcorr</code> DATABUFFERFACTOR parameter
nDataSegments	int		8	the <code>mpifxcorr</code> NUMDATASEGMENTS parameter
jobSeries	string			the base filename of <code>.input</code> and <code>.calc</code> files to be created; defaults to the base name of the <code>.v2d</code> file
startSeries	int		20	the default starting number for jobs created
sendLength	float	sec	0.262144	roughly the amount of data to send at a time from datastream processes to core processes
antennas	[string]		[] = all	a comma separated list of antennas to include in correlation
baselines	[string]		[] = all	a comma separated list of baselines to correlate; see below
padScans	bool		True	insert non-correlation scans in recording gaps to prevent <code>mpifxcorr</code> from complaining
invalidMask	int		0xFFFF	this bit-field selects which flag conditions are considered when writing flag file: 1=Recording, 2=On source, 4=Job time range, 8=Antenna in job
visBufferLength	int		32	number of visibility buffers to allocate in <code>mpifxcorr</code>
overSamp	int			force all basebands to use the given oversample factor
mode	string		normal	mode of operation; see below
threadsFile	string			overrides the name of the threads file to use
nCore	int			with <code>nThread</code> and <code>machines</code> , cause a <code>.threads</code> file to be made
nThreads	int			number of threads per core in <code>.threads</code> file
machines	[string]			comma separated list of machines to use as processors
				first is head node, then datastreams, then cores
maxReadSize	int	bytes	25000000	maximum number of bytes to read at a time
minReadSize	int	bytes	10000000	minimum number of bytes to read at a time

The *baselines* parameter supports the wildcard character \* an individual antenna name, or lists of antenna names separated by + on each side of a hyphen (-). Multiple baseline designators can be listed. Examples:

- A1-A2 : Only correlate one baseline
- A1-A2, A3-A4 : Correlate 2 baselines
- \*-\* : Correlate all baselines
- A1-\* or \*-A1 : Correlate all baselines to antenna A1
- A1+A2-\* : Correlate all baselines to antenna A1 or A2
- A1+A2-A3+A4+A5 : Correlate 6 baselines

A source section can be used to change the properties of an individual source, such as its position or name. In the future this is where multiple correlation centers for a given source will be specified. A source section is enclosed in a pair of curly braces after the keyword SOURCE followed by the name of a source, for example

```
SOURCE 3C273
{
  source parameters go here
}
```

or equivalently

```
SOURCE 3c273 { source parameters go here }
```

Name	Type	Units	Defaults	Comments
ra		J2000		right ascension, e.g., 12h34m12.6s or 12:34:12.6
dec		J2000		declination, e.g., 34d12'23.1" or 34:12:23.1
name	string			new name for source
calCode	char		' '	calibration code, typically A, B, C for calibrators, G for a gated pulsar, or blank for normal target
naifFile	string			name of leap seconds file (e.g., naif0010.tls for ephemeris)
ephemObject	string			name or number of object in ephemeris file
ephemFile	string			path of ephemeris file (either .bsp or .tle format)
doPointingCentre	bool		true	Whether the pointing centre should be correlated (only ever turned off for multi-phase center)
addPhaseCentre	string			contains info on a source to add; see below

To add additional phase centers, add one or more “addPhaseCentre” parameters to the source setup. In the parameter, the RA and dec must be provided. A name and/or calibrator code can be added as well. For example: `addPhaseCentre=name1010-1212/RA10:10:21.1/Dec-12:12:00.34` .

An antenna section allows properties of an individual antenna, such as position, name, or clock/LO offsets, to be adjusted. Note that the “late” convention is used in `clockOffset` and `clockRate`, unlike the “early” convention used in the `.vex` file itself (see §6.1).

Name	Type	Units	Defaults	Comments
name	string			new name to assign to this antenna
polSwap	bool		False	swap the polarizations (i.e., L $\leftrightarrow$ R) for
clockOffset	float	us	vex value	overrides the clock offset value from the
clockRate	float	us/s	vex value	overrides the clock offset rate value from
clockEpoch	date		vex value	overrides the epoch of the clock rate value
deltaClock	float	us	0.0	present if clockRate or clockOffset para adds to the clock offset (either the vex or clockOffset above
deltaClockRate	float	us/s	0.0	adds to the clock rate (either the vex or clockRate above
X	float	m	vex value	change the X coordinate of the antenna
Y	float	m	vex value	change the Y coordinate of the antenna
Z	float	m	vex value	change the Z coordinate of the antenna
format	string			force format to be one of VLBA, MKIV, one of the VDIF types
file	[string]		(none)	a comma separated list of data files to
filelist	string			a filename listing files for the DATA TA
networkPort	int			the eVLBI network port to use. This fo
windowSize	int			type in <code>.input</code> file
UDP_MTU	int			TCP window size for eVLBI. Set to < 0
vsn	string			same as setting windowSize to negative
zoom	string			override the Mark5 Module to be used
addZoomFreq	string			uses the global zoom configuration with
freqClockOffs	[float]	microsec		this antenna; <code>zoom=Zoom1</code> will match <code>Z</code> adds a zoom band with specified freq/bw <code>freq1810.0/bw4.0[/specAvg4][/noparent</code>
loOffsets	[float]	Hz		adds clock offsets to each recorded frequ <code>freqClockOffs=f1,f2,f3,f4</code> ; must be sam number of recorded freqs, first value mu
tcalFreq	int	Hz	0	adds LO offsets to each recorded frequ <code>loOffsets=f1,f2,f3,f4</code> ; must be same le number of recorded freqs.
phaseCalInt	int	MHz	1	enables switched power detection at spe zero turns off phase cal extraction, posi
toneGuard	float	MHz	0.125 of bw	the interval between tones to be extract
toneSelection	string	<b>smart</b>	tone selection algorithm; see below	when using <code>toneSelectionsmartormostde</code>
sampling	string	<b>REAL</b>	set to <b>COMPLEX</b> for complex sampled data	within this range of band edge, if possib
fake	bool		False	enable a fake data source

Possible values of “tone Selection” are:

smart	write the 2 most extreme tones at least <code>toneGuard</code> from band edge (default)
vex	write the tones listed in the vex file to FITS
none	don't write any tones to FITS
all	write all extracted tones to FITS
ends	write the 2 most extreme tones to FITS
most	write all tones not closer than <code>toneGuard</code> to band edge

Setup sections are enclosed in braces after the word `SETUP` and a name given to this setup section. The setup name is referenced by a `RULE` section (see below). A setup with the special name `default` will be

applied to any scans not otherwise assigned to setups by rule sections. If no setup sections are defined, a setup called `default`, with all default parameters, will be implicitly created and applied to all scans. The order of setup sections is immaterial.

Name	Type	Units	Defaults	Comments
tInt	float	sec	2	integration time
FFTSpecRes	float	MHz	0.125	frequency resolution of FFT
specRes	float	MHz	0.5	output freq res (must be mult. of FFTSpecRes)
nChan	int		16	number of channels per spectral window; must be $5^m \cdot 2^n$
specAvg	int		1	how many channels to average together after correlation
fringeRotOrder	int		1	the fringe rotation order: 0=post-F, 1=linear, 2=quadratic
strideLength	int		16	number of channels to “stride” for fringe rotation, etc.
xmacLength	int		128	number of channels to “stride” for cross-multiply accumulations
numBufferedFFTs	int		1	number of FFTs to do in a row for each datastream, before XMAC
doPolar	bool		True	correlate cross hands when possible
postFFFringe	bool		False	do fringe rotation after FFT?
binConfig	string		none	if specified, apply this pulsar bin config file to this setup
freqId	[int]		[] = all	frequency bands to correlate

Note that either “FFTSpecRes” and “specRes” can be used, or “nChan” and “specAvg” can be used, but the two sets cannot be mixed.

Zoom channels can be configured in a special section and referenced from ANTENNA sections to minimize complexity of the `.v2d` file. Each ZOOM section has a name and one or more “addZoomFreq” parameters, with the same format as they would have in the ANTENNA block.

Earth Orientation Parameter (EOP) data can be provided via one or more EOP sections. EOP data should be provided either in the `.v2d` file or in the vex file, but not both. Normally the vex file would be used to set EOP values, but there may be cases (eVLBI?) that want to use the vex file from `sched` without any modification. Like ANTENNA and SOURCE sections, each EOP section has a name. The name must be in a form that can be converted directly to a date (see above for legal date formats). Conventional use suggests that these dates should correspond to 0 hours UT; deviation from this practice is at the users risk. There are four parameters that should all be set within an EOP section:

Name	Type	Units	Defaults	Comments
tai_utc	float	sec		TAI minus UTC; the leap-second count
ut1_utc	float	sec		UT1 minus UTC; Earth rotation phase
xPole	float	arcsec		X component of spin axis offset
yPole	float	arcsec		Y component of spin axis offset

A rule section is used to assign a setup to a particular source name, calibration code (currently not supported), scan name, or vex mode. The order of rule sections *does* matter as the order determines the priority of the rules. The first rule that matches a scan is applied to that scan. The correlator setup used for scans that match a rule is determined by the parameter called “setup”. A special setup name `SKIP` causes matching scans not to be correlated. Any parameters not specified are interpreted as fully inclusive. Note that multiple rule sections can reference the same setup section. Multiple values may be applied to any of the parameters except for “setup”. This is accomplished by comma separation of the values in a single assignment or with repeated assignments. Thus

```

RULE rule1
{
  source = 3C84,3C273
  setup = BrightSourceSetup
}

```



is equivalent to

```
RULE rule2
{
  source = 3C84 3C273
  setup = BrightSourceSetup
}
```

is equivalent to

```
RULE rule3
{
  source = 3C84
  source = 3C273
  setup = BrightSourceSetup
}
```

The names given to rules (e.g., rule1, rule2 and rule3 above) are not used anywhere (yet) but are required to be unique.

Name	Type	Units	Comments
scan	[string]		one or more scan name, as specified in the vex file, to select with this rule
source	[string]		one or more source name, as specified in the vex file, to select with this rule
calCode	[char]		one or more calibration code to select with this rule
mode	[string]		one or more modes as defined in the vex file to select with this rule
setup	string		The name of the SETUP section to use, or SKIP if this rule describes scans not to correlate

Note that source names and calibration codes reassigned by source sections are not used. Only the names and calibration codes in the vex file are compared.

There are currently two modes of operation supported by `vex2difx`. The mode used in the vast majority of situations is called `normal` and is the default if none is specified. Currently one alternative mode, `profile`, is supported. This mode is useful for generating pulse profiles that would be useful for pulsar gating, scrunching, and binning. The difference compared to normal mode is that the standard autocorrelations are turned off and instead are computed as if they are cross correlations. This allows multiple pulsar bins to be stored. No formal cross correlations are performed. To be useful, one must create and specify a `.binconfig` file and select only the pulsar(s) from the experiment.

See <http://cira.ivec.org/dokuwiki/doku.php/difx/vex2difx> for more complete information and examples.

### 8.37 .xcb

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

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

### 8.38 .vex, .skd, .vex.obs, & .skd.obs

The `vex` (VLBI EXperiment) file [10] format is a standard observation description format used globally for scheduling observations and for driving the correlation thereof. The original `vex` file for an experiment is typically created by `sched` or `sked`. In the former case (the case used by most astronomical VLBI), the `vex` file has the unfortunate file extension `.skd`; in the later, the file extension is usually the less confusing `.vex`. These two `vex` formatted files contain only observation-scheduling based information. A small amount of information based on the actualities of the observation are added by `db2vex`, producing a new `vex` file with an additional file extension `.obs`. Please see `vex` documentation external to this manual for more information.

### 8.39 .vis

Program `zerocorr` (§7.77 produces visibility output in a format documented here. There is one line of output per generated spectral point. Each line has 8 columns as per the following table:

Line	Contents
1	Channel (spectral point) number (counts from zero)
2	Frequency relative to first spectral channel (Hz)
3	Real value of the visibility
4	Imaginary value of the visibility
5	Amplitude
6	Phase
7	Autocorrelation of the first datastream (real only)
8	Autocorrelation of the second datastream (real only)

## 8.40 .zerocorr

Program `zerocorr` (§7.77) is a simple cross correlator. It is limited to correlating one visibility spectrum from one baseband channel and can only make use of a constant offset delay model. The section documents the file that drives this program. This file consists of 17 lines of text. The first 7 lines describe properties of the data from the first antenna and the following (7) lines describe properties of the second antenna as follows:

Line	Contents
1 (8)	Input baseband data file name
2 (9)	Data format (e.g., Mark5B-2048-16-2)
3 (10)	Input sub-band to process (0-based index)
4 (11)	Offset into the file (bytes)
5 (12)	Size of FFT to perform over input bandwidth ( $2 \times n_{\text{chan}}$ )
6 (13)	First channel (spectral point) to correlate
7 (14)	Number of channels to correlate (negative indicates LSB)

The last three lines dictate the output data files and the number of FFTs to process:

Line	Contents
15	Name of output visibility ( <code>.vis</code> ; §8.39) file
16	Name of output <code>.lag</code> (§8.23) file
17	Number of FFTs to process (if -1, run on entire input files)

## 9 XML message types

The `difxmessage` library (§12.2.7) 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 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., Mark5 units 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
<b>DifxAlertMessage</b>	an error message.
<b>DifxCommand</b>	a command message.
<b>DifxDatastreamMessage</b>	<i>Not yet implemented</i>
<b>DifxDiagnosticMessage</b>	used by <b>mpifxcorr</b> to pass out diagnostic-type info such as buffer states
<b>DifxFileTransfer</b>	used by the new (USNO) GUI to cause a file to be sent to or from a specified host
<b>DifxFileOperation</b>	used by the new (USNO) GUI to cause some operation to a file (such as <b>mkdir</b> , <b>mv</b> , <b>rm</b> )
<b>DifxGetDirectory</b>	used by the new (USNO) GUI to request a Mark5 <b>.dir</b> file
<b>DifxInfoMessage</b>	<i>not used?</i>
<b>DifxLoadMessage</b>	CPU and memory usage (usually sent by <b>mk5daemon</b> ).
<b>DifxMachinesDefinition</b>	used by the new (USNO) GUI to remotely write a <b>.machines</b> file
<b>DifxParameter</b>	specify new parameter value to an <b>mpifxcorr</b> process.
<b>DifxSmartMessage</b>	contains smart data for one drive of a module; usually sent by <b>mk5daemon</b>
<b>DifxStart</b>	tell head node to start a difx job.
<b>DifxStop</b>	tell <b>mk5daemon</b> to stop a particular instance of <b>mpifxcorr</b> .
<b>DifxStatusMessage</b>	status of the <b>mpifxcorr</b> program.
<b>DifxTransientMessage</b>	used by the VFASTR project to indicate an event of interest
<b>DifxVex2DifxRun</b>	used by the new (USNO) GUI to remotely run <b>vex2difx</b>
<b>Mark5DriveStatsMessage</b>	Mark5 module conditioning statistics for one disc.
<b>Mark5StatusMessage</b>	status of the mark5 unit and modules.
<b>Mark5VersionMessage</b>	versions, board types and serial numbers of a Streamstor card.

Many of these message types are described in sections that follow. For those without documentation, the source file **difxmessage.h** in the **difxmessage** package contains the full set of parameters.

*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 [2] 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.

## 9.1 DifxAlertMessage

This section describes messages with *messageType* = `DifxAlertMessage`. 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:

```
<difxAlert>
  <alertMessage>message</alertMessage>
  <severity>severity</severity>
</difxAlert>
```

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	name	meaning
0	FATAL	processing has failed; a restart is needed
1	SEVERE	data from one or more station is affected badly
2	ERROR	data from one or more station may be affected; e.g., low weights
3	WARNING	minor error of no consequence to ongoing processing
4	INFO	informational only
5	VERBOSE	overly verbose information
6	DEBUG	debugging information

## 9.2 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 <code>SSReset</code> and <code>ssopen</code> to be run to reset Streamstor.
StartMark5A	start the Mark5A program.
StopMark5A	stop the Mark5A program.
KillMpifxcorr	kill with signal 9 (sigkill) any process with name <code>mpifxcorr</code> .
Clear	reset the <code>mk5daemon</code> ; useful sometimes if <code>mpifxcorr</code> crashes.
Reboot	causes machine to reboot.
Poweroff	causes machine to power down.
Copy <bank> <vsn> <scans>	causes scans to be copied to local disk.

### 9.3 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.

### 9.4 DifxParameter

The structure of a `DifxParameter` message *body* is as follows:

```
<difxParameter>
  <targetMipId>id</targetMipId>
  <name>name</name>
  <index1>index1</index1>
  .
  .
  .
  <indexN>indexN</indexN>
  <value>value</value>
</difxParameter>
```

Such a message is intended to allow a parameter, possibly qualified with array indices, to be set to a particular value. A possible use, for example, is to change a station clock value within a running DiFX instantiation or to enable generation of fast dump spectra for transient searching.

The italicized fields are as follows:

*targetMpiId* the MPI process Id (rank) to target with this message. Values zero and greater target specific MPI processes, with zero always being the manager process. Other special values include:

value	meaning
-1	all MPI processes
-2	all core (processing) processes
-3	all datastream processes

*name* the name of the parameter to set. For array types, the following *index* values specify the element to set.

*indexN* an integer specifying the index of the particular array axis.

*value* a string containing the value.

## 9.5 DifxSmartMessage

The Self-Monitoring, Analysis and Reporting Technology (SMART) protocol is used by hard drives for health monitoring. The Mark5 units support this and make the SMART values available via the Streamstor API. `mk5daemon` accesses this information periodically and on module insertion so that operators can be made aware of obvious and potential module problems as early as possible. The `DifxSmartMessage` multicast message is used to convey such SMART information for individual drives in a module. It should be expected that when such messages are sent a separate message will come for each drive (typically 8) of a module.

The *body* of this message type contains:

```
<difxSmart>
  <mjd>mjd</mjd>
  <vsn>vsn</vsn>
  <slot>slot</slot>
  <smart id=smartId value=value />
  .
  .
  .
</difxSmart>
```

Multiple `smart` tags (8 to 16 are usual) will usually be present in each smart message.

The italicized fields are as follows:

*mjd* The time at which the SMART value was extracted.

*vsn* The module number.

*slot* The slot of the hard drive in the module (0 to 7).

*smartId* The identifier for the value being represented. This is usually a small ( $\leq 300$ ) positive integer. More information can be found at <http://en.wikipedia.org/wiki/S.M.A.R.T.>

*value* The value of the monitor point corresponding to the *smartId*.

## 9.6 DifxTransientMessage

This section describes messages with *messageType* = `DifxTransientMessage`. This message is related to a commensal transient search project. A message of this type should be sent as soon as possible after detection; it is likely that no provisions will be made for data copying that does not start before resources assigned to the job are released. When a possible transient is identified by a detecting program which looks at autocorrelations it sends this message to all Mark5 units. Once correlation is complete, the `mk5daemon` program on appropriate Mark5 units will take a few seconds to copy data from the time range(s) of interest.

The *body* of this message type contains:

```
<difxTransient>
  <jobId>jobId</jobId>
  <startMJD>startMJD</startMJD>
  <stopMJD>stopMJD</stopMJD>
  <priority>priority</priority>
  <destDir>destDir</destDir>
  <comment>comment</comment>
</difxTransient>
```

The italicized fields are as follows:

*jobId* The job identification string. This is required so that only relevant Mark5 units take action on the received message.

*eventId* A string containing the name of the event as declared by the transient detector.

*startMJD* The start time (MJD) of the segment of data to copy.

*stopMJD* The stop time (MJD) of the segment of data to copy. Note that the total amount of data to be copied should be low enough to have an inconsequential impact on correlation throughput.

*priority* A floating point value indicating the relative importance of capturing this event. In jobs where many triggers occur this field will be used to select the most important ones to save to disk. Higher numbers indicate higher priority.

*destDir* (optional) A final directory to store the baseband data. If not provided, a default will be assumed. Note that behavior is undefined if different destination directories are provided within a single job.

*classification* (optional) A string provided by the transient detector containing a tentative classification.

*comment* (optional) A comment that could be appended to a log file.

All data and a log file will be stored in a subdirectory of the data staging area named *jobId*. The log file, at a minimum, will contain the list of events sent by the transient detection program and a log of the copying process, indicating any errors that may have occurred. The subdirectory will have a temporary name starting with a period (.) until all data copy for the job in question is complete. The use of this message type is demonstrated in Fig. 3.

## 9.7 DifxStart

This document type causes the head node to spawn a correlator job. The document contents describe which resources to use and which `.input` file to use.

The *body* of the message contains:



```

<difxStart>
  <input>input file</input>
  <force>forceOverwrite</force>
  <manager node="node" />
  <datastream nodes="nodes" />
  <process nodes="nodes" threads="count" />
  <env>envvar=value</env>
  <difxProgram>program</difxProgram>
  <difxVersion>version</difxVersion>
  <mpiWrapper>mpiWrapper</mpiWrapper>
  <mpiOptions>options</mpiOptions>
</difxStart>

```

In the above XML file, exactly one manager node must be supplied. There must be at least one datastream node (one per antenna being correlated). There must be at least one process node. Zero or more (up to a maximum of 8) environment variables may be set. The italicized fields are as follows:

*input file* complete path to the `.input` file for this correlation.

*forceOverwrite* cause any previous correlator output of this job to be deleted before starting the correlation. A value of 1 or `True` will enable overwrite.

*value* the value of the environment variable.

*node* the name of the node being assigned, e.g. `mark5fx02` or `swc000`.

*nodes* a list of node names. The list members should be space or comma separated.

*count* the maximum number of threads to schedule. If not specified, 1 will be assumed. This applies only to process nodes.

*envvar* an environment variable to set before running `mpifxcorr`.

*program* the name of the software correlator executable. This is optional and defaults to `mpifxcorr` if not set.

*version* the version (e.g., `DIFX-1.5.4`) of DiFX to run.

*mpiWrapper* the name of the program used to start the MPI processes. This field is optional and defaults to `mpirun` if none is provided.

*options* extra options to pass to `mpirun`. This is optional; sensible defaults are assumed if not explicitly set.

Note that multiple `<process />` tags can be specified, each with its own thread count. Each tag's thread count only affects those nodes specified in that tag. If *version* is provided, then a wrapper script called `runmpifxcorr.version` is expected to be in the default path which sets the environment for the version of DiF to actually run.

## 9.8 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 ant=antId wt=weight>
</difxStatus>

```

The italicized fields are as follows:

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

state	meaning
Spawning	the <code>mpifxcorr</code> processes are being started (not sent by <code>mpifxcorr</code> ).
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 <code>mpifxcorr</code> ).
Crashed	<code>mpifxcorr</code> crashed; usually sent by <code>mk5daemon</code> .

*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  $N_{\text{ant}} - 1$ .

*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.

## 9.9 DifxStop

Messages with *messageType* = `DifxStop` are typically sent by the DiFX Operator Interface to the `mk5daemon` running on the correlator head node to cause a particular instance of DiFX to be killed.

The *body* of the message contains:

```

<difxStop>
</difxStop>

```

## 9.10 Mark5DriveStatsMessage

Mark5 module conditioning is done periodically to ensure top performance of Mark5 modules. Each disk in the module gets written across its whole surface to identify bad areas and to *calibrate* the electronics. One message applies to one disk of the module

The *body* of the message contains:

```

<difxDriveStats>
  <serialNumber>serial</serialNumber>
  <modelName>model</modelName>
  <size>size</size>
  <moduleVSN>vsN</moduleVSN>
  <moduleSlot>slot</moduleSlot>
  <startMJD>startMJD</startMJD>
  <stopMJD>stopMJD</stopMJD>
  <binN>statsN</binN>

```

```

    <type>statsType</type>
    <startByte>startByte</startByte>
</difxDriveStats>

```

The italicized fields are as follows:

*serial* the serial number of the disk.

*model* the model number of the disk.

*size* the size of the disk, in GB.

*vsn* the module Volume Serial Number (VSN).

*slot* the location of the disk within the module, from 0 to 7.

*startMJD* the time when conditioning began.

*stopMJD* the time when conditioning ended.

*statsN* the histogram count for bin *N* for *N* in the range 0 to 7.

*statsType* type of operation which is one of `condition`, `condition_read`, `condition_write`, `read`, `write`, `unknown`, `test`.

*startByte* if not present, assumed to be zero; only relevant for some types of operations.

## 9.11 Mark5StatusMessage

This section describes messages with *messageType* = `Mark5StatusMessage`. This message type comes from either `mpifxcorr` or `mk5daemon` (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 hexadecimal number with the following bits: *TBD*

*activeBank* the active bank, either A or B for banks A and B respectively, N if the unit is in non-bank mode, or blank if no modules are active.

*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.
Copy	data is being copied off the module to a local disk.

*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.

## 9.12 Mark5VersionMessage

This section describes messages with *messageType* = `Mark5VersionMessage`. This message comes from `mk5daemon`. It is typically broadcast once upon the start of `mk5daemon` and when requested.

The *body* of the message contains:

```
<mark5Version>
  <ApiVer>ApiVer</ApiVer>
  <ApiDate>ApiDate</ApiDate>
  <FirmVer>FirmVer</FirmVer>
  <FirmDate>FirmDate</FirmDate>
  <MonVer>MonVer</MonVer>
  <XbarVer>XbarVer</XbarVer>
  <AtaVer>AtaVer</AtaVer>
  <UAtaVer>UAtaVer</UAtaVer>
  <DriverVer>DriverVer</DriverVer>
  <BoardType>BoardType</BoardType>
  <SerialNum>SerialNum</SerialNum>
  <DaughterBoard>
    <PCBType>PCBType</PCBType>
    <PCBSubType>PCBSubType</PCBSubType>
    <PCBVer>PCBVersion</PCBVer>
    <FPGAConfig>FPGAConfig</FPGAConfig>
```

```

        <FPGAConfigVer>FPGAConfigVersion</FPGAConfigVer>
    </DaughterBoard>
</mark5Version>

```

Note that the `DaughterBoard` tag and its subtags are optional and are not broadcast if a daughter board is not detected on the Mark5C unit. The italicized fields are as follows:

*ApiVer* The software API version of the Streamstor API.

*ApiDate* Date associated with the above.

*FirmVer* The version of the firmware that is loaded.

*FirmDate* Date associated with the above.

*MonVer* The version of the Monitor FPGA code.

*XbarVer* The version of the cross bar FPGA code

*AtaVer* The version of the ATA disk controller FPGA code.

*UAtaVer* The version of the UATA disk controller FPGA code.

*DriverVer* The version number of the driver code.

*BoardType* The type of Streamstor board.

*SerialNum* The serial number of the Streamstor board.

*PCBType* The type of Streamstor daughter board.

*PCBSubType* Subtype of the above, if any.

*PCBVersion* The version of the daughter board.

*FPGAConfig* Name of the FPGA configuration.

*FPGAConfigVersion* Version number of FPGA configuration.

## 10 Database tables

A database backend (currently Oracle) is used to store certain bits of information that are important for the operation of the VLBA-DiFX correlator. The same physical database is used to store monitor data from VLBA observations and log data from foreign stations that are processed through `fs2db`. This version of this document describes only new features, including database tables not used in the hardware correlator era and the software that populates and uses this information. The new software in question accesses the database through one of two mechanisms. Python programs use the `cx_Oracle` library and Java programs (e.g., the DiFX Operator Interface) use `JAXB`.

### 10.1 The DIFXQUEUE table

The DIFXQUEUE table is used to specify the state of the correlator queue. Each job can have a unique entry in this table. The structure of this table is based on the FXQUEUE table used by OMS, but this table is incompatible with OMS and should be treated as a completely parallel development. The program `difxqueue` will populate this table for each job being staged for correlation. Initially the STATUS field will be "QUEUED", but will change to one of the other options in the course of correlation. The DiFX Operator Interface (DOI) uses this database table directly.

Column	Type	Comments
PROPOSAL	VARCHAR2(10)	The proposal code
SEGMENT	VARCHAR2(2)	Segment (epoch) of proposal, or blank
JOB_PASS	VARCHAR2(32)	Name of correlator pass (e.g. “geodesy”)
JOB_NUMBER	INTEGER	Number of job in the pass
PRIORITY	INTEGER	Number indicating the priority of the job in the queue; 1 is high, 2 is default, and 3 is low
JOB_START	DATE	Observe time of job start
JOB_STOP	DATE	Observe time of job stop
SPEEDUP	FLOAT	Estimated speed-up factor for job
INPUT_FILE	VARCHAR2(512)	Full path of the VLBA-DiFX input file <sup>1</sup>
STATUS	VARCHAR2(32)	Status of the job, perhaps “QUEUED”, “KILLED”, “RUNNING”, “FAILED”, “UNKNOWN” or “COMPLETE”
NUM_ANT	INTEGER	Number of antennas in the job
CORR_TYPE	VARCHAR2(32)	Type of correlation (e.g., “PRODUCTION” or “CLOCK”)
CORR_VERSION	VARCHAR2(32)	The DiFX version string
NUM_FOREIGN	INTEGER	Number of non-VLBA antennas in job
OUTPUT_SIZE_EST	INTEGER	Estimated correlator output size (MB)

## 10.2 The DIFXLOG table

The DIFXLOG table contains a list of all correlation attempts and is based on the FXLOG table used by the hardware correlator and OMS. In the case of successful correlation, the CORR\_STATUS field will be set to “COMPLETE” and the SPEEDUP and OUTPUT\_SIZE fields will be set.

Column	Type	Comments
PROPOSAL	VARCHAR2(10)	The proposal code
SEGMENT	VARCHAR2(2)	Segment (epoch) of proposal, or blank
JOB_PASS	VARCHAR2(32)	Name of correlator pass (e.g. “geodesy”)
JOB_NUMBER	INTEGER	Number of job in the pass
CORR_START	DATE	Start time/date of correlation
CORR_STOP	DATE	Stop time/date of correlation
SPEEDUP	FLOAT	Measured speed-up factor
INPUT_FILE	VARCHAR2(512)	File name of .input file
OUTPUT_FILE	VARCHAR2(512)	File name of correlator output
OUTPUT_SIZE	INTEGER	Size (in 10 <sup>6</sup> bytes) of correlator output
CORR_STATUS	VARCHAR2(32)	Status of correlation, typically “COMPLETED”
CORR_TYPE	VARCHAR2(32)	Type of correlation (e.g., “PRODUCTION” or “CLOCK”)
CORR_VERSION	VARCHAR2(32)	The DiFX version string

## 10.3 The CONDITION table

The CONDITION table contains performance information for the hard disks comprising Mark5 modules. A separate table entry is made for each disk in a module so typically there will be 8 entries generated for each module conditioned. There are two paths to get data into this table. The `condition` program can be used to manually load condition reports from the SSErase program. Secondly, the `condition_watch` program automatically populates the database immediately after module conditioning upon receipt of a `Mark5ConditionMessage` that is now generated by a specially modified version of Haystack Observatory’s SSErase program.

Column	Type	Comments
SERIALNUM	VARCHAR2(32)	The hard disk serial number
MODEL	VARCHAR2(32)	Model number of hard disk
CAPACITY	INTEGER	Size of disk in 10 <sup>9</sup> bytes
MODULEVSN	VARCHAR2(10)	The name of the module containing the disk
SLOT	INTEGER	The slot number within the module (0 to 7)
STARTTIME	DATE	Date/time of conditioning start
STOPTIME	DATE	Date/time of conditioning completion
BIN0	INTEGER	Bin 0 of performance histogram (< 1.125 ms)
BIN1	INTEGER	Bin 1 of performance histogram (< 2.25 ms)
BIN2	INTEGER	Bin 2 of performance histogram (< 4.5 ms)
BIN3	INTEGER	Bin 3 of performance histogram (< 9 ms)
BIN4	INTEGER	Bin 4 of performance histogram (< 18 ms)
BIN5	INTEGER	Bin 5 of performance histogram (< 36 ms)
BIN6	INTEGER	Bin 6 of performance histogram (< 72 ms)
BIN7	INTEGER	Bin 7 of performance histogram ( $\geq$ 72 ms)

## 11 DiFX alert messages

This section attempts to list all of the messages you might see coming from the software correlator with some explanation about their meaning. For some messages, certain actions to be taken are suggested. Each subsection below contains descriptions of the messages for a particular severity level, ordered from most severe to least severe. There are almost 500 distinct messages that could be produced by `mpifxcorr` as of DiFX version 1.5.2, thus no effort has been made to be complete in the descriptions here. Effort has been made to document the most important ones in detail. Messages are sorted first by their severity level and are then roughly alphabetical within each subsection.

### 11.1 Fatal

All fatal errors will cause immediate termination of the correlation project. Most such errors would occur at the very start of a job as the input files are being processed.

- **Bin phase breakpoints are not in linear ascending order!!!:** The pulsar bins are not listed in phase order in the `.binConfig` file. Each `BIN PHASE END` entry must be greater than the previous and they must all be in the range 0 to 1.
- **Cannot create output directory *outDir*: flag - aborting!!!:** The output directory could not be created. The most common cause is an existing output file from a previous attempt to correlate the job in question. Other possible causes include: permission issues, inaccessibility of output directory to the DiFX head node, and output filesystem being full.
- **Cannot locate *stationName* in delay file *delayFile* - aborting!!!:** The specified delay file does not contain needed information for a station called *stationName*. This should never happen for data going through the correlator in a standard way.
- **Cannot open Streamstor device. Either this Mark5 unit has crashed, you do not have read/write permission to /dev/windrvr6, or some other process has full control of the Streamstor device.:** This message will only come from a Mark5 unit that is requested to play back data. The most likely cause of such a problem is the Streamstor card getting stuck in a compromised state, although fresh correlator installations that may not have left `/dev/windrvr6` with global read/write permission is a second likely cause of this problem. The fix likely requires a reboot of the Mark5 unit. On occasion a full power cycle of the Mark5 unit (not just a soft reboot) is required.

- **Cannot open file *inputFile* - aborting!!!:** The `.input` file named *inputFile* is not readable by the software correlator. This could be due to one of a number of issues, including: read permission problems, the request file not existing, or the file existing but not visible from one or more of the software correlator nodes.
- **Cannot open delay file *delayFile* - aborting!!!:** The `.delay` file named *delayFile* is not readable by the software correlator. This could be due to one of a number of issues, including: read permission problems, the request file not existing, or the file existing but not visible from one or more of the software correlator nodes.
- **Cannot open output file *outputFile* - aborting!!!:** The output file could not be created. The most common cause is an existing output file from a previous attempt to correlate the job in question. Other possible causes include: permission issues, inaccessibility of output directory to the DiFX head node, and output filesystem being full.
- **Cannot open pulsar config file *binConfigFile* - aborting!!!:** The specified pulsar bin file, *binConfigFile* cannot be opened. This could be due to one of a number of issues, including: read permission problems, the request file not existing, or the file existing but not visible from one or more of the software correlator nodes.
- **Cannot open uvw file *uvwFile* - aborting!!!:** The specified `.uvw` file, *uvwFile* cannot be opened. This could be due to one of a number of issues, including: read permission problems, the request file not existing, or the file existing but not visible from one or more of the software correlator nodes.
- **Cannot quad interpolate delays with post-f fringe rotation - aborting!!!:** Two mutually exclusive options (QUAD DELAY INTERP and POST-F FRINGE ROT) were both enabled in the `.input` file.
- **Caught an MPI exception!!! *errorString*:** A process communication error has occurred causing the correlator to terminate. The outcome of such an event cannot be good; contact a DiFX developer.
- **Config encountered inconsistent setup in config file - aborting!!!:** One or more of the configurations (group of settings defined in the `.input` file) is either illegal or incompletely defined. Usually this message will come with another more detailed error message. In any case, either there is a correlator version mismatch or there is something wrong with the `.input` file.
- **Core received a request to process data from time *time* which does not have a config - aborting!!!:** This is likely due to failed consistency check in the software correlator that resulted from a logic error in the code. This should be reported to a DiFX developer.
- **Could not find station *stationName* in the uvw file when making rpfits header!!! This station is used in the correlation so I will abort!!!:** This error only occurs with RPFITS output format which is not supported by this documentation – please seek other sources of assistance if needed.
- **Could not locate any of the specified sources in the specified time range - aborting!!!:** The correlator gave up since none of the sources to be correlated appeared in the `.uvw` file. This should never happen for data going through the correlator in a standard way.
- **Could not locate a polyco to cover the timerange ...:** A pulsar polynomial for a certain time period could not be found. The person supplying the pulsar polynomial should be contacted.
- **DataStream assumes long long is 8 bytes, it is x bytes - aborting!!!:** This should not occur on any modern operating system. If this message is seen, please contact a DiFX developer and be sure to indicate exactly which operating system and computer type you are using.



- **DataStream assumes int is 4 bytes, it is  $x$  bytes - aborting!!!:** This should not occur on any modern operating system. If this message is seen, please contact a DiFX developer and be sure to indicate exactly which operating system and computer type you are using.
- **DataStream *mpiid*: expected  $x$  bytes, got  $y$  bytes - aborting!!!:** In an eVLBI read, the wrong number of bytes was received, indicating a mismatch in send/receive setups. Check the setup at both ends and try again.
- **Datastream *mpiid*: implied UDP packet size is negative - aborting!!!:** When considering the size of a UDP header, an unphysical packet size results. This should only occur when attempting UDP based eVLBI.
- **Datastream *mpiid*: read too few UDP packets: bytestoread= $x$  udp\_offset= $y$  bytes= $z$  - aborting!!!:** Fewer than expected UDP packets were received in an eVLBI transfer (FIXME – more details please!)
- **Datastream *mpiid*: read too many UDP packets: bytestoread= $x$  udp\_offset= $y$  bytes= $z$  - aborting!!!:** More than expected UDP packets were received in an eVLBI transfer (FIXME – more details please!)
- **Datastream *mpiid*: could not allocate databuffer (length *size*) - aborting!!!:** A memory allocation failed. This would probably be due to either a developer error or attempt to run a job on an underpowered computer. In any case, the developer should be contacted.
- **Developer error: Cannot handle delays more negative than *maxDelay*. Need to unimplement the datastream check for negative delays to indicate bad data - aborting!!!:** The maximum negative delay (which is quite large) has been exceeded. This is almost certainly a logic error in the software and should be reported to a DiFX developer.
- **Developer error: in Mk5Mode::Mk5Mode, mark5stream is null:** An internal error related to initializing a Mark5 decoder has occurred. Contact a DiFX developer.
- **Developer error: in Mk5Mode::Mk5Mode, framesamples is inconsistent ( $x/y$ ):** An internal inconsistency has been found in the Mark5 data frame size that would lead to downstream errors. This could only be caused by a software logic error. Contact a DiFX developer.
- **Developer error: UVW has not been created!!!:** This message would come from an internal consistency check that failed. If this occurs, the DiFX developers should be notified as this indicates a logic error inside the software correlator.
- **genMk5FormatName : *formatType* format : framebytes = *frameBytes* is not allowed:** An illegal frame size was specified in the `.input` file. This should never happen for data going through the correlator in a standard way. If you see this message it is probably due to a programming error and the DiFX developers should be notified.
- **genMk5FormatName : unsupported format encountered:** A data format not handled by the software correlator has been requested in the `.input` file.
- **Input file out of order!:** There is an error in the input file. This might be caused by using an input file formatted for one version of `mpifxcorr` on a different version or by an incorrectly written file.
- **Manager aborting correlation!:** The configuration of a visibility buffer was not OK. There are many possible causes for this, but is most likely due to a logic error in the software. Contact a DiFX developer if this is encountered.
- **Mk5DataStream::readnetwork bytestoread too large ( $x/y$ ) - aborting!!!:** An eVLBI read size is too large. (FIXME – more details here please!)

- **mpifxcorr must be invoked with at least  $x$  processors (was invoked with  $y$  processors) - aborting!!!:** For a correlation of  $N$  datastreams (usually equal to the number of antennas), at least  $N + 2$ , but preferably even more, processes must be started.
- **NativeMk5DataStream::NativeMk5DataStream stub called, meaning mpifxcorr was not compiled for natively support, but it was requested (with MODULE in .input file) - aborting!!!:** Correlation directly off a Mark5 module requires compiling against the Streamstor libraries which was apparently not done but requested by the .input file. Contact the person responsible for your correlator setup and ask them to properly link the Streamstor libraries to the mpifxcorr executable.
- **No config section in input file:** The input file is missing its config section and hence correlation cannot proceed. This should never happen for data going through the correlator in a standard way.
- **Not enough baselines are supplied in the baseline table ( $x$ ) compared to the number of baselines ( $y$ )!!!:** The .input file requests more baselines in the common table than there are enumerated in the baseline table. This should never happen for data going through the correlator in a standard way.
- **Not enough datastreams are supplied in the datastream table ( $x$ ) compared to the number of datastreams ( $y$ )!!!** The .input file requests more datastreams (nominally equal to the number of antennas) in the common table than there are enumerated in the baseline table. This should never happen for data going through the correlator in a standard way.
- **Please invoke with mpifxcorr ...:** The correlator was not started according to its usage. See §7.50 for more details.
- **Polyco *polycoId* / *subcount* is malformed** The pulsar polynomial file is not compliant (possibly due to manual editing).
- **RPFITS not compiled in - aborting:** RPFITS output format is requested, but support for RPFITS is not compiled into mpifxcorr. Either requested output format, or recompile mpifxcorr with RPFITS. Note: This document does not contain instructions for RPFITS installations – you are on your own!
- **Samplesperblock is less than 1, current implementation cannot handle this situation - aborting!!!:** An illegal data sub-mode has been requested. Contact a developer.
- **Unknown data format *formatName*:** A data format (e.g., VLBA, Mark4, LBA) has been requested that cannot be processed.
- **Unknown data source *sourceName*:** A data source (e.g., FILE, MODULE, EVLBI) has been requested that cannot be used.

## 11.2 Severe

Severe errors typically reflect an unexpected software error. All severe errors are related to a process control (threading) failure or a failure in a numerical routine. Errors of the severe type are likely to cause widespread erroneous results or complete failure of correlation. Except during periods of software development errors of these two types are unexpected. If encountered, they should be reported to a DiFX developer and the correlation should be reattempted once. Since there are many errors in the “severe” class, all unlikely and with the same procedure for working around, individual errors of this type are not listed below.

### 11.3 Error

Messages in the “Error” class are typically fairly significant, often cascading to fatal messages and termination of the correlation. Many errors of the Mark5 variety result data loss ranging from fractions of a second to the full job in length. For errors of this type, operator judgement is needed: whether to restart correlation or keep going will depend on many circumstances. Note that many of the eVLBI and real-time monitor errors are not documented here yet.

- **All data from this module was discarded: ...:** Due either to malformed data or a directory file with incorrect values, no data was deemed suitable for correlation for a particular Mark5 module. It is probably worth trying to extract the module directory structure again and trying the job again and/or moving the module to a different unit for playback. It is likely that other jobs using this module will face a similar problem.
- **$n$  consecutive sync errors. Something is probably wrong!:** A large number of sync errors were seen. This probably means that the Mark5 data being read is somehow corrupted.
- **All bandwidths for a given datastream must be equal:** Currently, `mpifxcorr` does not allow different bandwidths on different frequency bands. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **All configs must have the same telescopes! Config  $m$  datastream  $n$  refers to different telescopes:** The same set of telescopes (antennas) must be used throughout a job. Two configurations have been found that use different antenna subsets. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **All LBASTD Modes must have 2 bit sampling - overriding input specification!!!:** The Australian LBASTD data format modes can only handle 2-bit (4 level) quantization at the moment. The `.input` file value for `QUANTISATION BITS` is being ignored here. Probably the maker of the `.input` file made a mistake.
- **Attempting to get a delay from offset time  $time$ , will take first—last source:** Correlation is requested for a time either before or after the list of scans. Data affected by this is likely to have the wrong delay applied and is unlikely to be useful. The creator of the input files should be contacted.
- **Attempting to refer to freq outside local table!!!:** The frequency index of a datastream table exceeds the length of the frequency table in the `.input` file. This will most certainly cause this frequency band to be incorrectly correlated.
- **Baseline table entry  $m$ , frequency  $n$ , polarisation product  $p$  for datastream  $q$  refers to a band outside datastream  $q$ 's range ( $r$ ):** The band index for a baseline table exceeds its legal range. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Baseline table entry  $m$  has a datastream index outside the datastream table range! Its two indices are  $n$ ,  $p$ :** The baseline table has a datastream index that exceeds the number of datastreams. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Cannot clear Mark5 write protect:** The software correlator attempts to set the Disk Module State to `PLAYED`. This requires temporarily disabling write protection. If this fails, then either the Mark5 unit is in a bad state or the module has a problem.
- **Cannot open data file  $dataFile$ :** The datastream process cannot open the specified file containing baseband data to be correlated. Correlation will proceed, but no data for an antenna for the duration of this file will be correlated. Usually this error should be a cause for concern.
- **Cannot open polyco file  $polycoFile$ :** Either file permissions, file location, or non-existence is preventing the file called `polycoFile` from being opened. This will likely end badly.

- **Cannot put Mark5 unit in bank mode:** The command to put the Mark5 unit in single bank mode failed and further commands to the Mark5 unit will probably fail as well. Usually this only happens if the Mark5 unit is in a bad state. The Mark5 unit probably needs a reboot.
- **Cannot read data from Mark5 module...:** Read from a Mark5 module failed. No further attempt to read data from the module will be made. It is strongly recommended that the Mark5 unit be rebooted and the correlation be reattempted. If the error is reproducible, the additional information contained at the end of this error message may help diagnose the problem.
- **Cannot read the Mark5 module label:** The command to retrieve the module label, which includes the volume serial number (VSN) and the previous state, has failed. This implies trouble with the Mark5 unit or module. First the module should be moved to a different Mark5 unit and the correlation reattempted. Upon further failure, the module should be checked for problems
- **Cannot set Mark5 data replacement mode / fill pattern:** Either the command to tell the Mark5 unit to enter real-time playback mode or the command to set the fill pattern have failed; expect more problems with this Mark5 unit. The Mark5 unit probably needs a reboot.
- **Cannot set the Mark5 module state:** The software correlator failed to set the Disk Module State to PLAYED. If this fails, then either the Mark5 unit is in a bad state or the module has a problem.
- **Cannot set Mark5 write protect:** The software correlator attempts to set the Disk Module State to PLAYED. This requires temporarily disabling write protection. If this fails, then either the Mark5 unit is in a bad state or the module has a problem.
- **Cannot unpack Mark5 format data at sampleoffset  $n$  from buffer  $time$ :** An error of this kind likely represents a logic error in the software correlator and should be reported to a DiFX developer.
- **Config  $m$  baseline index  $n$  refers to baseline  $p$  which is outside the range of the baseline table:** The baseline index of a config table exceeds the number of baselines in the baseline table. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Config  $m$  baseline index  $n$  refers to baseline  $p$  which is out of order with the previous baseline ...:** Entries in the datastream table are not in the expected order. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Could not find a polarisation pair, will be put in position  $x!!!$ :** This is due either to an uncaught inconsistency in the `.input` file or a logic error in the software correlator. Contact a DiFX developer.
- **Could not find any bands for frequency  $m$  of datastream  $n$ :** This is due either to an uncaught inconsistency in the `.input` file or a logic error in the software correlator. Contact a DiFX developer.
- **Could not open `threadFile` - will set all numthreads to 1!!!:** The requested `.threads` file could not be opened. Possible causes include: permission issues, inaccessibility of directory to the DiFX head node, and the file simply not existing. The impact of this is that correlation will proceed at a potentially much reduced speed since each CPU will use only a single processing thread. Accuracy of the results will not be affected.
- **Could not parse LBA file header:** An LBA format file appears corrupt. Data for one antenna for the duration of this file will not be correlated.
- **Datastream table entry  $m$  has a frequency index (freq  $n$ ) that refers outside the frequency table range ( $p$ ):** The `.input` file has a frequency index error. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.

- **Datastream table entry  $m$  has an input band local frequency index (band  $n$ ) that refers outside the local frequency table range ( $p$ ):** The `.input` file has a frequency index error. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Datastream table entry  $m$  has a telescope index that refers outside the telescope table range ( $n$ ):** The `.input` file has a telescope index error. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **FFT chunk time for config  $m$ , datastream  $n$  is not a whole number of nanoseconds ( $p$ ):** In order to keep track of time properly, each FFT must start on an integer nanosecond boundary. There are currently no modes supported where this should be a problem, so if you see this message, there is a more serious problem. Contact a DiFX developer!
- **First datastream for baseline  $n$  has a higher number than second datastream - reversing!!!:** The entries of the baseline table (indicating which antenna pairs to correlate) should always have the datastream indices in ascending order. If you see this warning, the correlator is correcting your baseline ordering, but be aware that this might be hinting that something else might be awry with the `.input` file. The creator of the `.input` file should be contacted.
- **Hit the end of the file! Setting the numthread for Core  $n$  to 1:** The `.threads` file ended unexpectedly early and one or more core processes will be forced to use a single processing thread, with potentially crippling performance penalty. Accuracy of the results will not be affected.
- **Increment per read in nanoseconds is  $x$  - too large to fit in an int:** The way timekeeping works in DiFX, data chunks larger than  $2^{31} - 1$  nanoseconds in length are not possible at the moment. The maker of the `.input` file needs to reduce the read size by changing some of the parameters (such as `NUM CHANNELS`, `BLOCKS PER SEND`, or possibly others).
- **lastoffsetns less than 0 still! =  $x$ :** This message probably indicates a logic error in the correlator program so should be reported to a DiFX developer.
- **Mk5DataStream::calculateControlParams : vlbaoffset= $x$  bufferindex= $y$  atsegment= $z$ :** A Mark5 data frame was found unaligned. A corresponding subintegration of data will be invalidated. A large number of messages of this type probably indicates corrupted data.
- **Module *VSN* contains undecoded scans!:** The module directory for module *VSN* has problems. Please correct the problem with the directory (as stored in `$MARK5_DIR_PATH`) and try again.
- **Module *VSN* not found in unit!:** The `.machines` file suggested that a specified Mark5 module would be found this this Mark5 unit but it was not. Check to make sure the module is in the unit.
- **Most of the data from this module was discarded: ...:** Due either to malformed data or a directory file with incorrect values, a large fraction of the data for this job from a particular module was not decodable. It is probably worth trying to extract the module directory structure again and trying the job again and/or moving the module to a different unit for playback. It is likely that other jobs using this module will face a similar problem.
- **No valid data found. Stopping playback!:** According to the directory for the module in question there is no valid data available to correlate. It is possible that a directory reconstruction will allow some or all of the data on the module to be retrieved.
- **Not all datastreams accounted for in the datastream table for config  $m$ :** All datastreams (roughly equivalent to antennas) must be represented in each configuration within the `.input` file. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.

- **Number of input bands for datastream  $m$  ( $n$ ) does not match with Mark5 file  $fileName$  ( $p$ ), will be ignored!!!:** The description of Mark5 baseband data (VLBA or Mark4 format) in the `.input` file is inconsistent with the actual content. The creator of the `.input` file should verify that the format is correctly specified. All data related to this condition will be left uncorrelated.
- **Oversamplefactor ( $m$ ) is less than decimation factor ( $n$ ):** The oversample factor must be an integer  $r$  times the decimation factor. Essentially oversampling is handled by two mechanisms: decimation of the input data stream in the data unpacker and through spectral selection at the time of FITS file creation. The oversampling factor of these two approaches must multiply to be the total *Oversamplefactor*. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Pulsar phase as calculated from the polyco will not be accurate over entire range of  $time$  as the frequency is changing too rapidly. The maximum safe calc length would be  $x$  - try reducing blocks per send or numchannels ...:** A detail of the way the pulsar polynomial is used is likely to cause imperfect binning. Please consult with the PI and the creator of the `.input` file.
- **Stale data was received from core  $n$  regarding time  $time$  seconds - it will be ignored!!!:** One subintegration of data is being discarded as it did not arrive in time for the data to be written to disk. If this is a chronic problem, increasing the value of `VIS BUFFER LENGTH` in the `.input` might help. A small number of errors of this type is not a problem.
- **Telescope  $antName$  could not be found in the uvw file!!!:** Data for this antenna will not have useful UVW values in its output file. Contact the creator of the `.input` file.
- **There must be an integer number of sends per datasegment. Presently databufferfactor is  $m$ , and numdatasegments is  $n$ :** The `.input` file contains an inconsistency in its parameterization of the various buffer sizes. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Trying to read past the end of file!!!:** One of the files read by `mpifxforr` ended prematurely. The maker of the `.input` file should be contacted.
- **Unknown output format  $outFormat$  (case sensitive choices are RPFITS, SWIN and ASCII):** `mpifxcorr` currently supports 3 output formats (ASCII, DIFX and RPFITS) and the requested one does not match one of these. This will lead to termination of the correlator. The creator of the `.input` file should be contacted.
- **Unsupported format or mode requested!!!:** A data format has been requested that is recognized but not supported by the correlator. Scans using this format (probably all scans for the antenna in question) will produce no data. The maker of the `.input` file should be contacted.
- **Waited 6 seconds for a Mark5 read and gave up.:** A Mark5 read timed out. A small number of such errors can be tolerated, especially for a module that is known to be bad, but many such messages should prompt a second correlation attempt after module move / unit reboot.
- **We thought we were reading something starting with 'something', when we actually got 'somethingElse':** The ordering or content of one of the files being read by `mpifxcorr` does not match expectations and is thus non-conformant. The maker of the file should be contacted. Note that this could be caused by a version mismatch between in the input files and the correlator.
- **XLRCardReset() failed. Remainder of data from this antenna will not be correlated and a reboot of this Mark5 unit is probably needed.:** It is probably worth reattempting correlation after Mark5 reboot and possible module move.

- **XLROpen() failed. Remainder of data from this antenna will not be correlated and a reboot of this Mark5 unit is probably needed.:** After a successful Streamstor card reset, the card was not able to be accessed. It is probably worth reattempting correlation after Mark5 reboot and possible module move.

## 11.4 Warning

Warning level messages typically relate to a situation that may result in the loss of a very small amount of data or indicate some other irregularity to the operator. A single warning should not be of concern, but a large number of warnings should be noted.

- ***n* consecutive fill patterns at time *time*:** Instead of reading data from the module, the Streamstor card returned fill pattern for this many consecutive data frames. If the module is flakey, it may be possible to recover more data after moving the module to a different Mark5 unit, but usually warnings of this type indicate inevitable data loss due to fill pattern replacement.
- ***n* consecutive sync errors starting at readpos *p* ...:** Data was read off the module, but the sync word was not found. This either indicates attempted playback of the wrong data format, completely corrupted data, or valid data that has slipped samples and is thus unfortunately not processable by the software correlator. It might be worth a recorrelation attempt after moving the module, but not likely.
- **Cannot find a valid configindex to set Mk5-related info. Flagging this subint:** Application of the delay model caused a small amount of data to cross a scan boundary in a manner that required the flagging of an entire subintegration. Such an event should be very rare.
- **Connection to monitor socket still pending:** When real-time monitoring of the output visibilities is requested but no connection to a monitor program has been establish this warning may appear. A warning of this type is not associated with any data loss.
- **Copying a polyco with no frequency information!:** A pulsar polynomial without frequency information has been found. This could present problems in setting the gate properly across frequency channels, but could be intentional.
- **Could not find station *stationName* in the uvw file when making rpfits header!!! This station is not used in this correlation so its parameters will be initialised to 0!!!:** The `.input` file included a station not in the `.uvw` file, but this station is not used so the results won't be affected. This warning will only be issued for data being written in RPFITS format. The resultant RPFITS file will still list the missing antenna, but it's information will be bogus. This could confuse downstream software.
- **Could not open command monitoring socket! Aborting message receive thread.:** Real-time monitoring was wanted, but a problem at the operating system level (perhaps permissions?) prevented the needed socket from being created. This has no bearing on the quality of the output data.
- **Data was received which is too recent (*x*sec + *y*ns)! ...:** A portion of one visibility record will have incomplete weight due to forced flushing of the long term accumulator buffer for that visibility record. A single warning of this type on source change is nothing to worry about, but constant warnings of this should be reported as this may indicate a different problem.
- **DataStream *mpiid*: could not identify Mark5 segment time (*formatName*):** An eVLBI data packet could not be decoded and hence its time cannot be determined. This is usually not a problem since time can be dead-reckoned from previous data and the corrupt packet won't be used anyway.

- **Filterbank not yet supported!!!:** The filterbank mode, enabled with the `FILTERBANK USED` option in the `.input` file, was requested, but is currently not supported. When supported, the filterbank mode will offer options for *crisper* spectral channels.
- **Fractional start time of  $x$  seconds plus  $y$  ns was specified, but the start time corresponded to a configuration not specified in the input file and hence we are skipping  $z$  seconds ahead! The ns offset will be set to 0!!!:** This warning indicates that the start time of the job is not contained within a scan to be correlated and the subsequent start time will be rounded to the next integer second. Except in cases where exact timestamp matching is needed this is not a problem. The warning is issued as it is not normal for the start time of a job to be outside a scan. If you see this a lot, contact the person generating your jobs and let them know.
- **FXMANAGER caught a signal and is going to shut down the correlator:** A terminate signal was sent to the manager process indicating that the correlator should be stopped immediately. The resultant output file will be incomplete, but the early stop was probably on purpose so that should be expected.
- **Hit end of first line prematurely - check your polyco conforms to standard! Some values may not have been set properly, but likely everything is ok:** The pulsar polynomial file had an unexpectedly short first line. Some parameters may not have been properly set so the pulsar gating may not work. Retrying correlation will not help! If you are concerned, contact the producer of the polynomial.
- **Hit end of second line prematurely - check your polyco conforms to standard! Some values may not have been set properly. This often happens for non-binary pulsars. Likely everything is ok.** It is possible that the pulsar polynomial file is incomplete, but more likely nothing is wrong.
- **Incorrectly set config index at scan boundary! Flagging this subint:** A subintegration that crosses the end of a scan got flagged as a result of a potential format change at this time. This should happen no more than once per scan and affects only a small fraction of one integration period in most cases.
- **Internal Error, trying to copy pass buffer size:** This is an eVLBI related error. (FIXME – please add more description here)
- **Mk5DataStream::calculateControlParams : bufferindex= $x$  >= bufferbytes= $y$ :** A fraction of a dataframe is being discarded due to a frame misalignment. This affects a very small amount of data and should be very rare.
- **Module label is not terminated!:** A Mark5 module has an oversized extended serial number label. This in itself is not a problem but may indicate either corruption on the module or poor seating of either the Mark5 module or one of the cables/cards inside the Mark5 unit.
- **Module label record separator not found!:** No “Disk Module State” was stored on this module. This is probably due to recording on a very old version of Mark5A or is possibly the result of some other incompatibility.
- **MPI Id *mpiid*: warning - received a parameter instruction regarding *paramName* which cannot be honored and will be ignored!** The software correlator can receive a number of different commands that can affect its dumping of its long-term accumulator to an external piggy-back processor. If an unrecognized parameter is received a warning of this type will be issued. This is completely unrelated to correlator data quality.



- **No more data for project on module *mpiid*:** Although data on the module extends past the end of the current job, none of the remaining data is relevant for this job so playback is stopping early. This is only a problem if it is due to an incorrect transcription of the module directory. It might be prudent to look at the observe log to see if data is expected.
- **No more data on module *mpiid*:** The Mark5 unit sending this message is stopping its reading before the end of the job since no more data exists. This is only a problem if it is due to an incorrect transcription of the module directory. It might be prudent to look at the observe log to see if data is expected.
- **Post-f fringe rotation not yet tested - use at your own risk!!!:** Post FFT fringe rotation (enabled with the POST-F FRINGE ROT `.input` file option) has not been extensively tested. Results may be OK, but you are on your own!
- **Trying to read *s* seconds past the end of the UVW array!:** The `.uvw` file does not cover an entire scan. If more than a couple such warnings are seen, the creator of the `.input` and `.calc` file should be notified.
- **Waited *s* sec state=*state*:** A Mark5 module is taking longer than expected to respond. Additional messages will follow if this situation is serious.
- **XLRCardReset() being called!:** If a read timeout occurs, the Streamstor card in the Mark5 unit will be reset. This message indicates that this non-standard procedure is occurring. The result of this reset will usually either be success (in which case a message will indicate such), failure (in which case an error message will state that no more data will be read from the module), or a Mark5 unit hang, and hence the correlation will stop. If this is not successful, it is recommended that the correlation be retried after rebooting the affected Mark5 unit and possibly moving the module to a different Mark5 unit.

## 11.5 Info

Info messages convey normal messages to the operator.

## 11.6 Verbose

Verbose messages convey normal messages to the operator that are either not very important or come in vast quantities; they are typically filtered out of data logging to prevent unnecessary bloat.

## 11.7 Debug

Debug messages are useful only to developers and don't usually indicate error conditions. In some cases they might be useful in diagnosing a problem. None of the debug messages are explicitly documented here because they will be very version dependent and in general provide little value to the operator.

# 12 Installation and upgrade guide

There are at least three methods of installing DiFX employed by its various users. Ironically, these do not include some of the more common installation mechanisms such as those offered natively by various Linux distributions (e.g., `.rpm` or `.deb` files). DiFX has many modules, some of which have dependencies that are not easily met or that are not needed. Some modules have optional dependencies. Finally, many folks may wish to have several versions installed at one time. These considerations and the effort involved in overcoming some of them have led to the current situation where package-by-package compilation is the standard mechanism for installation.

The sections that follow document two of these methods. First an install method using the `difxbuild` script is described. Then a more manual method is described.

## 12.1 Installation with `difxbuild`

### 12.1.1 Introduction

This installation guide is based on the python program `difxbuild`. This program allows installation and management of multiple versions, each on multiple platforms, and associate setup scripts. First some terminology: A *version* is a official numbered release of DiFX or an unofficial intermediate version drawn from the subversion repository. Examples are the recent stable 2.1 release and the head of the development, called trunk. Each *version* described here has a name. The name for the two versions mentioned here are `DIFX-2.1` and `DIFX-DEVEL` respectively. An *architecture* (or *arch* for short) is a computer type, usually defined by the instruction set of the CPU. Currently `difxbuild` explicitly knows about two architectures: `x86_64` for 64-bit Intel CPUs and `i386` for 32-bit Intel CPUs. The architecture of your machine can be determined on the command line with `uname -p`. Finally a *platform* is a particular configuration defined both by the *architecture* and the set of dependent software installed on it. For example, if multiple Mark5 units with different software development kit (SDK) versions are being used, each would be a different *platform*. For each DiFX installation there is a primary, or default, *platform* simply identified by the name of the *version*. Each additional *platform* is assigned an additional name (which could be `SDK8` and `SDK9` for the Mark5 situation) and is identified by concatenating the version name, a hyphen, and the additional name (e.g., `DIFX-2.1-SDK8`).

The installation process has a number of steps, including bootstrapping, source acquisition and autotooling, building (separately on each *platform*). Any of these steps can be repeated if needed, however, in many cases it does not make sense to repeat a single step out of order, so following steps may need to be performed to be meaningful.

In the description below, each place where a command is to be typed into the computer, it is displayed after a `→`.

### 12.1.2 Assumptions

To simplify the installation and eventual execution of DiFX, some assumptions are explicitly made:

1. It is assumed that before the DiFX installation is started that a fully usable Linux operating system is already running and a few bits of software are installed. This specifically includes the Intel Integrated Performance Primitives (IPP) which must be directly acquired through Intel's web site.
2. It is assumed that all machines running DiFX cross mount the same filesystem, that the local name of the cross-mounted filesystem is the same on each machine, and that all parts of the DiFX installation will reside on such a partition.
3. It is assumed that during source acquisition steps the machine on which `code;difxbuild;code` is being invoked has access to the internet (specifically `http` and `svn`).
4. It is assumed here that `bash` (or a compatible equivalent) is the shell used both by root and the user. If this is not met, it is up to the user to make any needed procedural changes.

### 12.1.3 Installation Part 1

Part 1 of installation deals with aspects of the installation that are specific to one *version* but all *architectures* and *platforms*. For each new *version* of DiFX, these steps will be repeated.

#### Bootstrapping:

The bootstrapping step can start from a pristine computer (as long as the above assumptions are met) and will generate a skeleton DiFX environment from a rather simple input file. This bootstrap input file

consists of a few lines of ASCII text that describe in a minimal manner the intended parameters of the DiFX installation. A complete description of such a file can be found in §8.5.

Below is an example `.bootstrap` file:

```
#-----
# here are version-specific parameters
#-----

# version of DiFX installed by this file
version = DIFX-DEVEL

#-----
# below here, all installed versions should look the same
#-----

# identify which node should run the core process
headnode = node-1

# top level directory for all DiFX software
difxbase = /home/usno/difx

# location of installed Intel Integrated Performance Primitives
iproot = /home/usno/intel

# define mark5 alternate architecture
altplatform1      = SDK9
altplatform1arch  = i686
altplatform1test  = [[ x'pkg-config --modversion streamstor' &gt; "x9.0" ]]
altplatform1host  = mark5fx-usno-1

# MPI network selection: restrict which network devices are used
mca = btl_tcp_if_include=p2p1
```

This file could logically be called `trunk.bootstrap` as is assumed here. Note a similar file for DiFX version 2.1 could be made by simply changing *version*. Bootstrapping is executed as follows (with optional `-v` option for increased verbosity):

```
→ difxbuild -v bootstrap trunk.bootstrap
```

If this completes successfully, you will be told to source the new setup file:

```
→ . /home/usno/difx/DIFX_DEVEL/setup_difx
```

This command must be issued before installation can continue. It should be reissued after each new shell is started, and must be reissued if changes are made to the `.bootstrap` file and bootstrapping is redone.

#### Check out source code from subversion:

A single command will cause the built-in selection of components to be downloaded from the ATNF subversion repository:

```
→ difxbuild -v svn all
```

The `all` parameter here, and in later commands, refers to all components (modules) supported by `difxbuild` for the version of DiFX being installed. To see which components this would apply to:

```
→ difxbuild list
```

If the `all` is excluded, the component corresponding to your current working directory (which would be none at this point) would be selected. Alternately, a list of components can be selected. Each component's source will be put in a separate subdirectory of `$DIFX_SRC`.

### Configure the source trees for out-of-tree building:

This step runs the “autotools” on the selected components. To achieve the purpose of supporting multiple *platforms*, all building is performed out of the source directories, so this step stops short of running `configure` itself.

```
→ difxbuild -v autotool all
```

#### 12.1.4 Set this version of DiFX as the default version

If you want this version of DiFX to be the default:

```
→ difxbuild -v default
```

This step simply makes a symlink to the newly created `setup_difx` script. Note that this step can be performed at any time. Changing to a different default version is done by sourcing the `setup_difx` script for that version and running this command.

#### 12.1.5 Installation Part 2

Part 2 of the installation deals with installations of *architecture* dependent code that can work across different *versions* (and *platforms* as long as they are of the same *architecture*). Sharing these bits of code across different *versions* requires that the base directory, as specified in the bootstrapping stage, are the same for each *version*. Repeat these steps for each *architecture* by logging into a representative machine of each *architecture*, sourcing the appropriate `setup_difx` file, and continuing... Note that several extra libraries such as PGPLOT are almost certainly not needed for any of the alternate platforms.

#### Installing OpenMPI:

Most Linux operating systems come with some version of OpenMPI these days, but most won't work for DiFX installations with multiple *architectures* as a particular configure-time parameter (`--enable-heterogeneous`) is usually not set. To download and install the latest stable version of OpenMPI:

```
→ difxbuild -v openmpi
```

#### Installing Caltech's PGPLOT library (optional):

If you want to build the “sniffer” plotting tools or hops, you need to install the `pgplot` plotting library:

```
→ difxbuild -v pgplot
```

#### 12.1.6 Installation Part 3

The 3rd part of installation must be done once for each *platform* (and always separately for each *version*, there are no shortcuts here!) This is the actual source code building step. For the non-primary *platforms*, simply log onto one of the machines representing that *platform* and be sure to source the appropriate `setup_difx` file and then proceed.

#### Build DiFX:

This part is simple, but may take a few minutes:

```
→ difxbuild -v build all
```

#### 12.1.7 Installation Part 4

The final step of installation is configuring the account of the user that will run DiFX. It is assumed here that this account is called `oper` and the account used for installation was `difxmgr` (but these are for example only; any usernames can be used). The only remaining steps are to ensure the environment is correctly configured by copying some files from the `difxmgr` account.

```
→ echo ". /home/usno/difx/bin/setup_difx" >> /.profile
```

```
→ ln -s /.profile /.bashrc
```

At this point once `oper` logs back in DiFX should be ready to run.

### 12.1.8 Upgrading the installation

## 12.2 Manual installation

This section describes module-by-module installation. This install method is not recommended in general and documentation for this may be out of date or eventually removed from this document. This method does give a deeper understanding of what actually happens behind the scenes when using the other methods and so may be useful to read through in any case.

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`: §12.2.4. 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 §3.2) 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](mailto:wbrisken@nrao.edu). Be warned that these instructions may change.

All of the subversion repositories below point to a `difx-1.5` tagged release 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 `tags/difx-1.5` 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!

### 12.2.1 expat

Expat is a standard library for simple XML parsing. By default it is installed on almost all Linux distributions. If it is not, it can be downloaded and installed based on instructions that can be found at its web page: <http://expat.sourceforge.net/>.

### 12.2.2 cx\_Oracle

In the implementation of the VLBA-DiFX operations plans, with the exception of the DOI, all of the access to VLBA database is done using python programs that employ the cx\_Oracle library. This library directly talks to Oracle databases; its use in Python makes for nearly effortless database interfacing. To install:

1. Download the latest source distribution from <http://cx-oracle.sourceforge.net/> (ver. 5.1.2 as of this writing)
2. Decompress the contents into perhaps *sourcedir*; enter the newly created directory
3. Run `python setup.py build`
4. Make sure install directory exists: `mkdir -p $DIFXROOT/lib/python2.4/site-packages`
5. Run `python setup.py install --prefix=$DIFXROOT`

Notes:

1. You should substitute `python2.4` with the string appropriate for your python version.
2. `lib` may need to be replaced with `lib64` in the path above.
3. Proper installation can be tested by running `python` and typing `import cx_Oracle` at the `>>>` prompt. If another prompt is given without any “ImportError” message, then it should be installed properly.
4. The `setup.py` file from version 5.1.2 seems to have an incompatibility with RedHat Enterprise Linux 6 (and there may be other variants of this incompatibility). Inserting `extraCompileArgs.append("-D_USE_XOPEN2K8")` at a logical outer-level location around line 200 seems to fix this problem.

### 12.2.3 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. 1.4.2 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.

### 12.2.4 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: `/home/swc/difx/intel/ipp/6.0.2.076/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.

### 12.2.5 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.

### 12.2.6 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/libraries/difxio/branches/difx-1.5 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.

### 12.2.7 difxmessage (optional)

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 DiFX system. See §9 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/libraries/difxmessage/branches/difx-1.5 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.

### 12.2.8 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/libraries/mark5access/branches/difx-1.5 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.

### 12.2.9 mpifxcorr

The core of the DiFX software correlator is `mpifxcorr`. Installation and running this program requires that MPI (§12.2.3), IPP (§12.2.4), `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/difx-1.5 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`

### 12.2.10 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 DiFX, the only program that makes use of this server is `calcif2` (§7.1). To install:

1. `cd sourcedir`
2. Check out the subversion repository:  
`svn co https://svn.atnf.csiro.au/difx/applications/calcserver/branches/difx-1.5 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`

Since `calcserver` is a single-instance program that is always running as a service, it is usually convenient to have this program start upon boot of the `calcserver` host. The `calcserver` distribution produces a file called `srcDir/calcserver/init.d/calcserver` that can be copied (as root) to the system `/etc/init.d` directory. After doing so, the `/etc/rc.d` directories may need to be updated to run this script at the right time. On RedHat systems, this is done with:

```
/sbin/chkconfig --add calcserver
```

### 12.2.11 calcif2

The `calcif2` package contains several programs that are useful for DiFX input file creation and managing correlation, most notably `calcif2`. Note that this package used to be called `job2difx`. To install:

1. `cd sourcedir`
2. Check out the subversion repository:  
`svn co https://svn.atnf.csiro.au/difx/utilities/branches/difx-1.5/calcif2 calcif2`

3. Enter the new directory `cd calcif2`
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 calcif2` should return `prefixdir/bin/calcif2`. Several other programs should also be installed, including: `genmachines`, `getjobs`, `jobdisks`, `joblist`, `jobstatus`, `difxsniff`, `mk5take`, `mk5return`, and `vlog`. If you wish to upgrade the installation:

1. `cd sourcedir/calcif2`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

### 12.2.12 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/applications/difx2fits/branches/difx-1.5 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`

### 12.2.13 `mk5daemon` (optional)

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 §7.46 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/applications/mk5daemon/branches/difx-1.5 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`

### 12.2.14 vex2difx

The `vex2difx` program aims to convert any legal, complete vex format experiment description file into `.input` and `.calc` files for use with DiFX. To install:

1. `cd sourcedir`
2. Check out the subversion repository:  
`svn co https://svn.atnf.csiro.au/difx/applications/vex2difx/branches/difx-1.5 vex2difx`
3. Enter the new directory `cd vex2difx`
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 vex2difx` should return `prefixdir/bin/vex2difx`. If you wish to upgrade the installation:

1. `cd sourcedir/vex2difx`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`

### 12.2.15 difx\_db (NRAO only)

Package `difx_db` contains several scripts that either make direct connection to the VLBA database or require site-specific access. Thus, this package is not available from the standard DiFX repositories. If any programs in this package seem especially appropriate for use at a correlator other than the VLBA DiFX correlator, please let me know and I'll see what I can do to make it more generally useful. To install:

1. `cd sourcedir`
2. Check out the subversion repository
3. Enter the new directory `cd difx_db`
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. `automake -a`

8. Generate the Makefile: `./configure --prefix=prefixdir`
9. Build it: `make`
10. Install it: `make install`
11. (*as root*) Change file ownership: `chown root prefixdir/e2ecopy`
12. (*as root*) Set UID: `chmod +s prefixdir/e2ecopy`

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

1. `cd sourcedir/vex2difx`
2. Get updates from the repository: `svn update`
3. Build it: `make`
4. Install it: `make install`
5. (*as root*) Change file ownership: `chown root prefixdir/e2ecopy`
6. (*as root*) Set UID: `chmod +s prefixdir/e2ecopy`

## 13 Acknowledgements

Many people have helped in significant ways to get DiFX adapted for the VLBA: 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 the VLBA; Steve Tingay and Matthew Bailes for allowing/encouraging Adam to write DiFX and helping support some of my travel in Australia; Chris Phillips, John Morgan, Helge Rottmann, Mark Kettenis, Olaf Wucknitz, Mariano Muscas, Geoff Crew, John Spitzak, Roger Cappallo, Walter Max-Moerbeck and Dmitry Baksheev for contributing code, bug reports, bug fixes, and ideas; Maria Davis, Roopesh Ojha and Dan Veillette for performing careful comparisons with geodetic data; Walter Alef for hosting the first and second DiFX workshops in 2007 and 2008 at MPIfR, Bonn; Miguel Guerra for helping define the database and XML structures and his work on the operator interface; Claire Chandler and Jon Romney/ for project oversight, test planning, advice, and voices of reason; Steven Durand for acquisition of computer and Mark5 equipment; David Boboltz, Geoff Bower, Mark Claussen, Vivek Dhawan, Alan Fey, Vincent Fish, Ed Fomalont, David Gordon, Miller Goss, Yuri Kovalev, Matt Lister, Enno Middelberg, James Miller-Jones, Amy Mioduszewski, Leonid Petrov, Mark Reid, Loránt Sjouwerman and Craig Walker for working directly with or examining the output of DiFX and providing valuable feedback; Cormac Reynolds for maintaining the Difx wiki which hosts valuable information, hosting the third DiFX workshop in Perth, and providing lots of code; Jan Wagner, Sergei Pogrebenko, Alexander Neidhardt, Martin Ettl, Jongsoo Kim and Randall Wayth for reporting bugs; Eric Greisen for making AIPS work well with DiFX output; John Benson for working with me to get data into the VLBA archive; James Robnett, Nicole Geiger, David Halstead and Pat van Buskirk for substantial computer and networking support and useful discussions on clustering; Mike Titus and Arthur Niell for providing test data; Leonia Kogan for carefully reviewing the conventions used; Pete Whiteis and Bill Sahr for contributing to helper programs; Ken Owens and Cindy Gold at Conduant for helping resolve Mark5 issues; Doug Gerrard, Bob McGoldrick, Adrian Rascon, K. Scott Rowe, Lothar Dahlmayer, and Jeff Long for assembling and maintaining the VLBA DiFX correlator; Juan Cordova, Paul Dyer, Lisa Foley, Heidi Medlin, Ken Hartley, Alan Kerr, Jim Ogle, Paul Padilla, Peggy Perley, Tony Perreault, Betty Ragan, Meri Stanley and Anthony Sowinski for providing operations assistance; Chet Ruszczyk, Dan Smythe, and John Ball for extensive Mark5 support over the last decade and Emma Goldberg for helping me work around the idiosyncrasies of L<sup>A</sup>T<sub>E</sub>X 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 category, I apologize — let me know and I'll make sure to add you for the next version's document.

## References

- [1] *VLBA-DIFX Operations Plan*, Brisken, W., 2009, VLBA Sensitivity Upgrade Memo 25, <http://www.vlba.nrao.edu/memos/sensi/sensimemo25.pdf>
- [2] *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, <http://xxx.lanl.gov/abs/astro-ph/0702141>
- [3] *DiFX2: A more flexible, efficient, robust and powerful software correlator*, A. T. Deller, W. F. Brisken, C. J. Phillips, J. Morgan, W. Alef, R. Cappallo, E. Middelberg, J. Romney, H. Rottmann, S. J. Tingay, R. Wayth, 2011, PASP, 123, 275, <http://arxiv.org/abs/1101.0885> *The FITS Interferometry Data Interchange Format*, Flatters, C., AIPS Memo 102, <ftp://ftp.aoc.nrao.edu/pub/software/aips/TEXT/PUBL/AIPSMEMO102.PS>
- [4] *The FITS Interferometry Data Interchange Convention*, Greisen, E., AIPS Memo 114r *in prep*, <ftp://ftp.aoc.nrao.edu/pub/software/aips/TEXT/PUBL/AIPSMEM114.PDF>
- [5] *Summary of the b-factor for the VLBA FX correlator*, Kogan, L., VLBA Scientific Memo 12, <http://www.vlba.nrao.edu/memos/sci/sci12memo.pdf>
- [6] *Tempo*, <http://www.atnf.csiro.au/research/pulsar/tempo/>
- [7] *JIRA*, <http://bugs.aoc.nrao.edu>
- [8] *NGAS*, <http://www.eso.org/projects/dfs/dfs-shared/web/ngas/>
- [9] *Difx Operator Interface Users Guide*, Guerra, M., **Coming soon!**
- [10] *VEX parameter definitions*, <http://www.haystack.mit.edu/tech/vlbi/mark5/vex.html>
- [11] *DiFX Developer Pages*, <http://cira.ivec.org/dokuwiki/doku.php/difx/start>

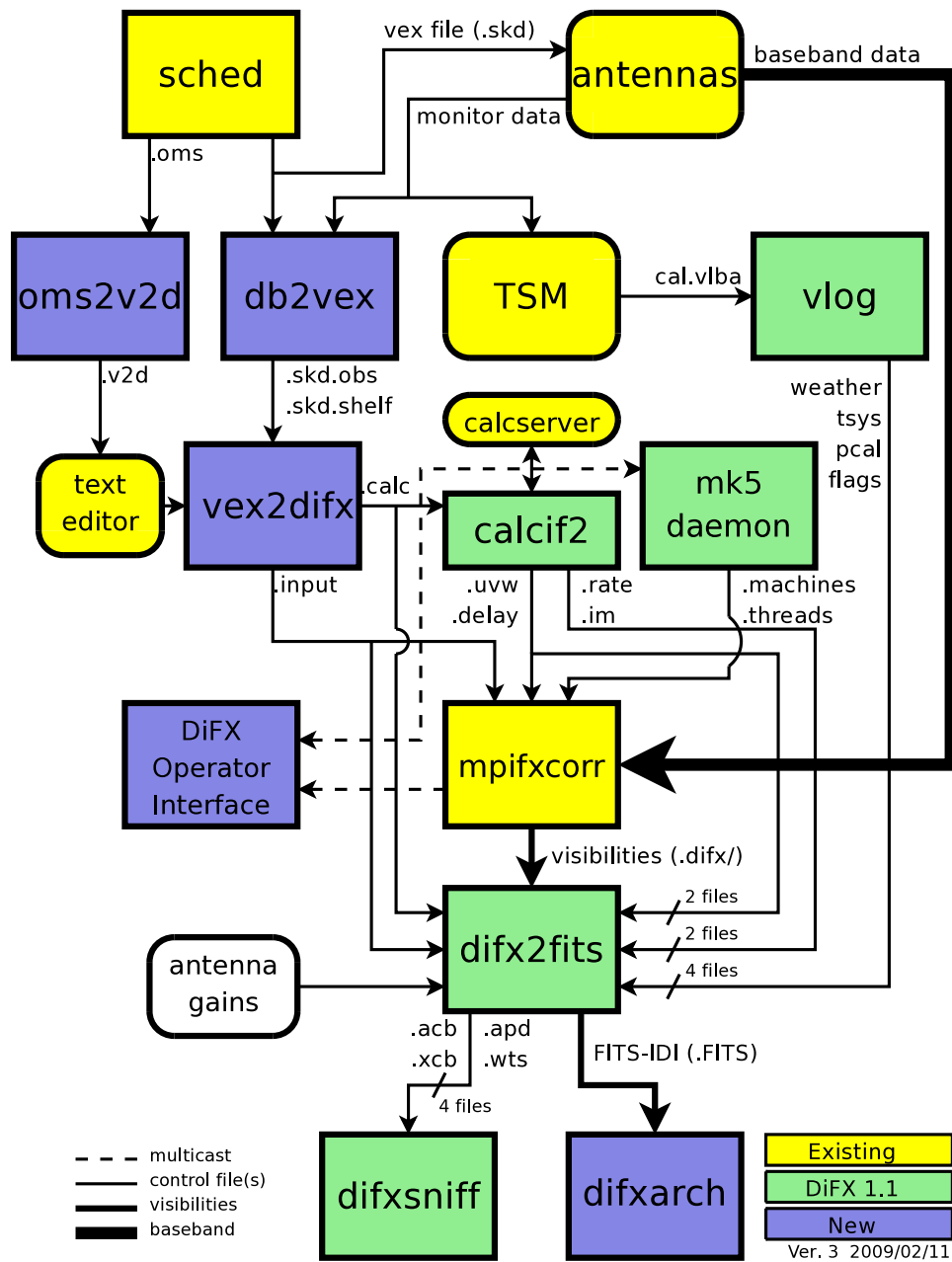


Figure 1: *The DiFX software correlator block diagram as implemented for the VLBA.*



```

/home/swc/ difx root directory: a 1.5 TB partition on swc000
├── difx/ all data related to operations ends up under this directory
│   ├── directories/ cached Mark5 module directories; environment variable
│   │   │   MARK5_DIR_PATH points here
│   │   ├── NRA0-120.dir
│   │   ├── NRA0-123.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
│   ├── intel/ directory containing various Intel Performance Primitives library versions
│   ├── projects/ active project data; environment variable JOB_ROOT
│   │   │   points here
│   │   ├── bc120e/
│   │   └── mt831/
│   ├── queue/ working directory for correlation
│   │   └── MT920/
│   ├── tar/ various source code tar balls
│   └── tests/ active test data; environment variable TESTS points here
├── DiFX-1.5/ version 1.5 files; set prefix to this during installation
│   ├── bin/ ver 1.5 programs
│   │   ├── calcif2
│   │   ├── vex2difx
│   │   ├── mpifxcorr
│   │   └── startdifx
│   ├── lib/ libraries for ver 1.5
│   ├── setup_difx script to set environment for version 1.5
│   └── src/ frozen source code for ver 1.5
│       ├── calcserver/
│       ├── difx2fits/
│       ├── difxio/
│       ├── difxmessage/
│       ├── job2difx/
│       │   └── calcif/
│       ├── mark5access/
│       ├── mk5daemon/
│       ├── mpifxcorr/
│       ├── openmpi-1.2.7/
│       └── vex2difx/
├── 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 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 DiFX-trunk are shown. Entries ending in ‘/’ are themselves directories.

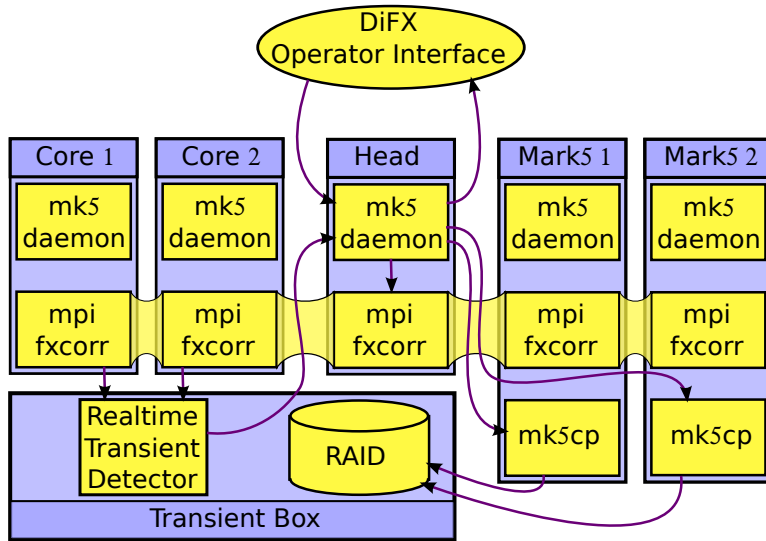


Figure 3: *The sequence of events leading to transient data capture.* First the DiFX Operator Interface sends a `DifxStart` message to the head node, causing the `mpifxcorr` process to be started. The core nodes of `mpifxcorr` will transmit autocorrelation data to a real-time transient detection algorithm. As interesting events are identified, `DifxTransientMessage` documents are sent to the head node; the `mk5daemon` process keeps a prioritized list of events. After correlation completes but before `mk5daemon` on the head node starts a series of `mk5scp` processes to run to copy the baseband data from the Mark5 modules before the next job starts.

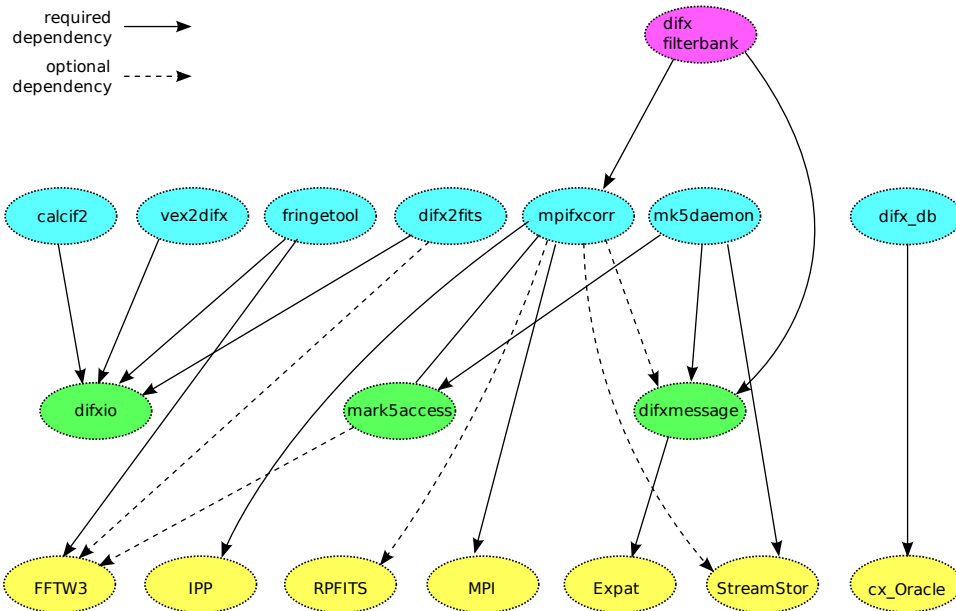


Figure 4: *A diagram illustrating the required and optional packages and their installation dependencies.*