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) and a configuration file (described below) and generates one or more .input files for use with difx. Each .input file is accompanied by a .calc file which is used by calcif2 to generate the .delay and .uvw files needed at correlation time. vex2difx along with calcif2 supercedes the functionality of vex2config and vex2model.

The vex2difx philosophy

Users and future developers of vex2difx should be aware of the approach used in designing vex2difx which can be summarized as follows:

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 for requests.

The vex file

The VLBI scheduling programs sched and sked both produce vex files that are used to control antennas for observations. Certain information that is not available prior to an observation needs to be provided to vex2difx in some way. One way is to append this data to the vex file. The alternative is to provide it in the .v2d file (as shown further down). This information includes:

1. The Earth orientation parameters ($EOP block in the vex file, or EOP blocks in the .v2d file)
2. The antenna clock offsets ($CLOCK block in the vex file, or clock values in the ANTENNA blocks of the .v2d file)
3. The volume serial numbers for the recording media ($TAPELOG_OBS block, or file lists in the ANTENNA blocks of the .v2d file)

Population of these three tables is necessarily a correlator/array specific operation and is the responsibility of the vex2difx user to arrange.

Note, only formal vex files are supported as input to vex2difx. Similar looking ovex files used at some/all Mark4 correlators are not acceptable, however, with a small amount of work an ovex file can be hand converted to a valid vex file. It would not be hard to write a conversion script to do this automatically.

The configuration file

The configuration 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.

Parameter 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 number or string representing Universal Time. Several formats are supported:
  - Modified Julian Day : `54345.341944`
  - Vex time format : `2009y245d08h12m24s`
  - VLBA-like format : `2009SEP02-08:12:24` (Note - between date and time!)
  - ISO 8601 format : `2009-09-02T08:12:24`

Specifying data formats

The `format` parameter of an ANTENNA or DATASTREAM section in the `.v2d` file, or the `track_frame_format` within a vex TRACKS section gives `vex2difx` information needed to determine how the data is arranged on the media. In the past (before DiFX 2.5) the two sources of format information had different formatting options. With DiFX 2.5 a new unified format decoding infrastructure has been added that give more flexibility. With this formats, either in vex or `.v2d` files can be specified in one of several ways:

- <fmt>
- <fmt>/<threads>/<size>/<bits> e.g., INTERLACEDVDIF/3:2:1:0/5032/2 or VDIF/7,8/8032/2 *VDIF only*
  - The comma separator for threads must be used within a `.vex` file
  - The colon separator for threads must be used within a `.v2d` file
- <fmt>/<size>/<bits> e.g., VDIF/5032/2, VDIFC/8032/8 *VDIF only*
- <fmt><size> e.g., VDIF5032 *VDIF only*
- <fmt>_<size>-<Mbps>-<nChan>-<bits> e.g., VDIF_5032-2048-4-2
- <fmt>/<size> e.g. VDIF/8032 (<size> < 32, bits assumed to be 2) *VDIF ONLY*
- <fmt>/<bits> e.g. VDIFC/8 (<bits> < 32, framesize assumed to be ??) *VDIF ONLY*
- <fmt>1_<fanout> e.g., VLBA1_4 *VLBA and Mark4 only*
- <fmt>-<Mbps>-<nChan>-<bits>
- <fmt>1_<fanout>-<Mbps>-<nChan>-<bits> *VLBA and Mark4 only*

The format class, <fmt>, can be one of the following:

- VDIF
- VDIFL (legacy VDIF)
- VDIFC (VDIF with complex samples)
- VDIFD (VDIF with double sideband complex samples)
- INTERLACEDVDIF (explicitly multi-thread VDIF – often interchangeable with VDIF)
- Mark5B
- KVN5B
- VLBA
- Mark4 or MKIV
- S2
- LBA
- LBAVSOP

Some general tips:

- A list of threads can be colon or comma separated.
- If a list of threads is provided it is assumed that the format is INTERLACEDVDIF, even if only VDIF is used to specify the format class.
- The size field always refers to the entire length of a data frame, including any headers.
- If the number of recorded channels provided is an integer multiple, *m*, of the thread count, then it is assumed that each thread has multiple channels. The ordering of the channels in the vex file is mapped to order of channels in vex as follows: the first *m* channels belong to the numerically first thread, the next *m* channels belong to the next thread, … Note 1. this is not yet implemented in mpifxcorr, and 2. vex2 will provide a more natural way to proceed.

Global Parameters

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

```
maxGap = 2000 # seconds
```

or

```
mjdStart = 52342.522 mjdStop=52342.532
```

Parameter name	Type	Units	Default	Comments
vex	string		REQUIRED	filename of the vex file to process; this is the only required parameter
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			mjd times of forced manual job breaks
minSubarray	int		2	don't make jobs for subarrays with fewer than this many antennas
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%) integration time to ensure integer blocks per send (newly re-enabled)
maxSize	float	MB	2000	The maximum output fits file size, estimated
singleScan	bool		False	if True, split each scan into its own job

Parameter name	Type	Units	Default	Comments
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
dataBufferFactor	int		32	the mpifxcorr DATABUFFERFACTOR parameter; see mpifxcorr documentation
nDataSegments	int		8	the mpifxcorr NUMDATASEGMENTS parameter
jobSeries	string		job	the base filename of .input and .calc files to be created
startSeries	int		1	the default starting number for jobs created
sendLength	float	sec	0	roughly the amount of data to send at a time from datastream processes to core processes
sendSize	int	bytes	5000000	roughly the send size from datastream to core
antennas	string		all ants.	a comma separated list of antennas to include in correlation
baselines	string		all bls.	a comma separated list of baselines; see below
padScans	bool		True	insert non-correlation scans in recording gaps to prevent mpifxcorr 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 mpifxcorr
simFXCORR	bool		False	simulate the VLBA hardware correlator integration and start times
overSamp	int			force all baseband channels to use the provided overSampling
mode	string		normal	options: normal and profile; see section below
threadsFile	string			overrides the name of the threads file to use
nCore	int			with nThread, cause a .threads file to be written
nThread	int			Number of threads per core to write to .threads file
machines	string			a list of machine names used to populate a .machines file
maxReadSize	int	bytes	25000000	Max read size in bytes (larger values cause issues with Mk5 module playback)
minReadSize	int	bytes	10000000	Min read size in bytes (smaller values mean probable inefficiency)
delayModel	string		calcif2	The executable (must be in path) of the delay model program to run (DiFX 2.5 and later)

Note that the baselines parameter supports the following syntaxes: A1-A2 A1+A2+A3-A4+A5 A1-* A1+A2-* and so on. For each list member, all baselines consistant with an antenna match on both sides will be kept.

SOURCE sections

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, e.g.:

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

or equivalently

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

Parameter name	Type	Units	Default	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			Path to a leap second kernel file for SPICE. Only used with near-field correlations
ephemObject	string			Name of the object from the ephemFile to be associated with this source. Only used for near-field correlations
ephemFile	string			Path to a planetary ephemeris file for SPICE. Only used with near-field correlations. bsp or tle files are allowed.
doPointingCentre	bool		true	Whether the pointing centre should be correlated (only ever turned off for multi-phase centre
addPhaseCentre	string			contains info on a source to add, with ra, dec and optionally name/calcode with no spaces, ”/“ separation and ”@“ in place of ”=“ e.g., “addPhaseCentre = name@1010-1212/RA@10:10:21.1/Dec@-12:12:00.34”

ANTENNA sections

An antenna section allows properties of an individual antenna, such as position, name, or clock/LO offsets, to be adjusted.

Parameter name	Type	Units	Default	Comments
name	string			New name to assign to this antenna
polSwap	bool		False	Swap the polarizations (i.e. L ←→ R) for this antenna
clockOffset	float	us	vex value	Overrides the clock offset value from the vex file; used in conjunction with clockEpoch
clockRate	float	us/s	vex value	Overrides the clock offset rate value from the vex file; used in conjunction with clockEpoch
clockEpoch	date		vex value	Overrides the epoch of the clock rate value; must be present if clockRate or clockOffset parameter is set
deltaClock	float	us	0.0	Adds to the clock offset (either the vex value or the clockOffset above)
deltaClockRate	float	us/s	0.0	Adds to the clock rate (either the vex value or the clockRate above)

Parameter name	Type	Units	Default	Comments
X	float	m	vex value	Change the X coordinate of the antenna location
Y	float	m	vex value	Change the Y coordinate of the antenna location
Z	float	m	vex value	Change the Z coordinate of the antenna location
format	string			Force format to be one of VLBA, MKIV, Mark5B, S2, VDIF or INTERLACEDVDIF, LBAVSOP, LBASTD
file	strings		(none)	A comma separated list of files that will be copied verbatim to the DATA TABLE of the input file
filelist	string			A filename listing files for the DATA TABLE and optionally mjdStart and mjdStop for each
networkPort	int			the eVLBI network port to use. This forces NETWORK media type in .input
windowSize	int			TCP window size for eVLBI. Set to <0 for UDP
UDP_MTU	int			Same as setting windowSize to negative of value
vsn	string			Override the Mark5 Module to be used
zoom	string			Uses the global zoom configuration with matching name for this antenna, e.g., zoom=Zoom1 will match the ZOOM block called Zoom1
addZoomFreq	string		noparent=true	Adds a zoom band with specified freq/bw as shown: freq@1810.0/bw@4.0[/specAvg@4][/noparent@ftrue]
freqClockOffs	string	us		Adds clock offsets to each recorded frequency using the following format:freqClockOffs=f1,f2,f3,f4 (must be same length as number of recorded freqs, first value must be zero). For a per-polarisation offset, use f1p1:f1p2delta,f2p1:f2p2delta,f3p1:f3p2delta,... where the f?p2delta values are the difference for the second polarisation of that frequency relative to the first.
loOffsets	string	Hz		Adds LO offsets to each recorded frequency using the following format:loOffsets=f1,f2,f3,f4 (must be same length as number of recorded freqs)
tcalFreq	int	Hz	0	Enables switched power detection at specified frequency
phaseCalInt	int	MHz	1	Zero turns off phase cal extraction, positive value is the interval between tones to be extracted
toneGuard	float	MHz	0.125 of bandwidth	When using toneSelection `smart` or `most` don't select tones within this range of band edge, if possible
toneSelection	string		`smart`	Use an algorithm to choose tones for you. Read the code to learn more.
sampling	string		REAL	Set to `COMPLEX` for complex sampled data or `COMPLEX_DSB` for double sideband
fake	bool		False	enable a fake data source
mjdStart	date		obs. start	discard any data from this antenna before this time
mjdStop	date		obs. stop	discard any data from this antenna after this time
machine	string			*Coming in ver. 2.5* if writing a .machines file, link this machine to this ANTENNA's datastream process
datastreams	strings		(none)	*Coming in ver. 2.5* links to DATASTREAM sections; below for more info

The addZoomFreq parameter freq always specifies the **lower edge** of the frequency channel, regardless of whether or not the parent band is USB or LSB.

The optional arguments for addZoomFreq control spectral averaging (currently constrained to be same as the parent band) and whether or not the parent band is still correlated - default is that it is *not* correlated. These are more for potential future compatibility.

If it is intended to run difx2fits, “FreqId” should be used in the SETUP section to select frequencies corresponding to parent band(s) for “addZoomFreq”, in the case where only a subset of recorded IFs have zoom bands. This avoids bands of different width being present in the same output job, which will cause difx2fits to fail.

the “freqClockOffs” parameter is intended for fixing small differences between frequency subbands, introduced by e.g. different cabling to parallel backends. It **cannot** be used to fix large offsets (e.g. integer seconds) between frequency subbands. The reason is that the samples from different frequency subbands are interleaved, so you have one block of data being FFT'd, and then these small corrections are applied after the FFT. So, at most they could correct for offsets of length one FFT duration - beyond that, there is no overlap between the two antennas any more! **At present, there is no way to correct for large (e.g. integer second offsets) on some but not all frequency channels, other than multiple correlation passes with the different clocks and a messy post-processing combination of the results (e.g. SPLIT, DBCON in AIPS).**

Legal values for `toneSelection` are `vex` `none` `all` `ends` `smart` or `most`:

`smart`	write the 2 most extreme tones at least toneGuard 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 toneGuard to band edge

VDIF is primarily supported in DiFX2. If a format of simply VDIF is given, the frame size and number of bits will be assumed to be 5032 bytes and 2 bits, respectively. Otherwise, you can specify frame size and number of bits with a format line like: “format=VDIF/5032/2” (for 5032 bytes and 2 bits, again). For interlaced VDIF (where multiple threads are present in one stream), presently the following simple case is supported - all threads must have the same frame size and number of bits, and the same bandwidth, and all must contain exactly one subband. The INTERLACEDVDIF format takes a list of such threads and multiplexes it on the fly back into a single multiple-subband VDIF thread. For INTERLACEDVDIF, you must present a fully specified format line, which has one additional parameter compared to normal VDIF: a list of the threadIds which are to be muxed. For example, if you had 4 single subband VDIF threads interlaced in a file, and they had thread Ids of 0, 1, 16 and 17, and the order of the subbands (when compared to the list of channels in the vex file) was 0, 1, 16 and 17 then the format line would be: “format=INTERLACEDVDIF/0:1:16:17/1032/2” for 1032 byte frames and 2 bit sampling.

Please note that vex uses as a clock sign convention that is positive for a formatter with its clock running fast (i.e., the second tick happens too early). The clockOffset and clockRate in this ANTENNA section, as well as FITS files, have the opposite sign convention. Hint: If you use the delays returned by AIPS FRING program as clock modifiers without changing their sign, you should end up with output that has no residual delay.

DATASTREAM sections (coming in DiFX 2.5)

New in upcoming DiFX version 2.5 will be support for multiple datastreams per antenna. Since vex1.5 does not have the concept of multiple datastreams per antenna the additional information must be provided explicitly in the .v2d file. Within .v2d files datastreams are linked to antennas. Logically speaking the datastreams are functions not only of antenna but also of setup; cases that have varying recording modes through an experiment invariably have changes to the datastreams, as used by DiFX, as well. Thus the implementation described here does not provide a fully general solution. In cases where this breaks down it is likely that multiple .v2d files, each acting on a subset of the setups used, will allow the needed flexibility. Note that when vex2 is fully supported, the STREAMS block within vex will give users access to the full generality of DiFX on a setup-by-setup basis.

To enable multiple datastreams for an antenna, simply define 2 or more DATASTREAM sections (described below) and link them with the appropriate antenna by using the datastreams parameter of ANTENNA sections. By default if there are *N* DATASTREAMS defined for an antenna, each will get one *N*th of the channels with the order of the channels preserved, meaning that the order of the datastreams argument does matter. This can be overridden with an nBand parameter.

Parameter name	Type	Units	Default	Comments
format	string			the data format for this (see below for more details)
sampling	string		REAL	Set to `COMPLEX` for complex sampled data or `COMPLEX_DSB` for double sideband
file	strings		(none)	A comma separated list of files that will be copied verbatim to the DATA TABLE of the input file
filelist	string			A filename listing files for the DATA TABLE and optionally mjdStart and mjdStop for each
networkPort	int			the eVLBI network port to use. This forces NETWORK media type in .input
windowSize	int			TCP window size for eVLBI. Set to <0 for UDP
UDP_MTU	int			Same as setting windowSize to negative of value
vsn	string			Override the Mark5 Module to be used
fake	bool		False	enable a fake data source
nBand	int			number of bands (baseband channels) to assign to this datastream
machine	string			if writing a `.machines` file, link this machine to this datastream's process

Specifying data formats is often tricky, especially in cases where vex doesn't properly support the particular format type (e.g., VDIF with vex1.5), or in the multiple datastream case. It is suggested to use a full format descriptor (e.g., “format=INTERLACEDVDIF/0:1:16:17/1032/2” rather than just “format=INTERLACEDVDIF”) even if the information should be present to fill in the gaps. In general vex2difx tries to require minimal information, but sometimes its assumptions may differ from yours.

SETUP sections

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. Note: The use of nChan (plus optionally specAvg) to set final (and FFT) spectral resolution is discouraged. It is maintained for backwards compatibility

and convenience, but if you have different subband bandwidths, you **cannot** use nChan, and must instead use specRes (and FFTSpecRes, if you want to explicitly set the FFT spectral resolution, for example in multifield projects).

Parameter name	Type	Units	Default	Comments
tInt	float	sec	2	integration time
FFTSpecRes	float	MHz	0.125	spectral resolution of first stage FFTs
specRes	float	MHz	0.5	spectral resolution of visibilities produced
nChan	int		16	number of post-averaged channels per spectral window; currently must be a power of 2. Do not use in combination with specRes/FFTSpecRes; nChan is only for convenience in simple cases (all stations have the same bandwidth for all subbands)
doPolar	bool		True	correlate cross hands when possible?
subintNS	int	ns	160000000	The mpifxcorr SUBINT NS; should eventually be set to a smarter default
guardNS	int	ns	0	The mpifxcorr GUARD NS; 2000 is usually sufficient. With DiFX 2.5, set to zero and mpifxcorr will calculate for you. Override may be needed for non-sidereal sources
maxNSBetweenUVShifts	int	ns	2000000000	Used for multiphase centre stuff. if better time resolution than 1 threads portion of a subint is required
maxNSBetweenACAvg	int	ns	2000000000	Used for STA dumping (transient searches) if better time resolution than 1 threads portion of a subint is required
specAvg	int		8	The spectral averaging to perform inside the correlator, at the end of a subint
fringeRotOrder	int		1	The fringe rotation order - 0=post-F, 1=linear, 2=quadratic
strideLength	int		16	The number of channels to “stride” for fringe rotation, fractional sample correction etc. With DiFX 2.5, set to zero for automatic setting on a per-datastream basis (usually good). This is almost mandatory for non-commensurate sample rates.
xmacLength	int		128	The number of channels to “stride” for cross-multiply accumulations. With DiFX 2.5, set to zero for automatic setting (usually good).
numBufferedFFTs	int		1	The number of FFTs to do in a row for each datastream, before XMAC'ing them all
postFFringe	bool		False	do fringe rotation after FFT?
binConfig	string		none	if specified, apply this pulsar bin configuration file to this setup
freqId	int list		none	a comma separated list of integers that are freq table indexes to select which bands to correlate; default is to correlate all. **Note:** this should be used to select parent bands for zoom frequencies if difx2fits is to be run.

Parameter name	Type	Units	Default	Comments
phasedArray	string			if specified, tells DiFX to produce a phased array output instead of cross correlations, using the setup specified in this phased array config file

EOP sections

It is possible to specify the Earth Orientation Parameters (EOPs) through the .v2d file. Normally these values will be appended to the vex file, but there may be cases where a completely unmodified vex file is desired (eVLBI maybe?). 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. It is not advised to mix EOP values stored in the vex and .v2d files.

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

Example section

```
EOP 55005 { tai_utc=34 ut1_utc=0.236958 xPole=0.10597 yPole=0.53906 }
```

RULE sections

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 but are required to be unique.

Parameter name	Type	Units	Default	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.

ZOOM sections

A zoom section specifies a list of zoom bands, using the same syntax used to add a zoom band to an individual antenna. This zoom setup can then be selected by any number of antennas (making it simpler, with less typing, to add the same zoom setup to many antennas)

Parameter name	Type	Units	Default	Comments
addZoomFreq	string			Adds a zoom band with specified freq/bw as shown: freq@1810.0/bw@4.0[/specAvg@4][/noparent@false]

Defaults are that noparent is true (e.g., parent bands are not correlated), and spectral averaging is performed as specified for the parent band (e.g., based on specRes and fftSpecRes.

An example ZOOM section that adds 4 zoom bands of width 8 MHz might look like:

```
ZOOM zoom1
{
  addZoomFreq = freq@1600.49/bw@8.0
  addZoomFreq = freq@1608.49/bw@8.0
  addZoomFreq = freq@1616.49/bw@8.0
  addZoomFreq = freq@1624.49/bw@8.0
}
```

COMMENT sections //coming in DiFX 2.5//

Starting with version 2.5.0 one can include COMMENT blocks in the .v2d file. These have no effect on the files written by vex2difx but allow comments (likely instructions to the person executing vex2difx) to be seen at the end of the output to the terminal. Each COMMENT block will make a new comment, separated by one line of whitespace in the output. A comment block starts with COMMENT { and ends with a } For example:

```
COMMENT
{
  Correlate this four times to see if we get the same answer.
}
```

Modes

Currently vex2difx operates in one of two modes:

- *normal* The output of vex2difx is a set of files useful for correlating the data. This is, as the name suggests, the normal mode of operation for vex2difx.
- *profile* This mode specializes in making files useful for forming pulse profiles in preparation for pulsar gating. 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.

Command line arguments

vex2difx is executed on the command line with:

`vex2difx` [options] inputFile

Although no command line options can change the way that vex2difx processes a file, there are some options that the user may find useful:

- `-h` or `–help` Print usage information to the screen. This is the same as if no arguments were supplied to vex2difx.
- `-o` or `–output` Writes a configuration file called *inputFile*`.params` which is a valid configuration file identical to *inputFile*, but with all assumed defaults populated. This is useful to see what was actually assumed.
- `-v` or `–verbose` Prints much more information to the screen. Use this option twice for even more information.
- `-d` or `–delete-old` Deletes all output from previous runs of vex2difx with same prefix. This is most useful when rerunning and a smaller number of jobs are created.
- `-s` or `–strict` Treat some warnings as errors and quit.

Reporting problems

If you have a problem with vex2difx, please email the difx users email group. Be sure to include the following in the email:

1. A description of the problem
2. The v2d file supplied to vex2difx
3. The vex file pointed to from the v2d file
4. the captured output when running vex2difx with extra verbosity (use options `-v -v`)

Examples

Trivial case

The following example demonstrates the simplest case where all defaults are assumed

```
vex=trivial.vex
```

Simple case

The following is a more realistic case for a simple experiment

```
vex=simple.vex

SETUP default
{
  nChan=64
  tInt =3.0
}
```

Source coordinate change

This shows how to change the coordinates of two sources in a file

```
vex=coords.vex

SOURCE J1232+131 { ra=12h32m15.12s dec=13d07'12.5" }
SOURCE PLANETX   { ra=11h59m59.999s dec=-12d59'59.88" }

SETUP default
{
  nChan=128
}
```

Two setups

This is a more complicated file showing how to apply different correlator setups to different sources

```
vex=twosetups.vex
maxGap=1000  # don't split the jobs at every source change,
             # instead, make just 2 interleaved jobs
antennas=BR,FD,HN,MK  # select only these four antennas for now

SETUP target
{
  nchan=1024
  tInt =1.2
}

SETUP calibrator
{
  nchan=32
  tInt =4
}

RULE calRule
{
  source=J1234+1231,3C84,3C273
  setup =calibrator
}

RULE targetRule
{
  # note: not specifying any restrictions so all sources that don't
  # match above will match here
  setup = target
}
```

The above could have used a default setup rather than a catch-all rule and resulted in the same output.

Specifying media

vex2difx allows .input file generation for two types of media. A single .input file can have different media types for different stations. Ensuring specification of media 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) files. It is OK 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. This “filelist” file only needs to be visible to vex2difx. This file contains a list of filenames and optionally start and stop dates (in one of the formats listed above). Comments can be started with a # and are ended by the end-of-line character. Like for the “file” keyword, the filenames listed must be in time order, even if start and stop dates are supplied. An example “filelist” file 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
```

```
ANTENNA default { networkPort = 320 }  # all antennas without ANTENNA setups
will get this
```

Splitting of jobs

Certain events cause a forced job break that could, in some cases, end up requiring many individual software correlations to complete processing of a project. Effort has been made to minimize the number of these cases. The following situations will cause a job break: change in clock model at a station, change of a Mark5 module, change in number of channels or sub-bands, multiple simultaneous subarrays, and leap seconds. Future versions of vex2difx and DiFX may relax some of these circumstances. Some parameters have defaults that may cause more job splitting than is desired (such as maxLength) and can be tuned.

Mark5B issues

The Mark5B format, including its 2048 Mbps extension, is now supported by vex2difx. The “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 complete $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 : &Ch03 : mag  : 1 : 07;
  fanout_def = A : &Ch04 : sign : 1 : 08;
  fanout_def = A : &Ch04 : mag  : 1 : 09;
  fanout_def = A : &Ch05 : sign : 1 : 10;
  fanout_def = A : &Ch05 : mag  : 1 : 11;
  fanout_def = A : &Ch06 : sign : 1 : 12;
  fanout_def = A : &Ch06 : mag  : 1 : 13;
  fanout_def = A : &Ch07 : sign : 1 : 14;
  fanout_def = A : &Ch07 : mag  : 1 : 15;
  fanout_def = A : &Ch08 : sign : 1 : 16;
  fanout_def = A : &Ch08 : mag  : 1 : 17;
  fanout_def = A : &Ch09 : sign : 1 : 18;
  fanout_def = A : &Ch09 : mag  : 1 : 19;
  fanout_def = A : &Ch10 : sign : 1 : 20;
  fanout_def = A : &Ch10 : mag  : 1 : 21;
  fanout_def = A : &Ch11 : sign : 1 : 22;
```

```
  fanout_def = A : &Ch11 : mag  : 1 : 23;
  fanout_def = A : &Ch12 : sign : 1 : 24;
  fanout_def = A : &Ch12 : mag  : 1 : 25;
  fanout_def = A : &Ch13 : sign : 1 : 26;
  fanout_def = A : &Ch13 : mag  : 1 : 27;
  fanout_def = A : &Ch14 : sign : 1 : 28;
  fanout_def = A : &Ch14 : mag  : 1 : 29;
  fanout_def = A : &Ch15 : sign : 1 : 30;
  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;
```

VDIF issues

VDIF, including “Legacy VDIF” is supported vex2difx. (Legacy support was added in 2.3). The “track assignments” for VDIF needs to be clarified. VDIF has no “tracks” and channels are specifically stored “in order” within a bitstream. The concept of “fanout” is also no longer used.

To indicate that the data is in VDIF format, one must either ensure that a statement of the form

```
track_frame_format = VDIF;
```

for non-legacy data or

```
track_frame_format = VDIFL;
```

for legacy data

must be present in the appropriate $TRACKS section or

```
format = VDIF
```

or

```
format = VDIFL
```

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

```
TOBEADDED
```

About the source code

vex2difx is written in c++ and makes heavy use of the standard template library. This makes applying standard algorithms (sorting, traversing, associating) container members simple and error-

free. An object-oriented approach is used. The base class for many of the classes is Interval, which is simply a pair of modified Julian days specifying a time interval. From this many other classes, such as scan, job, experiment, flag, … are derived. These classes are defined and implemented in files in the vexdatamodel/ subdirectory of the source code. This makes simple operations on Interval objects (such as sorting and determining overlap) apply automatically to the higher level objects. The vex parsing library from the Field System was borrowed from Goddard Space Flight Center. This code is duplicated with little change within the vex/ subdirectory of the vex2difx source tree. Source file vexload.cpp contains the code that calls the vex parser routines to populate the VexData structure which is then used as the model from which to make jobs. vex2difx uses the difxio library for writing DiFX .input and .calc files. Currently the .flag files are written natively within vex2difx, however this may change.

To aid diagnosis of an experiment and forming jobs, vex2difx keeps an internal list of *events*. An event could be the experiment starting or stopping, recording at a station starting or stopping, a leap second, an antenna joining or leaving a scan, and others. Event types are enumerated in the vextables.h source file.

Splitting of an experiment into one or more jobs is one of the main functions of vex2difx. The first step in this process is to divide the experiment into JobGroups. A JobGroup is a collection of scans that can be combined into one FITS file. Examples of cases where a JobGroup boundary must be made include changing number of spectral channels or polarizations. The JobGroup boundaries happen at exacting times, dictated entirely by the scheduled scans. The second layer of splitting considers media changes. Often there is a gap between the end of recording on one Mark5 module and beginning recording on the next. vex2difx aims to be smart about choosing when to split jobs to minimize the total number of jobs created.

vex2difx TODO list

List of remaining issues

- Add a “default” option for the antenna table
- ~~Improve handling of case mismatches (e.g., Ny vs. NY)~~
- ~~VDIF support~~
- Better handle modes/setups that don't use all provided antennas
- Extensive testing of many modes
- Support pulsar polyco with gate open/close support for simple gating
- ~~Support nAntenna != nDatastream~~
- ~~Don't require $DIFX_VERSION (see bugs below)~~
- ~~Warn if source, scan, antenna, or mode name referenced from .v2d file is not in vex file~~
- ~~eVLBI support~~
- ~~Support time formats other than decimal MJD~~
- ~~Mark5B support~~
- ~~Set up correlation off disk files rather than modules~~
- ~~Ability to select which baselines are retained or dumped (akin to antennas=)~~
- ~~Support IF selection~~
- ~~VLBA hardware correlator time alignment option~~
- ~~Improve blocks per send calculation~~
- ~~Support for polarization swapping~~
- ~~Support for ANTENNA sections~~
- ~~Improved ra, dec parsing~~

- ~~Write `.flag` file indicating baselines/antennas to turf after correlation~~
- Allow setting DataStream buffer as a total size in MB or seconds (e.g., 256 MB or 10sec)
- Use links from GLOBAL and STATION tables to CLOCK, EOP and TAPELOG_OBS as appropriate (example file: gk022.vix)

BUGS

- ~~Core dumps if DIFX_VERSION is not set~~ (fixed in DiFX-1.5 branch and trunk, 2010Mar04)
- ~~In eVLBI (NETWORK) mode, warns about missing media specification and does not add the Network Table to input file~~ (fixed in trunk, 2010July)
- Default AvChans is not 1
- If an antenna is not listed in the global `antennas` list, but has a ANTENNA section vex2difx should just skip this section, not return an error.

Feature Requests

- ~~By default, the "CORE CONF FILENAME" does not need to include the experiment name. "threads" is good enough (CJP)~~ (set with threadsFile= 2010July)
- ~~If only a single job is to be run, the output filename does not need the _# prefix. For the way ATNF eVLBI run this is particularly messy (as each time DiFX is run the wall clock start time is added to the file name)(CJP).~~ (enabled by setting startSeries=0 2010July03)
- ~~When correlating single pol antennas against dual pol, the (possible) crosspol products should be added to the baseline table, at least optionally. (CJP)~~
- ~~Simple VDIF support is needed (CJP)~~
- ~~Complex data type support is needed [boolean, default False] (CJP)~~
- Support input files with MSDOS or Linux style EOL markers (WFB)