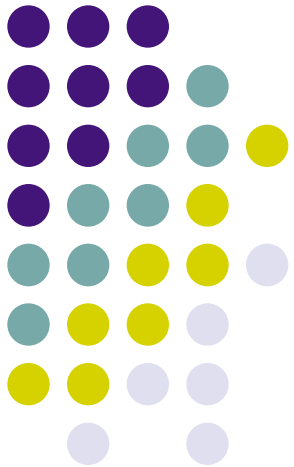
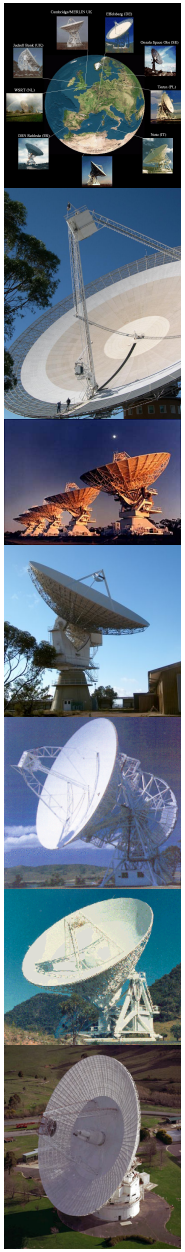
VDIF support in DiFX

Adam Deller
ASTRON

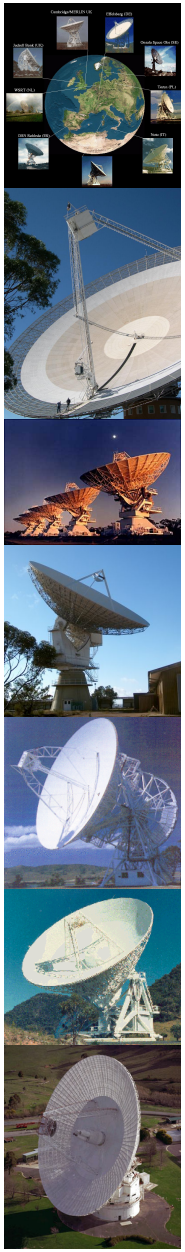


5th DiFX workshop, Haystack Observatory



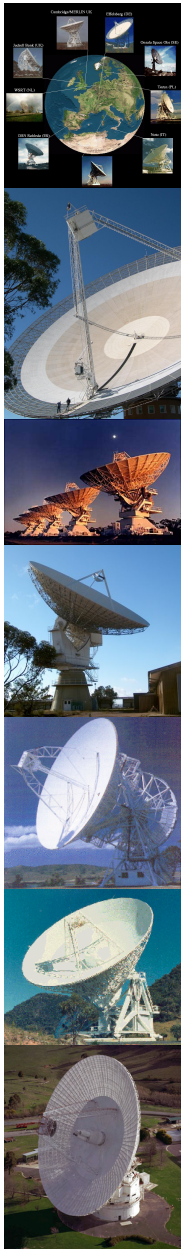
VLBI Data Interchange Format

- VDIF is a standard for storage of VLBI data
- Development began in 2008, version 1.0 agreed upon in 2009
- See www.vlbi.org/vdif/



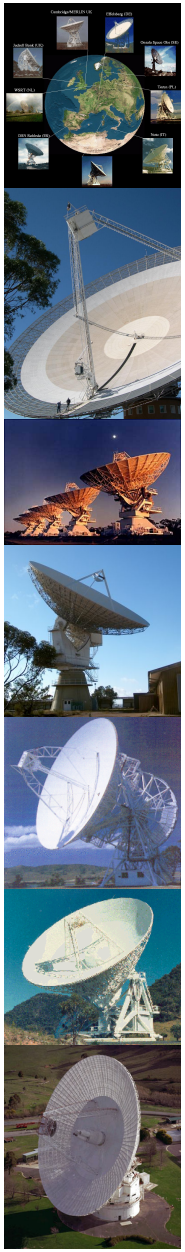
VLBI Data Interchange Format

- Driven by unsuitability of existing formats (Mark4, Mark5B, VLBA, PC-EVN) for modern backends, recorders, and correlators
  - These formats were either based upon or inherited tape-based baggage
  - No possibility for independent subband streams
  - Natural frame sizes poorly matched to UDP/TCP



VLBI Data Interchange Format

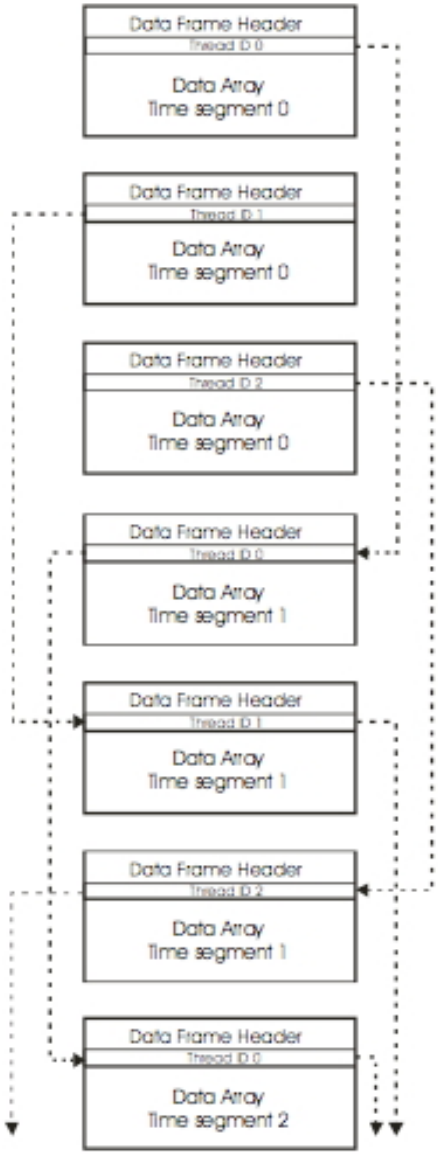
- VDIF allows for a number of independent “data threads”
  - Each of these can contain one or more subbands
- A data thread is comprised of a series of packets, each with a header
  - Generally, packets are sized to fit within an ethernet frame, although this is not required

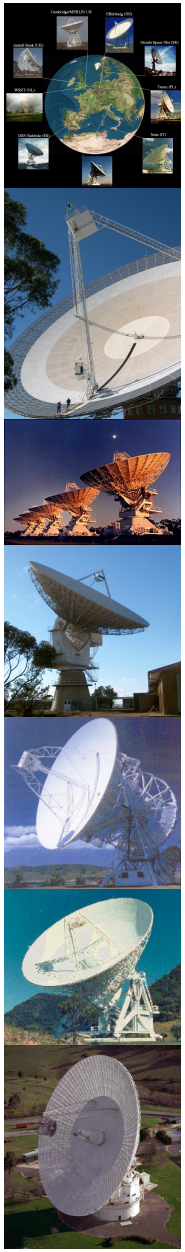


VLBI Data Interchange Format

	Bit 31 (MSB) Byte 3		Byte 2	Byte 1	Byte 0 Bit 0 (LSB)
Word 0	I_1	L_1	Seconds from reference epoch$_{30}$		
Word 1	Un-assigned$_2$	Ref Epoch$_6$	Data Frame # within second$_{24}$		
Word 2	V_3	$\log_2$(#chns)$_5$	Data Frame length (units of 8 bytes)$_{24}$		
Word 3	C_1	bits/sample-1$_5$	Thread ID$_{10}$	Station ID$_{16}$	
Word 4	EDV$_8$		Extended User Data$_{24}$		
Word 5	Extended User Data$_{32}$				
Word 6	Extended User Data$_{32}$				
Word 7	Extended User Data$_{32}$				

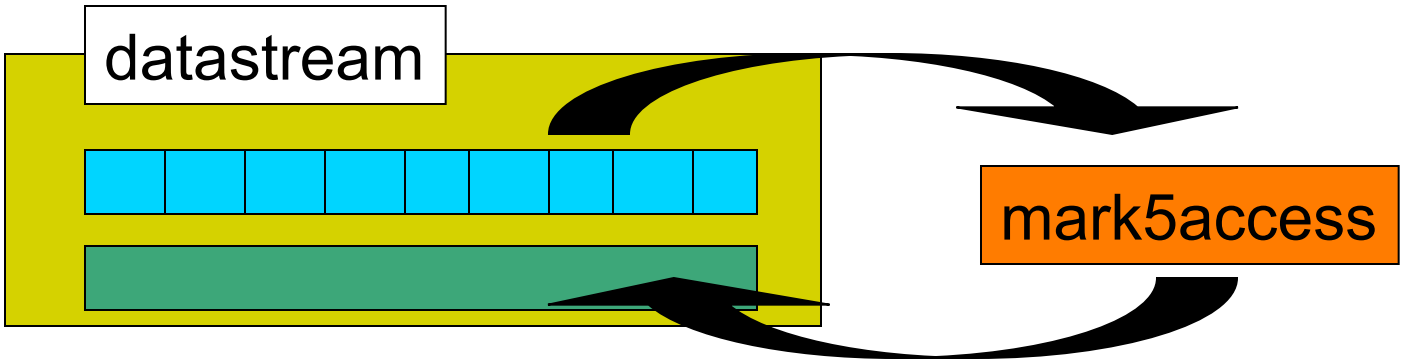
VLBI Data Interchange Format

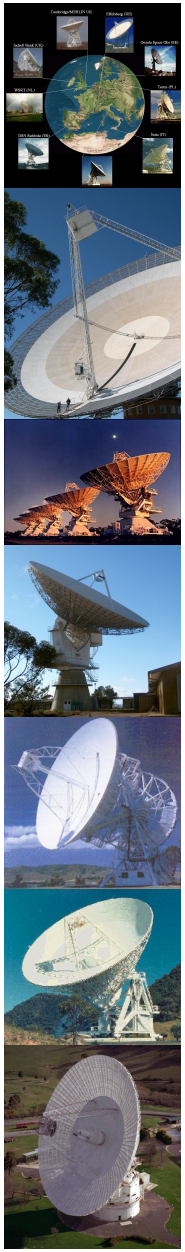




Status of VDIF support in DiFX

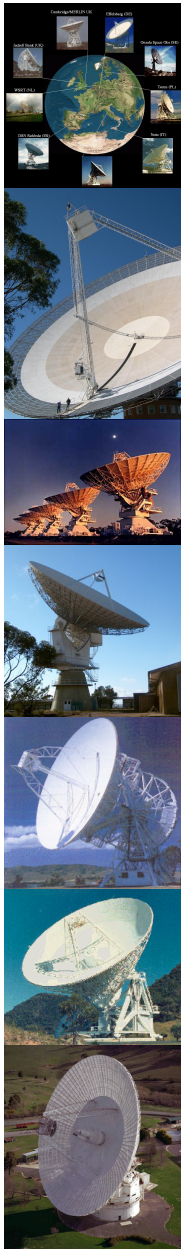
- Mark5access library (developed by Walter Brisken to unpack VLBA, Mark4 and Mark5B format data) extended to handle single-thread VDIF data
  - Situation very similar to existing formats
  - mark5access simply unpacks - the responsibility for ordering the packets etc rests with the caller, in this case mpifxcorr





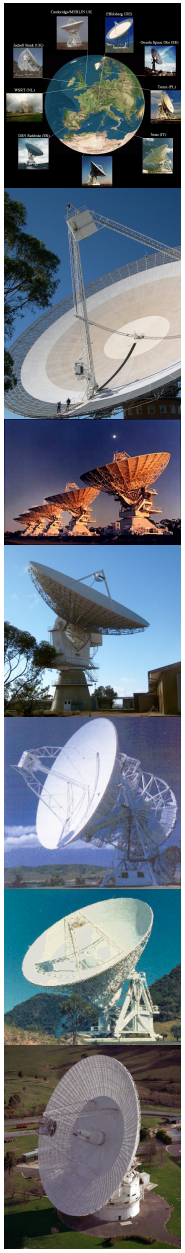
Challenges for VDIF in DiFX

1. Complex-sampled data
2. Out-of-order and/or missing packets
3. Multiple data sources for a single station



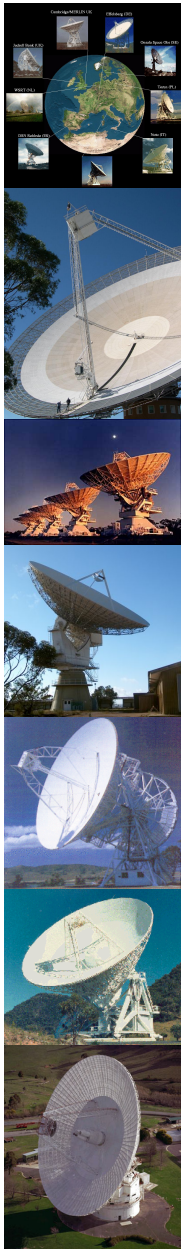
1. Complex-sampled data

- Now supported (not all sampling precisions, but the most common)
  - courtesy work by Chris Phillips
- Required work in correlator depths (unwinding many implicit assumptions of Nyquist sampling of real data)
  - Ultimately, complex data is actually simpler and more natural for mpifxcorr to handle
- Probably not yet bullet-proof



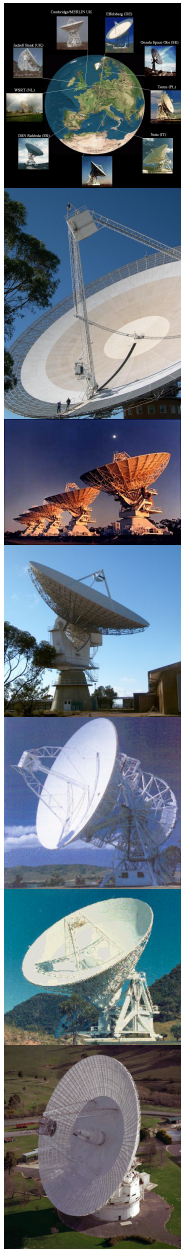
2. Out-of-order/missing packets

- Need a pre-buffering stage to sort packets and pad missing data
  1. Could make this external to mpifxcorr, and pass the result on via TCP (simple interface from the mpifxcorr point of view)
  2. Or, could just add an extra buffer inside the existing Datastream process inside mpifxcorr
- Closely related to the following problem…



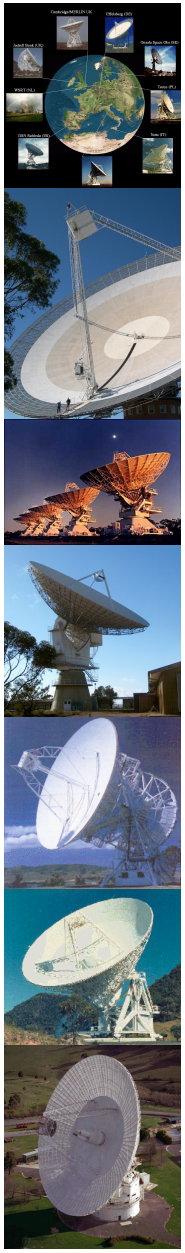
3. Single station, multiple sources

- Already supported by mpifxcorr for 5 years! But…
- Configuring this via vex2difx is not currently possible
  - Implicit assumption of a 1-1 match of “datastream” to “station”
  - Unwinding this is a large task
- Worse - mpifxcorr expects <u>independent</u> data sources for each datastream
  - Here, packets from different threads may be interleaved into one stream - need to share



Solutions 1. The ideal

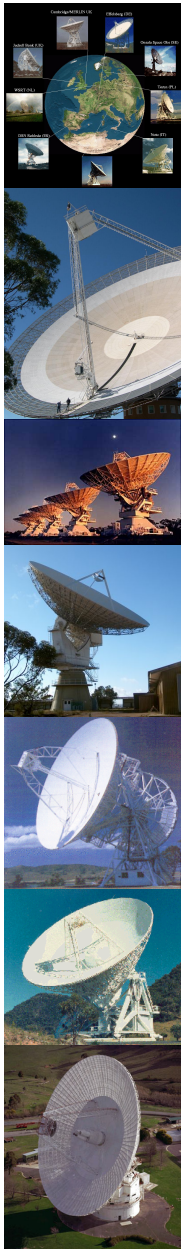
- A 1-to-many demultiplexer
  - Read a single VDIF stream in to a buffer, reorder it and split into N separate streams, send each stream on to a different location
  - Solve both problems in one step
- Could go even further; why restrict to VDIF format inputs?
  - On-the-fly reformatting
  - Could vastly simplify the interface to mpifxcorr, and allow frequency division multiplexing for non-VDIF formats



Solutions 1. The ideal

- A 1-to-many demultiplexer
  - Read a single VDIF stream in to a buffer, reorder it and split into N separate streams, s[illegible]ation
  - S[illegible]
- Cou[illegible]t to VD[illegible]
  - On-the-fly reformatting
  - Could vastly simplify the interface to mpifxcorr, and allow frequency division multiplexing for non-VDIF formats

A preliminary “VDIF server” has been begun, but virtually untested, far from feature complete, and totally unintegrated with vex2difx



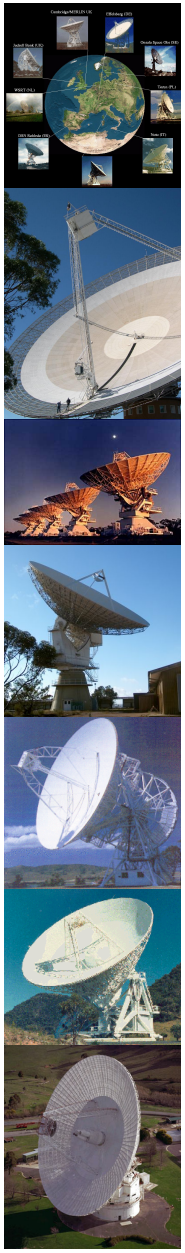
Solutions 2. The distasteful

- Solve the most common problem - trying to correlate a station with multiple identical VDIF threads (same # bits, etc) against “normal” stations
- Easiest - buffer inside Datastream, and re-multiplex back to a single VDIF thread
  - Oh, the aesthetic pain!
  - But; the infrastructure to deal with the output already exists, so this has been implemented

Solutions 2. The distasteful

- Solve the most common problem -
  - But; the infrastructure to deal with the output already exists, so this has been implemented

multiplexedVDIF is available as a mode in DiFX-2.0.1 onwards. It **is** integrated with vex2difx. However, it is not yet extensively tested, and the syntax to enable it is clumsy and long-winded.



Recap: VDIF status

- Single-thread VDIF, both real-sampled and complex-sampled data, is supported in DiFX
- Multiple-thread data is supported under restrictive circumstances
- The ideal future would include an uber-data reorderer external to mpifxcorr, but this is not yet on the agenda
  - But this is ultimately the best way forward to flexible, frequency distributed correlation