JOINT INSTITUTE FOR VLBI IN EUROPE

OUDE HOOGEVEENSEDIJK 4, DWINGELOO

Telephone: +31 (0) 592 596500
Fax: +31 (0) 592 596539

02 April 2020

TO: Distribution
FROM: Harro Verkouter
SUBJECT: *jive5ab* command set version 1.11 (*jive5ab* 2.4 and up)

1. *jive5ab* program

The commands detailed in this memo are implemented by a program named *jive5ab* and control the DIM functionality of the three types of Mark5 DIM VLBI data acquisition systems designed by MIT Haystack[1]: the Mark5A, B and C.

The program offers a limited set of DIM services on Mark5B DOM systems and hardware agnostic DIM services (including high-speed network packet capture) on regular computer systems, including Mark6 and FlexBuff.

In addition to this, *jive5ab* offers non-DIM and (real-time) e-VLBI functionality on all systems. *jive5ab* is fully compatible with 32- and 64-bit operating systems and can be compiled/run on 32- or 64-bit POSIX/Intel® based systems, e.g. Linux (Debian, RedHat, Ubuntu) or Mac OSX.

The *jive5ab* binary distribution will run on all systems[2] and auto-detects the hardware it is running on. Therefore this command set document will encompass all of the Mark5A, B, C and generic command sets. *jive5ab* will choose the appropriate one to support at runtime.

jive5ab implements the following DIM Command Set revisions: Mark5A v2.73, Mark5B v1.12 and Mark5C v1.0.

Note: sometimes *jive5ab* had to deviate from or had to choose one of multiple interpretations of commands found in the official Command Sets. All known deviations with respect to MIT Haystack software will be specifically documented and clearly marked in red.

Commands that are not part of the MIT Haystack Mark5A, 5B, or 5C command sets are documented like normal commands; highlighting them in red or any other visually distinctive way would be too distractive, given the number of them. Some commands were introduced only in later versions than 2.4.0. Where appropriate, these occurrences and their earliest appearance will be indicated.

We would like to thank Alan R. Whitney and Chester Ruszczyk from MIT Haystack observatory for making the original documentation source documents for the Mark5 command sets available - a real time- and life saver.

[1] http://www.haystack.mit.edu/tech/vlbi/mark5/index.html
[2] provided OS and Conduant SDK versions are compatible with where the binary was compiled on and linked with

2. Running the *jive5ab* program

The startup command-line for *jive5ab* is as follows:

```
$> jive5ab [-hbned6*] [-m <level>]  [-c <idx>] [-p <port>]
           [-f <fmt>]
```

where

`-h, --help`	help on startup parameters; other options are ignored if –h is present
`-m, --message-level <n>`	message level. Range 0 to *n* (positive integer), default 0. Higher *n* produces more detailed output. Useful level for operations is 0 – 3, a level greater than 3 is only useful for developers
`-c, --card <idx>`	card index of Streamstor card to use. Default is '1' which is appropriate for systems with only one Streamstor card installed. It is possible to use '-1' to have *jive5ab* skip driving the Streamstor card. This allows >1 *jive5ab* process to run on a Mark5 system. Use in conjuction with '-p' option
`-p, --port <port>`	TCP port number on which to listen for incoming commands. Default is 2620
`-b, -n –(no-)buffering`	do (*b*) or do not (*n*) copy recorded data to Mark5's main memory whilst recording ('buffering')
`-e, --echo`	even if the message level would indicate it, do not echo command/reply statements on the screen
`-d, --dual-bank`	start in dual bank mode
`-6, --mark6`	find Mark6 disk mountpoints by default in stead of FlexBuff ones (*jive5ab* > 2.6.0)
`-f, --format <fmt>`	set Mark6 (`fmt = mk6`) or FlexBuff (`fmt = flexbuff`) recording mode as default. Default is `flexbuff`. (*jive5ab* > 2.6.0)
`-B, --min-block-size <s>`	Override minimum block size for vbs/mk6 recordings (*jive5ab* ≥ 2.9.0)
`-*, --allow-root`	Do not drop privileges, continue running with root privileges if the program is being run `suid` root (*jive5ab* ≥ 3.0.0)
`-S, --sfxc-port <port>`	Start server for SFXC commands on port `<port>`

3. Notes on *jive5ab* command set

The following should be noted with respect to the command set:

1. All commands/queries are implemented using the VSI-S communications protocol and command/response syntax.
2. Commands/queries are case *insensitive*.
3. Versions of program *jive5ab* with a version earlier than the version mentioned in the header of this memo may not implement all commands indicated in this memo or, in some cases, may implement them in a different way (use 'version?' query to get the current *jive5ab* version - see 'System Queries and Responses'). Some commands only appeared in later versions of *jive5ab*, in which case it is documented what the earliest release supporting that command was.

4. VSI-S Command, Query and Response Syntax

The following explanation of the VSI-S syntax may be useful in understanding the structure of commands, queries and their respective responses. This explanation has been lifted directly from the VSI-S specification.

4.1 Command Syntax

Commands cause the system to take some action and are of the form

<keyword> = <field 1> : <field 2> : …. ;

where <keyword> is a VSI-S command keyword. The number of fields may either be fixed or indefinite; fields are separated by colons and terminated with a semi-colon. A field may be of type decimal integer, decimal real, integer hex, character, literal ASCII or a VSI-format time code. White space between tokens in the command line is ignored, however most character fields disallow embedded white space. For Field System compatibility, field length is limited to 32 characters except for the 'scan label' (see Section 6), which is limited to 64 characters.

4.2 Command-Response Syntax

Each command elicits a response of the form

!<keyword> = < return code > [:<DTS-specific return> :….] ;

where

<keyword> is the command keyword

<return code> is an ASCII integer as follows:

0 action successfully completed
1 action initiated or enabled, but not completed
2 command not implemented or not relevant to this DTS
3 syntax error

4 error encountered during attempt to execute
5 currently too busy to service request; try again later
6 inconsistent or conflicting request[3]
7 no such keyword
8 parameter error

<DTS-specific return> one or more optional fields specific to the particular DTS, following the standard fields defined by VSI-S; fields may be of any type, but should be informative about the details of the action or error.

4.3 *Query and Query-Response Syntax*

Queries return information about the system and are of the form

<keyword> ? <field 1> : <field 2> : …. ;

with a response of the form

!<keyword> ? <field 1(return code)> : <field 2> : <field 3> : …: [<DTS-specific return>];

where

<return code> is an ASCII integer as follows:

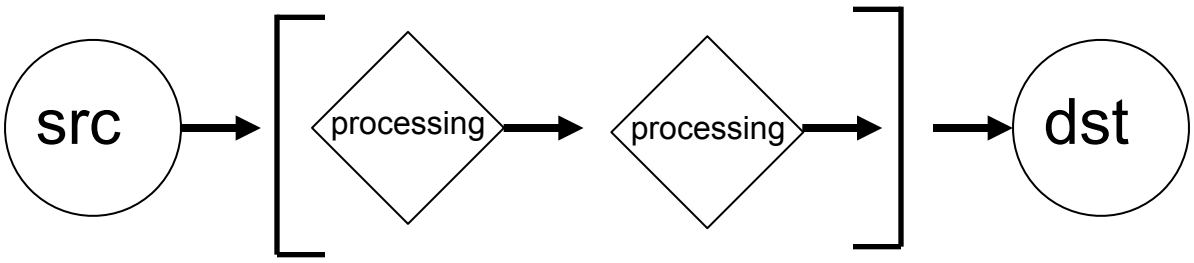
0 query successfully completed
1 action initiated or enabled, but not completed
2 query not implemented or not relevant to this DTS
3 syntax error
4 error encountered during attempt to execute query
5 currently too busy to service request; try again later
6 inconsistent or conflicting request
7 no such keyword
8 parameter error
9 indeterminate state

Note: *A 'blank' in a returned query field indicates the value of the parameter is unknown. A '?' in a returned query field indicates that not only is the parameter unknown, but that some sort of error condition likely exists.*

[3] For example, it is illegal to attempt to record during readback or position unloaded media.

5. *jive5ab* Data Transfer Modes explained

In *jive5ab's* internal architecture, a data transfer is a *processing chain*. Such an entity has a source, whence data comes from, and a destination, where the data leaves the system. In between these steps may be an arbitrary number of processing steps, processing the data as to the user's request. Graphically this can be represented as follows:



Each element in the processing chain is a reusable building block such that it can be used in different transfers. An example of *src* would be `file`, *dst* could be `net`, to form the `file2net` transfer. The same *src* `file` can also be used to form e.g. the `file2disk` transfer.

An example of a processing step is *channel dropping*. This allows the compression of the data stream by throwing away a set of bits out of each 64 bit word. The channel dropping step can be inserted into basically any transfer; `in2net` (used in real-time e-VLBI) is one example but it would work equally well in a `file2disk`.

The number of combinations that can be made with the existing set of sources, destinations and processing steps is already too large to make plots of all possibilities, refer to the list of commands further on: all transfer modes follow the naming scheme *src*`2`*dest*.

A list of current data transfer end points and/or data processing chains is found below, it may serve as guide of what to expect. Some transfer end points are named the same wether or not they are data source or data destination. Their position in the transfer name defines the actual nature of them.

in – read from the I/O board into the Streamstor card (formatter)

out – write from Streamstor card to the I/O board (correlator)

disk – read from or write to the Streamstor disk module

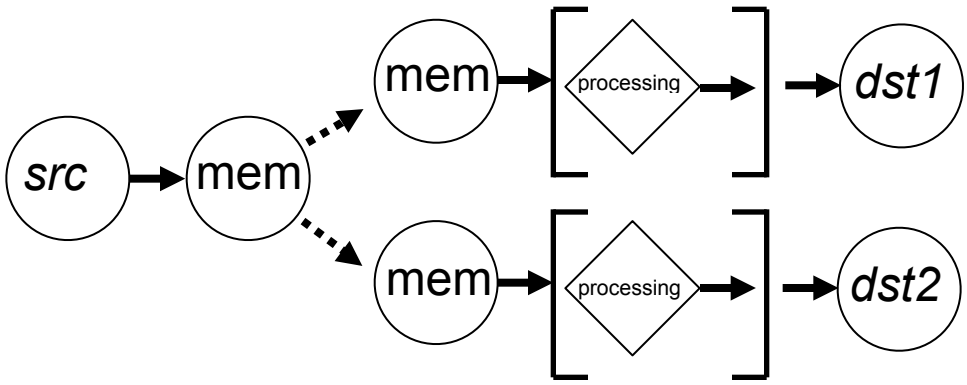
net – read from or write to the network; allow Mark5 to send/receive data to/from other machines

file – read from or write to a file on the filesystem of the host computer

jive5ab adds a number of non-standard transport end points besides these standard ones:

fill – produce a user defined fill pattern

mem – read or write to a circular buffer in the Mark5's main memory. It is worthwhile to note that this endpoint supports multiple readers. The data from the circular buffer can be independently read and processed by multiple transfers, provided enough resources to sustain the transfers are available. E.g. the data from this buffer could be written to a file and at the same time, independently, be sent to the network. This is graphically depicted below.



spill, spif, spin, spid, splet – these data sources combine a data source and corner turner processing step immediately following the receipt of data, producing legacy VDIF data (proper VDIF with EDV#0 for *jive5ab* ≥ 3.0.0). The following transfers read and split respectively from: *spill* = fill pattern (split fill = spill), *spif* = regular file from disk (split file = spif), *spin* = I/O board (split input = spin), *spid* = Streamstor disks (split disk = spid) and *splet* = network (split net = splet). Currently the only supported destinations are *file* and *net*.

fork - *jive5ab* exploits features offered by the Conduant Streamstor system. One of these is *forking*, or rather, *duplicating* of the data. The Streamstor Card has three I/O ports, connected to respectively the MIT Haystack I/O board, the disk modules and the PCI bus. The Streamstor Card can duplicate the data received on one of the ports to both of the remaining ports. Typically *fork* is used to store data to the disk modules and do something else in parallel, e.g. correlate (`net2fork` = `net2disk` + `net2out`) or send to network (`in2fork` = `in2disk`(aka `record`) + `in2net`).

6. Multiple independent Data Transfers

A feature of *jive5ab* is that can do multiple, independent, transfers at the same time, provided none of the endpoints are a unique resource. E.g. it is not possible to do `in2net` and `disk2file` at the same time because the I/O board + StreamStor combination can only be used in one direction at any instance of time. On the other hand multiple `file2net` and/or `net2file` can be run, next to e.g. a `record` or `disk2file`.

If one or more of the simultaneous transfers should be real-time, this can only be guaranteed provided sufficient resources (CPU, network, memory) are available to sustain them.

6.1 Implementation note — the concept of 'runtimes'

In order to support this, *jive5ab* has the notion of *runtimes*. Each *runtime* can do one data transfer. Initially, when *jive5ab* starts up, there is only one runtime present and all commands received by *jive5ab* are processed by this one; the default one.

runtimes are created on the fly, as you request them. Each *runtime* has a name and the first time a name is used, a new *runtime* with that name will be created. Due to historical reasons the default runtime's name is an un-informative "0" (zero)[4].

In order to send command(s) to a specific *runtime*, prefix the command(s) with the "`runtime = <name>;`" command. All commands on that line will be sent to (and processed by) the indicated *runtime*. When a transfer in a specific, non-default, runtime is done and is not expected to be reused it is best to delete it and its resources. Please see the `runtime` command.

To prevent concurrent access to the unique resource hardware ***only the default runtime has access to the Mark5 StreamStor hardware***. In other, non-default runtimes, commands like "`record=`" or "`fill2out=`" cannot be issued; an error code 7 ("no such keyword") is returned if such an attempt is made. Since *jive5ab* 2.6.0 `record=on` is valid in these runtimes and triggers FlexBuff/Mark6 network packet recording mode. Non-default runtimes on *all* supported systems have the same "`mode=`" command; they expose the "Mark5B/DOM/Mark5C/generic" version rather than the one intended for the current hardware. In fact all non-default runtimes only expose the "generic" command set, including `record=on` since *jive5ab* 2.6.0. On generic, non-Mark5 systems, the default runtime exposes the "generic" command set too.

6.2 Configuring (a) runtime(s)

It is important to realise that each runtime has its own set of mode, net_protocol and play_rate/clock_set values. It is not sufficient to rely on the FieldSystem to set the mode in the default runtime and expect things to work in other runtimes.

If a data format dependent transfer is to be done by a specific runtime it is necessary to configure the correct mode and play_rate or clock_set in that runtime. An example would be to do time stamp printing: this requires the data format and data rate to be specified for a successful decode.

[4] In hindsight a horrible choice. However, in practice the actual runtime name(s) should hardly ever be exposed to human users

If, on the other hand, a network transfer is to be done, the net_protocol should be set (in case the default is not acceptable).

Transfers like `in2net` and `disk2net` remember their last network destination for a subsequent "`=connect`" without argument. *jive5ab* extends this behaviour on a per runtime basis: the last used host/ip that was used in a "connect" in a runtime is stored. If a runtime is deleted, so is this information.

An example would be:

First time transferring a file to Bonn; note that all commands have to be given on one (1) line[5] or else they will be sent to/processed by the default runtime, not the 'xferToBonn' runtime:

```
runtime=xferToBonn; net_protocol=udt; file2net=connect : XX.YY.ZZ.UU : /path/to/file; file2net=on
```

For subsequent sending files to Bonn, the following line would suffice:

```
runtime=xferToBonn; file2net=connect : : /path/to/file2; file2net = on
```

6.3 Running a network transfer

In general all `*2net` + `net2*` transfer combinations follow the same mechanism for setting up, running and tearing down.

1. Set network parameters (`net_protocol`, `mtu`, `net_port`, `ipd`) to the desired values on both the source and destination host.
2. Issue `net2* = open : <...>` to the destination host to prepare that system for accepting incoming data and verify/configure the data sink.
3. Issue `*2net = connect : <...>` to the source host to verify/configure the data source and open the data channel.
4. Now the data flow may be started by issuing a `*2net = on [ : .... ]`. Some transfers accept options after the "on" (e.g. start byte/end byte numbers), refer to the specific command documentation for detailed information.
5. Some transfers allow pausing the transfer. Currently they are `in2net`, `in2file`, `in2fork`.
6. To stop the transfer properly, issue `net2* = close` to the recipient of the data stream and `*2net = disconnect` to the sender. The order is unimportant; *jive5ab* has no particularly strong feelings about this. In case it does you should inform the author forthwith.
7. To comply with Mark5A/DIMino standards, the transfers `disk2file`, `disk2net` and `file2disk` may also be terminated by issuing a "`reset = abort;`" command. Note that of those three only `disk2net` really resides under this section.

[5] an alternative would be to send each command separately, prefixing each command with "`runtime=xferToBonn; `"

7. Data format support across the range of systems

The Mark5 series of data recorders all have a 'native' data format. The following table lists *jive5ab* support and according to which reference the support for it has been coded.

Recorder	Format	Comments
Mark5A	Mark4, VLBA	Mark4 Memo 230 (rev 1.21)
Mark5B	Mark5B	Mark5 Memo 019 +`0xABADDEED` syncword +*jive5ab* 4 Gbps extension
Mark5C	VDIF, if any	http://www.vlbi.org/vdif/ (version 1.0)

Traditionally, on the Mark5A and B the supported data format is strictly tied to the firmware on the MIT Haystack I/O board installed. Mark5C does not do formatting but doesn't "support" all formats either. In the official DIM implementation, it is impossible to configure a non-Mark5B data format and in order to support Mark5B data on a Mark5A system an updated I/O board firmware needs to be installed (Mark5A+). *jive5ab* supports the Mark5B format up to 4 Gbps. Neither of the systems support VDIF (bar 5C) natively.

jive5ab supports all of the listed formats on all of the systems on the basis that for data transfers not involving the I/O board firmware, it should not matter what the format is.

Then there is the fact that for some transfers and/or processing the data format *must* be known whilst for others it can be totally opaque. For the latter cases the special format '*none*' has been introduced.

In general it is good practice to always configure the data format if it is known which data format is to be expected, be it from the formatter, the network or elsewhere.

Under the following circumstances the software can certainly not successfully operate without knowing the exact details of the incoming data format:

1. transfers which make use of corner turning; to be able to correctly decode the incoming data frames into the individual channels
2. printing the time stamps or verifying the incoming data format[6]
3. if the remaining time on a recording medium is to be correctly computed

The specification of the data format allows *jive5ab* to size its buffers and network packets (for UDP based transfers) appropriately and set up correct time stamp decoding. On the other hand, for a blind transfer of data from machine A to machine B[7] using a reliable protocol (e.g. TCP) the buffer- and packet sizes are completely irrelevant, and by extension the data format itself.

[6] d'oh!

[7] e.g. `file2net` on machine A and `net2disk` on machine B

7.1 Configuring data formats

To allow configuring non-native formats the "`mode=`" command on the various systems has been enriched. When configuring a data format the MIT Haystack I/O board should not support the hardware is left untouched and only *jive5ab*'s internal idea of the current format is updated.

On Mark5B/DOM, Mark5C and generic systems both Mark5A and Mark5B 'flavours' of setting the data format and data rate are supported – `mode`+`play_rate` (for MarkIV and VLBA data formats) as well as `mode`+`clock_set` for Mark5B formats.

In Walter Brisken's mark5access[8], currently part of the DiFX[9] source code tree, a canonical data format designation is used to describe the important properties of the data. *jive5ab* supports this format as well. With a single argument to the "`mode=`" command, the data format and track bit rate are set. This so-called 'magic mode' configuration persists until a hardware style "`mode=`", "`play_rate=`" or "`clock_set=`" command is issued to the system.

The canonical format is a single, case-insensitive, ASCII string, formatted as follows:

```
<format>[_<frame size>]-<rate>-<channels>-<bits>[/<decimation>]
```

where

<format> is the data format, with supported values:

- `Mark5B`
- `VDIF`
- `VDIFL` (VDIF w/ legacy headers)
- `VLBA`*`n_m`* with *`n,m`* integers
- `MKIV`*`n_m`* ,,

Note: the *`n_m`* for `VLBA` and `MKIV` formats are the fan mode. `1_4` means 1:4 fan-out and `2_1` means 2:1 fan-in. In practice, fan-in is not supported but the values of *`n`* and *`m`* are always used to compute the *actual* number of tracks used in the recording, in order to compute the track bit rate, which is necessary for correctly de- or encoding time stamps.

`_<frame size>` is the `VDIF` *payload* size (the Data Array size in the VDIF standard), thus excluding the header. This parameter **must** be supplied for VDIF and VDIFL and **may not** be supplied for any of the other formats.

`<rate>` is the total data rate in Mbps, excluding framing

`<channels>` is the number of baseband channels, typically 2^n

`<bits>` is the number of bits per sample

`<decimation>` (optional) the decimation that was applied. Note that *jive5ab* does support specifying it but ignores its value completely.

Examples:

`MKIV1_4-512-8-2` for 512 Mbps, 16 channels of 2 bit/sample MarkIV data

`VDIFL_8192-4096-32-2` for 4 Gbps legacy VDIF with 32 channels of 2 bit/sample, sent in UDP packets of 8192 bytes data (excluding 16 bytes of legacy VDIF header)

---

[8] https://github.com/demorest/mark5access (although this seems an old version)
[9] http://cira.ivec.org/dokuwiki/doku.php/difx/documentation

8. channel dropping and corner turning

jive5ab adds two generic data processing steps that can be inserted into practically any data transfer: channel dropping and corner turning.

Both can operate under real-time constraints provided enough processing resources (CPU, memory, network bandwidth) are available.

8.1 Channel dropping

Channel dropping is the technique by which a VLBI data stream can be compressed by selectively throwing away bits. E.g. all the bits carrying the data from one (or more) channel(s) could be marked as unwanted. The unwanted bits are replaced by bits from other channels such that the size of the total data frame is reduced to a data rate fitting over the network link. Channel dropping was developed for real-time e-VLBI observations but can apply to other transfers as well, though better ways exist (see corner turning).

Channel dropping technology allows *jive5ab* to send 31 out of the 32 channels of a 1024 Mbps observation (i.e. 990Mbps) over a 1Gbps ethernet link (=1000Mbps) rather than having to step down to 512Mbps observing mode and thus having close to 96% of the 1024 Mbps sensitivity rather than 70%.

Channel dropping is indicated to the system using the "`trackmask=`" command. Setting the track mask to a non-zero value indicates that channel dropping is to be inserted in the data processing chain. The argument is a 64-bit hexadecimal value. A '1' in a bit position means that that bit is to be kept. Bits having '0' will be discarded. Disable channel dropping by setting the track mask to 0 (zero).

As an example this feature can be used to make 1-bit data from 2-bit data by throwing away alternate bits, effectively halving the data, resulting in 'only' a 29% signal-to-noise degradation. For best results the magnitude bits should be thrown away. Depending on your actual setup this will either be:

`trackmask = 0x5555555555555555` or

`trackmask = 0xaaaaaaaaaaaaaaaa`

Note the 64 bit hexadecimal values. If less than 64 bits of track mask are provided, the lower bits of the 64 bits track mask will be set to this value and the higher order bits will be zeroed.

If two systems are doing a compressed network transfer (e.g. `in2net` + `net2out`) the track mask, data format and network parameters *must* be configured on both systems and, for maximum success, be set to identical values!

Those transfers honouring the channel dropping processing automatically know wether to insert a compression or decompression step.

The "`trackmask=`" command always returns '1' ("initiated but not completed yet") because in the background the (de)compression algorithms are being computed and their C-code implementation generated, compiled and dynamically loaded into the binary. The "`trackmask?`" query will return '1' as long as this process is running. When "`trackmask?`" returns '0' it is safe to use the channel dropping transfer.

At this moment, a channel dropping processing step is automatically inserted in certain transfers if the track mask ≠ 0. To wit, a compression step is inserted in the following transfers:

> `disk2net`, `file2net`, `fill2net`, `disk2file`, `fill2file`, `in2net`, `in2mem`, `in2file`, `in2fork`, `in2memfork`, `mem2net`, `disk2net`, `file2net`, `fill2net`

and a decompression step is automatically inserted in those:

> `net2out`, `net2disk`, `net2fork`, `net2file`, `net2mem`, `net2check`, `net2sfxc`, `file2check`,

8.2 Corner turning

Corner turning, sometimes called de-channelizing, is the processing by which the samples from the VLBI data frames are re-arranged into separate buffers such that each buffer contains a time series of one individual channel.

This has to do with the traditional recording format of VLBI data: typically the bits are arranged in sampling order. The samples for all channels for a particular time, followed by the samples for the next sample time. This implies that the next sample from a channel is found at a different memory address. After corner turning the samples of a single channel are found in a time series, adjacent to each other. This is the preferred format for correlators.

After corner turning the channel buffers are translated into legacy VDIF format. Each channel will be carried in its own VDIF data thread. *jive5ab* allows each data thread to be sent to a different network location or file on disk (including discarding it).

Corner turning/de-channelizing can be used for multiple purposes: bringing down the real-time data rate, distribute one incoming data stream over multiple destinations or convert any of the supported data formats into VDIF format.

In order to make the most use of corner turning an intimate knowledge of the actual layout of the bits in the data format is necessary. Please refer to the table in Section 6 for the supported data format definitions.

jive5ab contains hand-crafted assembly code (32- and 64-bit versions) designed to run on Intel®/ AMD® processors supporting the SSE2 (or higher) instruction set to fully exploit the CPU's potential. SSE2 offers instructions which process 128 bits at a time.

Due to limitations in the CPU registers (mostly the amount of them, in 32-bit mode) the corner turning routines that are most efficient are those who de-channelize a maximum of 8 channels of data.

Fortunately, looking at the data format memos mentioned in section 6, typically, 16 and 32 channel observations are, as seen from a bit-stream layout pattern, nothing more than an 8-channel layout repeated two or four times. As such, a 16 channel de-channelizer routine could first split the data into 2x 16-bit 'chunks', each containing 8 channels of data. Each of the two halves can subsequently be efficiently de-channelized using the 8-channel de-channelizer.

jive5ab supports daisy-chaining of splitting operations to allow the user to build efficient de-channelizer / chunker chains.

There is also the possibility of not splitting at all and thus the data frames will be just re-labelled with a legacy VDIF header. For none of the data formats this yields *valid* VDIF data though! Even for Mark5B data in standard astro mode conversion to VDIF is not this simple because the order of the sign- and magnitude bits is reversed between the two formats. Within *jive5ab*'s built-in de-channelizing routines exists an efficient sign-magnitude swapper[10] which can be inserted into the de-channelizing chain.

The basic operation of the corner turner follows this pattern:

- receive a data frame, tagged with stream label *X* from a previous step
- break the frame up into a number of pieces, say *N*, each *1/N*'th in size of the input frame and relabel them as stream *X*N* through *X*N+(N-1)*. This is done to be able to keep track of all the individual channels as they flow through the chain. This step may do complex bit-level movements to group as many bits of each channel together as is possible before appending the samples into that channel's specific buffer
- accumulate these frames for a (configurable) amount of input frames
- output the *N* new frames to the next step and start a new accumulation

The first step in the chain just reads data frames from the selected medium and tags them with stream 0. These frames are handed off to the first step in the corner turning chain. The last but one step will divide the output up into VDIF frames of the configured size, set the VDIF *thread-id* to *X*. The final step will write these VDIF frames to either the network, a file or discard them altogether.

The output VDIF streams will be labelled *0* .. *n*. This is important because the corner turning routines do not know anything about the observation; they strictly deal with bits and bytes and leave all interpretation and logic to the user.

The `spill2*`, `spin2*`, `spif2*`, `spid2*`, `splet2*` functions drive the corner turning capabilities[11]. The most important parameters to these commands are the corner turning chain and what to do with the output(s) of the corner turning engine.

In general the commands look like this:

```
sp*2* = … : <corner turning chain> : <outputX> = <dstY> [ : <outputZ> = <dstA>]
```

Depending on the actual data source, the … may contain a `connect` or a `file name` or other specifics and will not be discussed here.

In section **8.2.1** the `<corner turning chain>` will be described whilst the `<outputX> = <dstY>` will be covered in **8.2.2**.

[10] again not a real splitter, unless the definition of 'splitting' is expanded to support 1:1 splitting (which, in *jive5ab*, it is)

[11] See Section 5 for the origin of the names of these routines

8.2.1 The <corner turning chain>

With powerful things comes daunting syntax. This is a single field which follows a certain grammar to allow one to configure the goriest details of the corner turning engine. For clarity it is probably best to represent the syntax in an Extended Backus-Naur Form (EBNF)[12] like format. Double quotes indicate string literals, entries in curly braces ("{ … }") mean they may be repeated any number of times (including zero times).

```
chain                   =  step , {"+" step}

step                    =  built_in_step | dynamic_step ,
                           {"*" , n_accumulate }

built_in_step           =  "8bitx4" | "16bitx2" | "16bitx4" | "32bitx2" |
                           "swap_sign_mag" | "2Ch2bit1to2" | "4Ch2bit1to2"
                           | "8Ch2bit1to2_hv" | "8Ch2bit_hv" |
                           "16Ch2bit1to2_hv"

dynamic_step            =  n_inputbit , ">" , channels

channels                =  channel_def , { channel_def }

channel_def             =  "[" channel_bits "]"

channel_bits            =  bit_index , { "," , bit_index }

n_accumulate,           =  integer
bit_index, n_inputbit
```

In words this sais, sort of:

> A corner turning setup is formed by at least one corner turning step. Multiple steps can be added together ("+") to form a chain. Each step can either be a built-in step or a dynamically-defined step. Optionally, by "multiplying" a step by an integer it is possible to configure how many input frames the current step must accumulate[13]. A built-in step is just named by a literal string (see above). The more advanced dynamic-step has a sub-format. The code must know the input data word width and a description of which bits to take and where to place them to generate output channels. An ouput channel is defined using a comma-separated list of source bit index/indices inside square brackets ("[…]"). The amount of bracketed bit-lists is the number of channels output by this dynamic splitter.

Note that the dynamic-step is not real-time guaranteed. What it does is, based upon your bit-extraction specification, generate C-source code for an extraction function. This code will be compiled and dynamically loaded back into the program. Some optimization takes place but it remains C-code in stead of Assembler/SSE2 instructions. This corner turner's throughput will vary upon data rate and system performance, obviously.

A few examples of corner turning chains may be illustrative:

---

[12] https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form

[13] the default is the number of pieces this step breaks up the input frame in. E.g. "8bitx4" breaks a frame into four pieces and thus will accumulate four input frames before creating output

Q: Corner turn 1024 Mbps Mark4 data into individual channels.

A: This mode has 16 channels at 2 bit per sample: each channel consisting of a sign and a magnitude bit, so 32 bit streams. Typically these are recorded over 64 bit streams for 16 Mbps/track, thus fanout 1:2. There exists a built-in corner turner for 8 channels, 2bit fanout 1 to 2 ("8Ch2bit1to2_hv" - see above). This one processes 32 bit streams into 8 channels of output.

It turns out (see MarkIV Memo 230.1) that the 16 channels are recorded as 2x an 8 channel setup. So for corner turning the 16 channel data, we can first split the data frame in 2x 32 bits (also a built-in splitter!) and then use the built-in 8 channel splitter. So we end up with a corner turning chain like this:

sp*2* = … : 32bitx2 + 8Ch2bit1to2_hv : ….

Q: Break 4 Gbps Mark5B data up into 4x 1 Gbps streams

A: This is an easy one! Mark5B format is very simple! The sign and magnitude bits are situated next to each other in pairs. There is no fan-in or fan-out. 4 Gbps Mark5B data is generated as 32 channels of 2bit sampled data, for a total of 64 bit streams.

We can just take 16 bits at a time because that groups together 8 channels of data. There exists a built-in 16bit splitter which we'll use.

There is one catch with this. The sign- and magnitude bits in Mark5B data are reversed in order compared to what VDIF (which is the format we'll output) dictates. There is a highly efficient built-in sign/magnitude swapper which does not split but just swaps all the bits in its input.

We end up with the following corner turning chain:

sp*2* = … : swap_sign_mag + 16bitx4 : … or

sp*2* = … : 16bitx4 + swap_sign_mag : …

because it does not matter when we swap the sign/magnitude bits (logically). For the CPU the first might be more efficient than the latter but that test is left as an excercise to the reader.

Q: I only want to extract some specific bits from my data

A: That will be the dynamic splitter then. Let's assume 16 bit streams are recorded. Each time sample is therefore 16 bits wide. Also assume that it contains data from 8 channels, two bits for each channel. So every 16 bits contain one 2-bit sample of each channel[14]. For example two things are wanted: make 1-bit data as well as extract only two channels from the eight. Assume Mark5B layout: sample bits are recorded pairwise next to each other. Then the following would extract one bit each from channel 3 and 5 and produce two VDIF threads (channels) as output:

sp*2* = … : 16 > [6] [10] : …

Now VDIF thread#0 contains bits 6, 22, 38, …; thread#1 will contain bits 10, 26, 42 etc (each 16th bit, starting from 6 and 10). It is also possible to duplicate, reorder or rearrange bits at will:

sp*2* = … : 16 > [6,6] [1,0] [3,8] : …

[14] you were warned: for this one really intimate knowledge of the data format is needed!

8.2.2 Configuration of corner turner output: `<outputX> = <dstY>`

The output of the corner turner is *N* threads of legacy VDIF (since *jive5ab* 3.0.0 normal VDIF, EDV version 0) numbered with thread-id (or 'tag') *0* .. *N-1*, where *N* is obviously defined by the actual corner turning chain setup (see previous section, **8.2.1**).

Using the `<outputX> = <dstY>` syntax a (sub)set of VDIF threads can be sent to destination `Y`. `OutputX` may be a single thread-id, a comma-separated list of individual VDIF thread-ids or a range of thread-ids. `DstY` depends on the actual transfer type - either a `HOST[@PORT]` for the `*2net` transfers or a filename for the `*2file` transfers.

Note that the `[@PORT]` part of a network destination is optional; in case it is left out, the data port value set using net_protocol is used[15]. The '`@`' symbol was used to separate host and port because the '`:`' is already in use as the VSI/S parameter-separation character.

It is perfectly legal to over- or under specify the outputs; it is absolutely not necessary to mention exactly all output thread-ids.

> If the output section sees a VDIF thread-id that has no associated destination, it will be silently ignored. This can usefully be exploited for extracting only channels with useful data or limiting data rate.
>
> If the outputs are over-specified - more VDIF thread-ids have destinations than actual outputs generated by the corner turner, that's harmless. The non-produced VDIF thread-ids will not end up in the output on the premise that, in fact, these thread-ids are not produced.

There may be any number of `<outputX> = <dstY>` configured.

Examples:

Ensure all streams go to the same network destination:

```
sp*2net = … : 0-1024 = sfxc.jive.nl@46227
```

Write even thread-ids in one file, the odd ones in another:

```
sp*2file = … : 0,2,4,6,8 = /path/to/even.vdif
             : 1,3,5,7,9 = /path/to/odd.vdif
```

For your convenience, `<dstY>` may be a name 'template'. This happens when `<dstY>` contains the special string "`{tag}`". Each occurrence of this this pattern in `<dstY>` will be replaced by the number of the VDIF thread-id that will be written to that `<dstY>`:

```
sp*2file = … : 0,2,4,6,8 = /path/to/thread{tag}.vdif
```

would open file "/path/to/thread0.vdif" and only store VDIF frames of thread #0 in there. Likewise for threads 2, 4, 6 and 8.

---

[15] if `splet2net` (read from network - corner turn - output to network) is used, the output network parameters have to be set differently. The standard network configuration will define the input setup of the corner turner.

9. Comments on Record-, Start-scan- and Stop-scan Pointer

Three different pointers are maintained by the Mark5 system and it is important to understand what they are, what they mean, and how they are managed. The *record pointer* is associated only with recording data to the disks; the *start-scan* and *stop-scan pointer*s are used to control reading data from the disks.

9.1 Record pointer

The Mark5 system records data to a disk set much as if it were a tape. That is, recording starts from the beginning and gradually fills the disk set as scans are recorded one after another. The 'record pointer' indicates the current recording position (in bytes, always a multiple of 8) which, at any instant, is just the current total number of recorded bytes. Arbitrary recorded scans cannot be erased; however, individual scans may be erased in order from last to first. The entire disk set is erased by setting the record pointer back to zero using the 'reset=erase' command. Table 1 lists commands that modify the record pointer; Table 2 lists commands that are affected by the record pointer.

Command	Comment
reset=erase	Forces record pointer to zero.
reset=erase_last_scan	Sets record pointer to beginning of the disk space occupied by the last scan (effectively erases the last scan).
reset=condition	Effectively erase disk pack, full write + read cycle over the whole disk pack
record=on	Starts writing at current value of record pointer; advances record pointer as data are recorded.
file2disk/fill2disk	Data transfer from Linux disk to Mark5: Starts writing to Mark5 disks at current value of record pointer; record pointer advances as data are written.
net2disk	Data transfer from network to Mark5: Starts writing to Mark5 disks at current value of record pointer; record pointer advances as data are written.

Table 1: Commands that modify the record pointer

Command	Comment
record=on	Starts writing to Mark5 disks at record pointer; increments record pointer as recording progresses.
file2disk	Starts transfer to Mark5 disks at record pointer; increments record pointer as data transfer progresses.
net2disk	Starts transfer to Mark5 disks at record pointer; increments record pointer as data transfer progresses.

Table 2: Commands affected by the record pointer

The current value of the record pointer can be queried with the 'pointers?' query.

9.2 Start-scan and Stop-scan pointers

The 'start-*scan*' and 'stop-scan' pointers specify the start and end points for reading all or part of pre-recorded scan for data checking or data-transfer purposes. By default, these pointers are normally set to the beginning and end of a block of continuously recorded data, but for special purposes may be set to include only a portion of the recorded scan. Table 3 lists commands that modify these pointers; Table 4 lists commands that are affected by these pointers.

Command	**Comment**
reset=erase	Resets start-scan and stop-scan pointers to zero.
reset=condition	Resets start-scan and stop-scan pointers to zero.
reset=erase_last_scan	Sets start-scan pointer to beginning of 'new' last scan; sets stop-scan pointer to end of 'new' last scan.
record=off	Sets start-scan pointer to beginning of scan just recorded; sets stop-scan pointer to end of scan just recorded.
fill2disk	Sets start-scan pointer to beginning of scan just recorded; sets stop-scan pointer to end of scan just recorded.
fill2vbs	Sets start-scan pointer to beginning of scan just recorded; sets stop-scan pointer to end of scan just recorded.
file2disk	Sets start-scan pointer to beginning of scan just transferred; sets stop-scan pointer to end of scan just transferred.
net2disk	Sets start-scan pointer to beginning of scan just transferred; sets stop-scan pointer to end of scan just transferred. Since *jive5ab* ≥ 2.8
scan_set	Sets start-scan and stop-scan pointers to a data range within a scan as specified.

Table 3: Commands that modify the start-scan and stop-scan pointers

Command	**Comment**
data_check	Reads and checks a small amount of data beginning at start-scan pointer
scan_check	Checks small amount just after start-scan pointer and before end pointer.
disk2file	Unless specific start/stop byte numbers are specified, transfers data between start-scan and stop-scan pointers
disk2net	Unless specific start/stop byte numbers are specified, transfers data between start-scan and stop-scan pointers

Table 4: Commands affected by the start-scan and stop-scan pointers

A 'scan_set?' or 'pointers?' query returns information about the current value of the start-scan and stop-scan pointers.

10. Scan names, Scan Labels and Linux filenames

Mark5 defines a 'scan' as a continuously recorded set of data. Each scan is identified by a scan name, experiment name and station code, which are normally derived from the information in the associated VEX file used in the scheduling of the experiment (see http://lupus.gsfc.nasa.gov/vex/vex.html). An attempt to record a scan with a duplicate scan name on the same disk module will cause a trailing alphabetical character ('a-z', then 'A-Z') to be automatically appended to the scan name. If there are more than 52 scans with same user-specified name, the suffix sequence will repeat. Information about the experiment name, station code, bit-stream mask, and sample rate are stored in the associated directory entry.

A scan label is defined as the character string

<exp name>_<stn code>_<scan name>

where

<exp name> is the name of the experiment (e.g. 'grf103'); maximum 8 characters, but by convention corresponds to a standardized 6-character experiment name. If null, will be replaced with 'EXP'.

<stn code> is the station code (e.g. 'ef'); maximum 8 characters, but by convention corresponds to standardized 2-character codes. If null, will be replaced with 'STN'

<scan name> is the identifier for the scan (e.g. '254-1056'), usually assigned by the observation-scheduling program; max 31 characters, though may be augmented to 32 characters by automatically generated duplicate-breaking suffix character.

Maximum scan-label length, including embedded underscores and possible scan-name suffix character, is 50 characters. <experiment name>, <station code> and <scan name> may contain only standard alpha-numeric characters, except '+', '-' and '.' characters may also be used in <scan name>. All fields are case sensitive. No white space is allowed in any of these subfields. Lower-case characters in all subfields are preferred. An example scan label is:

grf103_ef_scan001

When a Mark5B scan (or portion of a scan) is copied to a Linux file with *disk2file*, a Linux filename compatible with the internationally agreed e-VLBI filenaming convention (reference http://www.haystack.edu/tech/vlbi/evlbi/memo.html memo #49) is assigned as

'<scan label>_bm=<bit-stream mask>.m5b'
(example: 'grf103_ef_scan001_bm=0x0000ffff.m5b')

Linux files to be transferred to a Mark5B disk via the '*file2disk'* should have filenames corresponding to the standardized format described above so that the associated Mark5B directory entries can be properly filled.

Note: The <scan name> is equivalent to what is called <scan_ID> in VEX files, except the set of legal characters in <scan name> is more restricted and must be observed.

11. Non-bank mode operation

(*jive5ab* ≥ 2.8, significant rewrite from orig doc.)

jive5ab always had a very basic support for non-bank mode. *jive5ab* ≥ 2.8 introduces "working support for non-bank mode of the Mark5 system" for some arbitrary value of "working". However, it is recommended, if possible, a FlexBuff or Mark6 recorder be used for > 2048 Mbps recordings instead:

Tests conducted at JIVE and HartRAO have shown the firmware being rather feeble in non-bank mode and may result in not-working, lock up, or even crashing the operating system kernel, **specifically when de-activing disk packs!**

Handled with kid gloves some mileage may yet be gotten out of the system - do read the section "**11.3 Rules for non-bank mode operation"** below carefully. They differ from the original Mark5C documentation.

11.1 Introduction

The normal operation of the Mark5 is in so-called 'bank' mode where only one disk module is active at any given time; bank mode operation is adequate for data rates up to 1024 Mbps, or 2048 Mbps with a SATA disk pack.

The StreamStor firmware allows for running the system in so-called "non-bank" mode: two disk packs operating as a single logical module, doubling the sustained write speed. *jive5ab* enables usage of this feature on all of the Mark5 platforms.

Given the Mark5B+'s single VSI port limit of 2048 Mbps, this "non-bank" configuration is mostly useful for Mark5C systems where data can be recorded from the 10 Gbps ethernet interface which does not have this limitation.

11.2 Notes on firmware limitation from an application perspective

When disk packs are erased, *then and only then* the firmware records on the disk pack(s) in which mode the disk pack(s) should be subsequently used: bank or non-bank mode. This depends on in which mode the StreamStor card is operating at the time of execution of the erase command.

One of the biggest issues with bank versus non-bank mode operation is that the firmware does not allow the application (*DIMino* or `jive5ab`) to query in which mode a disk pack was erased, i.e. the application cannot enquire how a disk pack *should* be used.

This leaves the application at the mercy of the firmware in combination with the operator to rely on both 'doing the right thing' as it is impossible to detect an attempt to use a disk pack in a mode it was not erased in.

The only thing the application can do is to try an operation and hope for the best. Unfortunately, sometimes 'the best' translates to any/all of: lock-up, hanging or crashing the O/S.

11.3 Rules for non-bank mode operation:

The major key to reliable operation in non-bank mode is to **always insert both disk packs in their original bank position they were in** at the time of creation of the non-bank mode pack. *jive5ab* can not enforce this because it cannot read the stickered barcode labels; all disk packs of a non-bank mode pack get the same (software) VSN. A `bank_set?` query when *jive5ab* is operating in non-bank mode returns the positions of the original VSNs at the time of creation. The stickered barcode labels on the disk packs should be used for visual identification.

In general: attempting any data access, read or write, when the green ready LED on a disk bay is blinking, when non-bank disk packs are inserted in the wrong order, or if a disk pack is used in a mode it was not erased in, may or may not work and may or may not confuse the firmware (necessitating SSReset) and may or may not crash your operating system kernel.

1. *jive5ab* must be running in non-bank mode, which can be achieved by:
    a. restarting `jive5ab` with the appropriate command line option (see Section 2)
    b. using the `personality= : [non]bank;` command to dynamically switch mode on any Mark5 system (requires *jive5ab* ≥ 2.8)
2. A non-bank module is created by inserting two disk packs, activating both, and sending `protect=off; reset=erase;` to *jive5ab* running in non-bank mode.
3. After the `reset=erase;` has completed, the user directory will have recorded in it the VSNs of the constituent disk packs the non-bank module was created of. Both disk packs now have the same VSN: the VSN of the module in bank A. This is a feature of the firmware combined with `reset=erase;` preserving the VSN of the module being erased.
4. We suggest assigning the newly created non-bank pack a different or special VSN, such that whenever the VSN of either of the modules is read one knows *a.)* this disk pack is part of a non-bank module and when `bank_set?` is queried: *b.)* what the constituent physical modules are and *c.)* in which order they need to be inserted. *jive5ab* ***must*** be operating in non-bank mode for `bank_set?` to return this information!
5. If only a single module of a non-bank module pair is ready, no operations involving recording or reading data are permitted, see the disclaimer above.
6. ***When deactivating a non-bank mode module pair, be sure to deactivate bank A before deactivating bank B. If you deactivate bank B first, then, depending on how slowly you deactivate bank A, your StreamStor may still work, lock up completely or crash the operating system. Hint: don't. Thus: deactivate A first, then B.***
7. A single disk pack can be returned to normal bank-mode operation by issuing a `reset=erase;` on the activated and selected bank whilst *jive5ab* is operating in bank mode. The module's original VSN will have to be restored manually.
8. Any attempts to use a bank-mode disk pack in non-bank mode or vice versa may or may not result in working, crashing or lock up. The firmware cannot inform *jive5ab* in which mode a disk pack was erased so there's no way to prevent this.

12. FlexBuff/Mark6 operations

jive5ab can turn any generic computer which contains a network interface and separate harddisks into a high-speed packet recorder. The model is that individual packets are grabbed from the network, collected in large-ish blocks in memory, and then scattered over the available/ configured mount points. On such systems, a `record=on:…` command will trigger the grab-packets-and-scatter-to-disk mode, using the current network parameters and selected mount points.

There exists software which will help in dealing with scattered data, from collecting these scattered blocks and presenting them as a single file to the operating system to listing/removing recordings[16].

The software does not care about the location of the mount points of your hard disks. However, for convenience, the FlexBuff and Mark6 layouts are easily supported. The `set_disks=` command exists to manage the mount points that *jive5ab* scatters the data over.

12.1 Mount points

It is important to realize the only real differences between FlexBuff and Mark6 are (1) Mark6 has removable disk packs and (2) the choice of mount points for the external disks. The table below shows the assumed mount point patterns for the both systems:

FlexBuff	**Mark6**
/mnt/diskNNN	/mnt/disks/MODULE/DISK/data

Where `NNN` is any number and `MODULE`, `DISK` are the Mark6 module slots 1..4 and 0..7 for up to eight disks for four modules slots. *jive5ab*'s compiled in default is to, at start-up, look for FlexBuff mount points and add all of those to the set of mount points to record on. A command line option (see Section 2) to make *jive5ab* scan for Mark6 disk packs instead.

The set of disks scanned at start up is *not* refreshed automatically. As a result, if, whilst *jive5ab* is running, disks are added, they will *not* be automatically picked up and used; disappeared disks will be skipped automatically. At runtime, the `set_disks=` command can be issued at any time — as long as no recording is happening — to change (including refresh) the set of current mount points.

12.2 The key to high-speed packet recording

If the target observing data rate exceeds ~ 1 Gbps, there is no way about it but that the operating system must be tuned. Without tinkering with the operating system's parameters, packet loss at

[16] The **vbs** utilities as described (+download) `http://www.jive.eu/~verkout/flexbuff/README.vbs`

high-ish incoming data rates is unavoidable. In this section the tuning parameters will be focussed on servers running the Linux operating system, but the principles apply generally.

Packet loss at the recorder seems to be solely driven by non-locality of data in combination with conservative default buffer sizes set in the operating system.
All tuning actions' goals are towards "***ensuring that packets received on a network card are completely handled on the CPU to which the network card is 'connected'***". Note the distinction between **CPU** and **CPU core**. These days, one CPU has multiple/many cores.

As there is no single standard server hardware, the following should be regarded as guidelines and are prone to manual translation to the specific hardware being tuned.

In general, the tuning encompasses the following steps, which are only broadly described here. All tunings, useful values and how to set them, and their rationale are explained in depth in the (consisting of > 95% comment) text-file-cum-almost-script `flexbuf.recording.txt`[17].

12.2.1 Tuning of the Operating System, fix application to certain CPU core(s)

Use of `sysctl(8)` to enlarge the very small default Linux UDP buffer sizes, which accounts for ~ 50% of the trick.

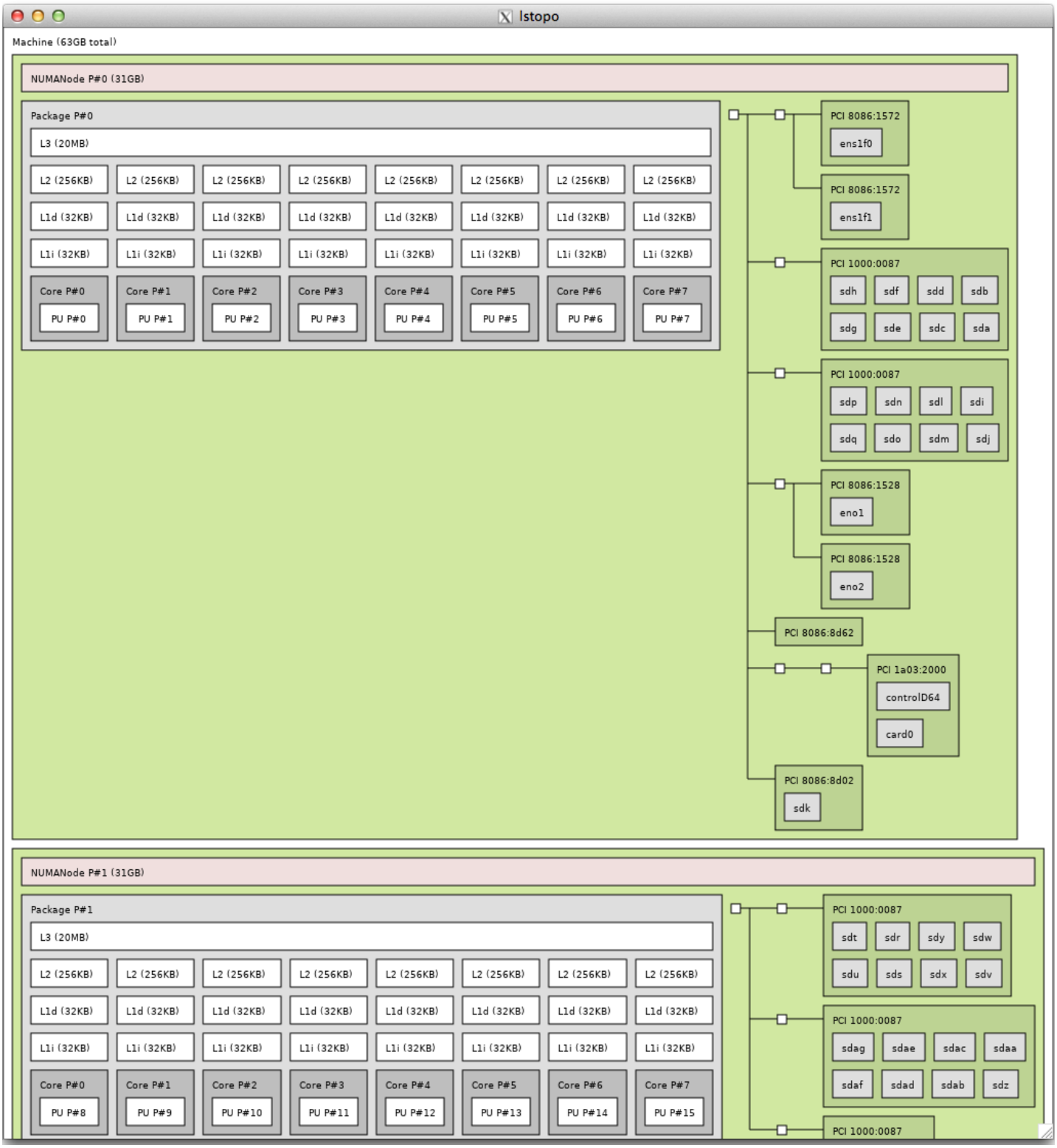
Disable hyperthreading, otherwise the operating system might schedule I/O tasks on hyperthreaded cores, which actually degrades I/O performance substantially.

Shaping interrupt request (IRQ) handling accounts for the rest of the solution. Linux comes configured to deal efficiently with a very diverse workload. In such a situation, dividing the IRQ load over all available CPU cores helps responsiveness of the system, even under heavy use. For our high-speed packet capture it is disastrous.

It is essential that all interrupts for one ethernet or mpt3sas (hard disk controller) card are processed by one CPU core. The CPU core to choose depends on how the hardware is connected in the system. Typcially, each CPU in a multi-CPU system, is connected to a specific bus. A CPU *can* access data from devices not connected directly *but* the bandwidth is limited to the inter-CPU bandwidth, which is typically smaller than the native bus speed. Therefore it is imperative that the CPU cores handling ethernet interrupts are located in the CPU directly connected to the network card. The `lstopo(1)` program (see screenshot next page) can be used to inspect the system. The `flexbuf.recording.txt` document describes how this can be done by modifying an IRQ's `smp_affinity` property (`/proc/irq/NNN/smp_affinity`).

The recording application (such as *jive5ab*) must be constrained to running on the remaining core(s) on that CPU. This is done using the `taskset(1)` or `numactl(1)` utility, typically wrapped in the `StartJ5` script, the latter which is available from the `scripts/` subdirectory of *jive5ab*'s source code.

[17] Available from `http://www.jive.eu/~verkout/flexbuff/flexbuf.recording.txt`


lstopo
Machine (63GB total)
NUMANode P#0 (31GB)
Package P#0
L3 (20MB)
L2 (256KB)
L1d (32KB)
L1i (32KB)
Core P#0
PU P#0
Core P#1
PU P#1
Core P#2
PU P#2
Core P#3
PU P#3
Core P#4
PU P#4
Core P#5
PU P#5
Core P#6
PU P#6
Core P#7
PU P#7
PCI 8086:1572
ens1f0
PCI 8086:1572
ens1f1
PCI 1000:0087
sdh
sdf
sdd
sdb
sdg
sde
sdc
sda
PCI 1000:0087
sdp
sdn
sdl
sdi
sdq
sdo
sdm
sdj
PCI 8086:1528
eno1
PCI 8086:1528
eno2
PCI 8086:8d62
PCI 1a03:2000
controlD64
card0
PCI 8086:8d02
sdk
NUMANode P#1 (31GB)
Package P#1
L3 (20MB)
L2 (256KB)
L1d (32KB)
L1i (32KB)
Core P#0
PU P#8
Core P#1
PU P#9
Core P#2
PU P#10
Core P#3
PU P#11
Core P#4
PU P#12
Core P#5
PU P#13
Core P#6
PU P#14
Core P#7
PU P#15
PCI 1000:0087
sdt
sdr
sdy
sdw
sdu
sds
sdx
sdv
PCI 1000:0087
sdag
sdae
sdac
sdaa
sdaf
sdad
sdab
sdz
PCI 1000:0087

12.2.2 Tuning inside *jive5ab* and example command sequence for FlexBuff/Mark6 recording

jive5ab offers a number of configurable parameters affecting high-speed packet recording, specifically the buffer/block sizes and number of disk-writing threads. The example below illustrates this.

When configuring the number of disk writer threads it seems logical to specify “the number of mountpoints” in your system, but that has proven to be counter-effective because the many disk writer threads start to compete over shared resources. To illustrate, using only nine (9) disk writer threads on a Mark6 with 32 mounted hard disks already yields a total write performance of ~ 50 Gbps. The default is one (1), which might be sufficient for recordings up to ~ 4 Gbps, depending on the age of the SATA disk — newer, larger, SATA disks tend to be faster, approaching the 6 Gbps SATA channel limit.

The following set of commands, when sent to *jive5ab*, are sufficient to trigger and customize a full FlexBuff/Mark6 recording. See the individual command documentation for each command’s purpose and full explanation of the available options.

FlexBuff/Mark6 recordings require a known data format, specifically to be able to compute (Mark4/5), or set explicitly (VDIF) the size of the frames to expect on the network:

```
mode = vdif_8192-1024-1-16;
```

It is also important to configure the network protocol; e.g. if a sequence number is present and what to do with it; what socket read buffer size to use, which chunk size for the vbs files:

```
net_protocol= pudp : 32M : 256M;
```

The default MTU is set to 1500 inside *jive5ab*. To correctly receive frames larger than that, this parameter must be updated:

```
mtu = 9000;
```

Data must be read from a (UDP) port number. This example exploits the *jive5ab* 3.0.0+ feature of `net_port=` to indicate listening on a specific local address in stead of the default “listen on all interfaces” — a multicast address for example:

```
net_port = 239.17.12.2@17206;
```

Optionally onfigure or refresh the list of disks (mount points) to write to:

```
set_disks = mk6;
```

Optionally configure the number of disk writer threads if the default is inadequate:

```
record = nthread : : 9 ;
```

A subsequent `record=on : <scan>` should start recording any incoming (UDP) packets.

12.3 Filtering/grouping VDIF data frames in separate recordings

jive5ab 3.0.0 introduces the possibility to let the recorder filter/group VDIF frames matching certain criteria automatically in separate recordings. A simple example is to record frames from each received VDIF thread in their own recording. Multi-thread VDIF files are really difficult to correctly handle, so letting the recorder turning a multi-thread UDP VDIF stream into separate, single thread, VDIF files may benefit VDIF-processing software downstream.

12.3.1 The basic ideas/high level overview

The idea is simple. Each datastream, identified by a string `<name>`, describes one or more VDIF match criteria. If a VDIF frame is received whilst there are datastream(s) defined, all currently defined match criteria are tested until one matches. The frame will be collected in the data files for the datastream whose match criterion returned true. If no match criterion returned true, the frame will be discarded.

The name of the recording where matching frames will be stored is formed from the scan name (as set by `record=on:<scan>`), suffixed by the expanded datastream "_`<name>`". Expanded because the datastream `<name>` may contain so-called *replacement fields*. There are two such defined fields: `{station}` and `{thread}`. It is important to realize the curly braces are part of the field. If any/all of these appear in `<name>`, their value will be replaced by the converted-to-string value of the corresponding field from the matched VDIF frame before yielding the *actual* suffix for the recording.

The datastream configuration is managed through the `datastream=` command.

12.3.2 Match specifications

As far *jive5ab* is concerned, each VDIF frame is characterized by four parameters. Besides the VDIF thread- and station id encoded in the VDIF frame header, *jive5ab* has access to the *sending* IPv4 address and port number. *jive5ab* allows matching VDIF frames on any of those field values.

A VDIF match specification has the following format:

```
[[[<ip|host>]@<port>]/][<station>.]thread(s)
```

where square brackets denote, as usual, optionality. Unspecified match fields are treated as "match everything". Most fields also have an explicit "match all" wildcard: `*`, see table.

`<ip|host>`	Optional resolveable host name or dotted-quad IPv4 address to match
`<port>`	The optional UDP port number to match
`<station>`	The optional VDIF station to match. Can be `0xXXXX` numerical or one or two ASCII characters or the wildcard `*`.
`thread(s)`	The non-optional comma separated thread id(s) to match, individual ids or ranges or `*`

12.3.3 Examples

Q: "*I want to automatically record each VDIF thread in in its own file.*"

A: Send the following command:

```
datastream = add : {thread} : * ;
```

In plain English this reads: Add a datastream by the `<name>` of just `{thread}`. Put into this datastream VDIF frames with thread-id matching `*` (i.e. any thread id). If a frame is received, the actual suffix of the recording will be expanded to "`_<threadid from matched frame>`", i.e. each VDIF thread is automagically recorded in its own recording.

It is equally simple to record only VDIF frames from a specific sender(s). Note that the `<name>`s do not contain *replacement fields*, so the expanded suffices are constant. The example also shows how only a fixed set of VDIF threads are filtered/recorded :

```
datastream = add : rdbeA : 192.168.1.10/0-4,12-16;
datastream = add : rdbeB : 192.168.1.11/5-11 ;
```

Now both devices (e.g. rdbeA and rdbeB in this example) can be configured to send their frames to the same *jive5ab*, which will separate the streams and prevent their crossing.

In the same manner VDIF frames can be filtered on VDIF station id as well:

```
datastream = add : L_Hh : Hh.* ;
```

Meaning "filter all threads for frames with VDIF station 'Hh' in the header", or, using the `{station}` replacement field separate into distinct L, R polarization recordings for each station (assuming even thread id's are RCP):

```
datastream = add : RCP-{station} : *.0,2,4,6 ;
datastream = add : LCP-{station} : *.1,3,5,7 ;
```

The VDIF station id can also be a 16-bit integer value, according to the VDIF standard:

```
datastream = add : invalid : 0xDEAD.* ;
datastream = add : ant01 : 0x1601.* ;
```

Or separate by UDP port number such that data coming from the same sending IPv4 can be separated by sending process, no two processes can send from the same UDP port:

```
datastream = add : sender1 : 192.168.1.10@42667/*.* ;
datastream = add : sender2 : 192.168.1.10@42668/*.* ;
```

13. *jive5ab* Command/Query Summary (by category)

In the tables below, each command's availability per system is listed. "5A", "5C" etc. mean the command is available on the indicated Mark5 system. "5I" = Mark5B / DIM, "5O" = Mark5B / DOM. The "G" category means availability on a generic, non Mark5 platform, **including Mark6/FlexBuff**. This includes *jive5ab* compiled for a Mark5 system without Mark5 support, and *all non-default* runtimes (see Section 6) on all systems.

13.1 General

Availability:	5A	5I	5O	5C	G	
disk_state	●	●	●	●		Set/get Disk Module Status (DMS): last significant disk operation
disk_state_mask	●	●	●	●		Set mask to enable changes in DMS
DTS_id?	●	●	●	●	●	Get system information (query only)
OS_rev1?	●					Get details of operating system (query only)
OS_rev2?	●					Get details of operating system (query only)
OS_rev?	●	●	●	●	●	Get details of operating system (query only)
mount	●	●	●	●		Power bank on as if keyed on (command only, `jive5ab` > 2.8.1, see unmount)
protect	●	●	●	●		Set/remove erase protection for active module
recover	●	●	●	●		Recover record pointer which was reset abnormally during recording
reset	●	●	●	●		Reset Mark5 unit (command only)
runtime	●	●	●	●	●	Control multiple simultaneous transfer environments
SS_rev1?	●					Get StreamStor firmware/software revision levels (query only)
SS_rev2?	●					Get StreamStor firmware/software revision levels (query only)
SS_rev?	●	●	●	●		Get StreamStor firmware/software revision levels (query only)
task_id	●				●	Set task ID (primarily for correlator use)
unmount	●	●	●	●		Power bank off as if keyed off (command only, `jive5ab` > 2.8.1)
version?	●	●	●	●	●	Get detailed version information of this *jive5ab* (query only)

13.2 Network Setup and monitoring

Availability:	5A	5I	5O	5C	G	
ack	●	●	●	●	●	Set UDP backtraffic acknowledge period (*jive5ab* > 2.7.3)
evlbi	●	●	●	●	●	Query e-VLBI UDP/UDT statistics (query only)
ipd	●	●	●	●	●	Set packet spacing/inter-packet delay
net_port	●	●	●	●	●	Set IPv4 port number for the data channel
net_protocol	●	●	●	●	●	Set network data-transfer protocol
mtu	●	●	●	●	●	Set network Maximum Transmission Unit (packet) size

13.3 Data Checking

Availability:	5A	5I	5O	5C	G	
data_check?	●	●	●	●		Check data starting at position of start-scan pointer (query only)
file_check?	●	●	●	●	●	Check data between start and end of file (query only)
scan_check?	●	●	●	●	●	Check data between start-scan and stop-scan pointers (query only) (G ≥ 2.6.2)
scan_set	●	●	●	●	●	Set start- and stop-scan pointers for scan/data check and disk2* (G ≥ 2.7.0)
track_check?	●					Check data on selected track (query only)
track_set	●					Select tracks for monitoring with DQA or 'track_check'

13.4 System Setup and Monitoring

Availability:	5A	5I	5O	5C	G	
1pps_source		●				Select source of 1pps synchronization tick
clock_set		●			●	Specify frequency and source of the CLOCK driving the DIM
disk_serial?	●	●	●	●		Return serial #s of all currently active disks (query only)
disk_size?	●	●	●	●		Return individual size of all currently active disks (query only)
DOT?		●				Get DOT (Data Observe Time) clock information (query only)
DOT_inc		●				Increment DOT clock
DOT_set		●				Set DOT clock on next external 1pps tick
error?	●	●	●	●	●	Get error number/message (query only)
get_stats?	●	●	●	●		Get disk performance statistics (query only)
group_def					●	Manage aliases for groups of disks for use in set_disks= (*jive5ab* ≥ 2.7.0)
layout?	●	●	●	●		Get current User Directory format (query only)
mode	●					Set data recording/readback mode/format (Mark5A)
mode		●				Set data recording/readback mode/format (Mark5B/DIM)
mode			●	●	●	Set data recording/readback mode/format (Mark5B/DOM, Mark5C, generic)
packet				●		Set/get packet acceptance criteria
personality	●	●	●	●		Set/get personality (available on 5A, 5B since *jive5ab* ≥ 2.8)
play_rate	●				●	Set playback data rate; set tvg rate
pointers?		●	●	●		Get current value of record, start- and stop-scan pointers (query only)
position?	●			●		Get current value of record and play pointers (query only)
replaced_blks?	●	●	●			Get number of replaced blocks on playback (query only)
reset	●	●	●	●		Reset Mark5 unit (command only)
rtime	●					Get remaining record time on current disk set (Mark5A)
rtime		●		●		Get remaining record time on current disk set (Mark5B/DIM, Mark5C)
rtime					●	Get remaining record time on current disk set (generic) (*jive5ab* ≥ 2.7.0)
set_disks					●	Select mount points to record on (FlexBuff/Mark6) (*jive5ab* ≥ 2.7.0)
start_stats	●	●	●	●		Start gathering disk-performance statistics
status?	●	●	●	●	●	Get system status (query only)
trackmask	●	●	●	●	●	Configure channel dropping setup
tstat?	●	●	●	●	●	Get current runtime status and performance
vsn	●	●	●	●		Write extended-VSN to permanent area

13.5 Data Transfer, Record, Play

Note: data transfers can be monitored using “tstat?”

Availability:	5A	5I	5O	5C	G	
datastream					●	Manage storing of VDIF frames in separate recordings (*jive5ab* ≥ 3.0.0)
disk2file	●	●	●	●	●	Transfer data between start- and stop-scan pointers to file (G ≥ 2.7.0)
disk2net	●	●	●	●	●	Transfer data between start- and stop-scan pointers to network (G ≥ 2.7.0)
file2disk	●	●	●	●		Transfer data from file to Mark5 disk pack
file2net	●	●	●	●	●	Transfer data from file on disk to network
fill2net/fill2file	●	●	●	●	●	Transfer fill pattern from host to network or file on disk
fill2disk	●	●	●	●		Record fill pattern to Mark5 disk pack (*jive5ab* ≥ 2.8)
fill2vbs					●	Record fill pattern to FlexBuff/Mark6 disks (*jive5ab* ≥ 2.8)
in2file	●	●		●		Transfer data directly from Mark5 input to file on disk
in2fork	●	●		●		Duplicate data from Mark5 input to Mark5 disks and network
in2mem	●	●		●		Transfer data directly from Mark5 input to *jive5ab* internal buffer
in2memfork	●	●		●		Duplicate data from Mark5 input to Mark5 disks and *jive5ab* internal buffer
in2net	●	●		●		Transfer data directly from Mark5 input to network
mem2file	●	●	●	●	●	Transfer data from *jive5ab* internal buffer to file on disk
mem2net	●	●	●	●	●	Transfer data from *jive5ab* internal buffer to network
mem2time	●	●	●	●	●	Decode data from *jive5ab* internal buffer into queryable time stamp
net2disk	●	●	●	●		Transfer data from network to Mark5 disk pack
net2file	●	●	●	●	●	Transfer data from network to file on disk
net2mem	●	●	●	●	●	Transfer data from network to file on disk
net2out	●					Transfer data from network to Mark5 output
play	●					Play data from current play pointer position
record	●	●		●		Turn recording on|off; set scan label
record					●	Turn recording on|off; set scan label; configure Mark6/FlexBuff setup
sp*2*	●	●	●	●	●	Configure, start, stop corner turning: split [in/fill/net/file/disk/vbs] to [net/file]

1pps_source – Select source of 1pps synchronization tick

Command syntax: 1pps_source = <1pps source> ;
Command response: ! 1pps_source = <return code> ;

Query syntax: 1pps_source? ;
Query response: ! 1pps_source ? <return code> : <1pps source> ;

Purpose: Select source of 1pps which will be used to synchronize the Mark 5B.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<1pps_source>	char	altA | altB | vsi	vsi	'altA' = AltA1PPS (LVDS signal on 14-pin VSI-H connector; rising edge) 'altB' = AltB1PPS (LVTTL signal on SMA connector on back panel; rising edge) 'vsi' = VSI (LVDS signal on 80-pin VSI-H connector; rising edge)

Monitor-only parameters:

Parameter	Type	Values	Comments
<1pps_source >	char	altA | altB | vsi	

Notes:
1. –

1pps_source

ack – Set UDP backtraffic acknowledge period (*jive5ab > 2.7.3, jive5ab-2.7.1-ack-udfix*)

Command syntax: ack = <ACK period> ;
Command response: !ack = <return code> ;

Query syntax: ack? ;
Query response: !ack ? <return code> : <ACK period> ;

Purpose: Set acknowledgement period for UDP readers; every <ACK period>'th received packet will generate back traffic

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<ACK period>	int	See Note 2.	10	Acknowledgement period in number-of-received-packets-between acknowledgements. See Note 1.

Monitor-only parameters:

Parameter	Type	Comments
<ACK period>	int	Current acknowledgement period

Notes:

1. Network equipment such as routers and switches have no truck with uni-directional network traffic: they learn the network port a destination is reachable on dynamically, in order to only forward traffic to that physical network port. This learning is done by broadcasting traffic to all ports in case the addressee is unknown and wait for any form of traffic originating from it, back to the sender. With the TCP/IP and UDT protocols this is already guaranteed. In order to prevent the unidirectional UDP based e-VLBI traffic 'blowing up' the network - the equipment continues to broadcast until learnt – backtraffic has to be generated at least once: as soon as the UDP reader receives a packet for the first time. This is, however, not enough. Most equipment remembers the learnt addressee/physical port combination for a few minutes at best. Some equipment even flushes their entire memory in case one of its ports is reset, requiring the device to re-learn all mappings. To this effect *jive5ab* > 2.7.3 features the "`ack`" command/query. *jive5ab* will generate backtraffic every <ACK period>'th received packet if any of the udp-based protocols (bar UDT) is used (see "`net_protocol`"). The compiled-in default is every 10th packet. This value was experimentally determined in May 2016 to be both prevent significant packet loss whilst not causing too much effect on return bandwidth as well as not causing too much extra overhead at the receiving end.
2. This feature was backported into the 2.7.1 stable series for internal use at JIVE. The public release of the "user directory fix" bug fix release of jive5ab-2.7.1-ack-udfix therefore includes this feature as well.
3. The ACK period is a signed integer. The result of setting a value is as follows:

ack < 0	Acknowledge every received packet; 1-for-1 backtraffic
ack = 0	Reset the 'ACK period' to the compiled-in default
ack > 0	Generate backtraffic every `ack` received packets

bank_info – Get bank information (query only)

Query syntax: bank_info? ;
Query response: !bank_info ? <return code> : <selected bank> : <#bytes remaining> : <other bank> : <#bytes remaining> ;

Purpose: Returns information on both selected and unselected banks, including remaining space available.

Monitor-only parameters:

Parameter	Type	Values	Comments
<selected bank>	char	A | B | nb	Currently selected bank, or 'nb' if operating in non-bank mode. See Note 3. '-' if disk module is faulty; see Note 1
<#bytes remaining>	int		Approximate #bytes remaining to be recorded on active module or on a non-bank-mode module pair. =0 if no module selected or faulty module.
<other bank>	char		Bank mode: Unselected bank, if module in unselected bank is mounted and ready; if no module or faulty module, '-' is returned. 'nb' mode: returned null
<#bytes remaining>	int		Bank mode: Approximate #bytes remaining to be recorded on inactive module; =0 if no module active, faulty module. 'nb' mode: returned null

Notes:

1. If no modules are inserted, an error code 6 is returned.
2. The estimate of <#bytes remaining> is made without taking into account any slow or bad disks. When recording is not in progress, an 'rtime?' query gives a more precise estimate of the available space for the selected bank.
3. *jive5ab* < 2.8 returns error code 6 "not in bank mode" if this query is executed whilst running in non-bank mode. *jive5ab* ≥ 2.8 implements the `bank_info?` query according to this specification.

bank_info

bank_set – Select active bank for recording or readback

Command syntax: bank_set = <bank> ;
Command response: ! bank_set = <return code> ;

Query syntax: bank_set? ;
Query response: ! bank_set ? <return code> : <active bank> : <active VSN> : <inactive bank> : <inactive VSN> ;

Purpose: When in bank mode, the selected bank becomes the 'active' bank for all Mark 5 activities.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<bank>	char	A | B | inc	A	'inc' increments to next bank in cyclical fashion around available bank; see Note 1. 'bank_set command will generate an error when operating in 'nb' mode; see Note 2.

Monitor-only parameters:

Parameter	Type	Values	Comments
<active bank>	char	A | B | nb	'A' or 'B' if there is an active bank; '-' if no active bank; 'nb' if operating in 'non-bank mode'. See Note 5!
<active VSN>	char		VSN of active module, if any; if operating in 'nb' mode: VSN of module **that should be** in Bank A.
<inactive bank>	char	B | A | -	'B' or 'A' if inactive bank is ready; '-' if module not ready; 'nb' if operating in 'non-bank mode'
<inactive VSN>	char		VSN of inactive module, if any; if operating in 'nb' mode: VSN of module **that should be** in Bank B

Notes:

1. If the requested bank is not the bank already selected, a completion code of '1' (delayed completion) is returned. Bank switching takes a variable amount of time up to about 3 seconds. While bank switching is in progress, many commands and queries will return a code of 5 (busy, try later) or 6 (conflicting request; in effect, neither bank is selected during this transition). If an attempt to switch the bank fails (e.g. if there is no 'ready' disk module in the other bank), a 'status?' or "error?' query will return error 1006, "Bank change failed." A 'bank_set?' query will indicate whether the bank has changed. Switching banks can also generate other errors if there are problems with the target bank. If no active banks are present in the system a completion code of '6' (inconsistent or conflicting request) will be returned.
2. When operating in 'nb' (i.e. non-bank) mode, a 'bank_set' command is illegal and will generate an error; a 'bank_set?' query is allowed to gather information. ~~The system will switch automatically to 'nb' mode if (and only if) both 'nb' modules are properly mounted and ready.~~
3. The 'bank_set' command may not be issued during recording or readback (will return an error).
4. When operating in bank mode, a 'bank_set?' query always returns the currently active module.
5. *jive5ab* ≥ 2.8 properly implements this query in non-bank mode. Versions prior to that will return error code 6, "not in bank mode"

bank_set

bank_switch – Enable/disable automatic bank switching (NYI)

Command syntax: bank_switch = <auto-switch on/off> ;
Command response: !bank_switch = <return code> ;

Query syntax: bank_switch? ;
Query response: !bank_switch ? <return code> : <auto-switch on/off> ;

Purpose: Enable/disable automatic bank switching for both record and readback.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<auto-switch mode>	char	off|on	off	If 'on', enables automatic bank switching; always 'off' in non-bank mode..

Notes:

1. 'bank_switch' command will return error if system is operating in 'nb' (non-bank) mode.
2. When automatic bank switching is enabled, the following actions are triggered when recording hits end-of-media (say, on Bank A):
   a. Bank A stops recording and updates its directory.
   b. Bank B is selected as the 'active' bank (assumes Bank B is ready).
   c. Recording starts on Bank B and continues until a 'record=off' command is issued.
3. During the bank-switching action, up to one second of data may be lost.
4. In the example above, if Bank B is not empty, the data on Bank B will be extended in the usual manner (i.e. no existing data on Bank B will be lost). In this case, automatic bank switching on readback will not work properly.
5. If the alternate Bank is not ready at the time switching is initiated, the recording or readback will stop.
6. The 'continuation segment' of the scan on the alternate disk module maintains the same scan label as the originating segment, except that the 'initial' and 'continuation' segments are identified by a trailing or preceding (respectively) '+' character added to the scan name subfield of the scan label when a 'scan_set?' query is executed.
7. None of the Mark5A, DIMino or drs programs implement this command as it cannot, effectively, be used at all. *jive5ab* has implemented it but its correction functioning should not be relied upon.

bank_switch

clock_set – Specify CLOCK parameters

Command syntax: clock_set = <clock frequency> : <clock_source> : [<clock-generator freq>] ;
Command response: !clock_set = <return code> ;

Query syntax: clock_set? ;
Query response: !clock_set ? <return code> : <clock frequency> : <clock source> : <clock-generator freq> ;

Purpose: Specify the frequency and source of the CLOCK driving the DIM

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<clock frequency>	int	2 | 4 | 8 | 16 | 32 | 64	-	DOT clock will advance according to specified frequency; see Note 1. Certain restrictions apply when <clock frequency> = 64; see Note 2. ~~Must be specified before a 'DOT_set' command is issued – see Note 3.~~
<clock_source>	int	ext | int	ext	ext – VSI clock int – Use clock from internal clock-frequency generator
<clock-generator freq>	real	0 – 40 (MHz)	<clock frequency> (but max 40MHz)	Frequency to which internal clock generator is set. See Note 1. Note that clock generator can be set a maximum of 40MHz.

Monitor-only parameters:

Parameter	Type	Values	Comments
<clock frequency>	int		

Notes:

1. The DOT clock timekeeping will advance by counting clock cycles according to the specified value of <clock frequency>, regardless of the actual clock frequency. For example, if <clock frequency> is specified as 32 MHz, but the actual value is 16 MHz, the length of a DOT second with be 32,000,000 clock cycles, occupying 2 wall-clock seconds. Occasionally, such a mis-setting may be deliberate, mostly for testing purposes; for example, setting <clock frequency> to 64MHz when the actual frequency of the clock is 32MHz would allow testing 64MHz functions of the DIM at half-speed. Likewise, if the clock source is chosen to be the internal clock generator, the specified <clock frequency> need not necessarily correspond; the ratio of the <clock-generator freq> to <clock frequency> will determine the DOT clock rate relative to actual wall clock rate (e.g. OS clock rate).
2. If the <clock_frequency> is 64 MHz, the combination of bit-stream mask and decimation must be set so as not to exceed 1024Mbps aggregate bit rate, except for Mark 5B+.
3. ~~A 'clock_set' command issue after a 'DOT_set' command will cause the value of the DOT clock to become indeterminant.~~ *jive5ab* ties the execution of 'DOT_set' (and thus the DOT clock) to the 1PPS tick and thus suffers no indeterminancy. However, the 1PPS only occurs after a 'clock_set' has been issued.
4. A 'DOT_set' command issued before a 'Clock_set' and '1pps_source' will cause an error.
5. ~~'clock_frq' may be used as a synonym for 'clock_set' for VSI compatibility.~~ *jive5ab* does not have the 'clock_frq' command.
6. Specifying the clock-generator frequency whilst setting the clock source to 'ext' will effectively re-program the internal clock chip, even though this has no effect. A warning is issued and in future releases the behaviour may be changed.

clock_set

data_check – Check data starting at position of start-scan pointer (query only)

Query syntax: data_check? ;
Query response: !data_check ? <return code> : <data source> : <start time> : <date code> : <frame#> : <frame header period> : <total recording rate> : <byte offset> : <#missing bytes>;

Purpose: Reads a small amount of data starting at the start-scan pointer position and attempts to determine the details of the data, including mode and data time. For most purposes, the 'scan_check' command is more useful.

Monitor-only parameters:

Parameter	Type	Values	Comments
<data source>	char	ext | tvg | ? vdif | legacy vdif | Mark5B [st :] vlba | mark4	ext – data from Mark 5B DIM input port tvg – data from internal tvg (as indicated by bit in disk frame header) ~~? – data not in Mark 5B format (might be Mark5A, for example); all subsequent return fields will be null~~ See note 6.
<start time>	time		Time tag at first disk frame header. See Note 4 below. See note 6.
<date code>	int		~~3-digit date code written in first disk frame header (modulo 1000 value of Modified Julian Day).~~ See note 6.
<frame#>	int		Extracted from first disk frame header; frame# is always zero on second tick. See note 6.
<frame header period>	time		Each disk frames contains a 16-byte header followed by 10000 bytes of data
<total recording rate>	real	(Mbps)	
<byte offset>	int		Byte offset from start-scan pointer to first disk frame header.
<#missing bytes>	int	bytes	Number of missing bytes between last and current 'data_check'; Should be =0 if immediately previous 'data_check' was within same scan Meaningless if immediately previous 'data_check' was in a different scan, or if data are not formatted VLBI data. Null if <#missing bytes> cannot be calculated; see Note 5.

Notes:

1. Starting at the start-scan pointer position, the 'data_check' query searches to find the first valid disk frame header.
2. The 'data_check' query will be honored only if record is off.
3. The 'data_check' query does not affect the start-scan pointer.
4. Regarding the <start time> value returned by the 'data_check?' and, 'scan_check?' queries: The year and DOY reported in <start time> represent the most recent date consistent with the 3-digit <date code> in the frame header time tag (modulo 1000 value of Modified Julian Day as defined in VLBA tape-format header); this algorithm reports the proper year and DOY provided the data were taken no more than 1000 days ago.
5. The <#missing bytes> parameter is calculated as the difference the expected number of bytes between two samples of recorded data based on embedded time tags and the actual observed number of bytes between the same time tags. The reported number is the *total* number of bytes missing (or added) between the two sample points.
6. *jive5ab* extends `data_check?` to recognize all data formats, not just the recorder's native data type (see Section 7). `data_check?`'s output may be subtly different depending on wether it is issued on a Mark5B/DIM, Mark6/FlexBuff or any of the other Mark5's. Specifically for VDIF the number of threads will also be returned or for other formats some fields may be omitted (e.g. `<frame #>`).

data_check

datastream – Manage automatic storing of VDIF frames in separate recordings (*jive5ab* ≥ 3.0.0)

Command syntax: datastream = add : <name> : <spec> [: <spec>]*;
datastream = remove : <name> ;
datastream = reset ;
datastream = clear ;

Command response: !datastream = <return code> ;

Query syntax: datastream? [<name>];

Query response: !datastream? <return code> [: <name | spec>]* ;

Purpose: Manage *jive5ab*'s ability to filter incoming VDIF frames by VDIF identifiers and record separately. See Section 12.3.

Settable parameters:

Parameter	Type	Comments
add remove reset clear	literal ASCII	'add' – add a new datastream definition to the currently defined set of datastreams 'remove' – similar, only now remove it 'reset' – keep definitions of datastreams but clear cache of matched/expanded streams. See Note 3. 'clear' – remove all defined datastreams, turn feature off
<name>	char	The name of the datastream to manage. See Note 1.
<spec>	char	A VDIF match specification, describing frames to match. See Note 2.

Monitor-only parameters:

Parameter	Type	Default	Comments
<name>	char	none	If no <name> specified, return all defined data stream <name>. Otherwise return all the <spec>'s defined for the indicated datastream <name>
<spec>	char		A VDIF match specifications, specifying the VDIF frames that would be collected in this stream

Notes:

1. A datastream is identified by its `<name>`. The name will also be the suffix to the recording that will hold frames matching the `<spec>`'s defined for this datastream. After `record=on:<scan>,` and frame(s) matching any of the `<spec>`'s for a datastream are received, a recording by the name of `<scan>_<name>` is created. The datastream `<name>` may contain any/all of the replacement fields `{station}` and/or `{thread}`, which will dynamically expand to the value of the corresponding field in a matched VDIF frame.
2. A `<spec>` is a matching specification of the form **[[<ip|host>][@<port>]/][station.]thread(s)**. Brackets, as usual, indicate optionality. Unspecified match fields count as "match everything". The shortest `<spec>` is `*`, i.e. match all threads from all senders and ports and all VDIF station ids. See Section 12.3 for details and examples.
3. Because `<name>` can contain replacement fields, *jive5ab* keeps an internal cache of actual recording suffixes. It is important to clear this cache between recordings. By using 'reset' all currently defined datastreams remain intact and only this cache is cleared, meaning the datastream configuration needs only be specified once, e.g. at the start of an experiment.

datastream

dir_info – Get directory information (query only)

Query syntax: dir_info? ;
Query response: !dir_info ? <return code> : <number of scans> : <total bytes recorded> : <total bytes available> ;

Purpose: Returns information from the data directory, including number of scans, total bytes recorded and remaining bytes available.

Monitor-only parameters:

Parameter	Type	Values	Comments
<number of scans>	int		Returns number of scans currently in the data directory. See Note 3!
<total bytes recorded>	int		Sum over all recorded scans
<total bytes available>	int		Sum of total available disk space (unrecorded plus recorded)

Notes:

1. The scan directory is automatically stored each time data are recorded to the disks.
2. On Mark6/FlexBuff the `dir_info?` returns the sum of the recorded area and free space over all disks currently returned from the `set_disks?` query. The number of scans is unknown and '?' will be returned.
3. *jive5ab* < 2.8 returns error code 6 "not in bank mode" when this query is executed whilst running in non-bank mode. *jive5ab* ≥ 2.8 implements this query according to this specification in both bank and non-bank mode.

dir_info

disk_model – Get disk model numbers (query only)

Query syntax: disk_model? ;
Query response: !disk_model ? <return code> : <disk model#> : <disk model#> : ……;

Purpose: Returns a list of model numbers in currently selected disk module.

Monitor-only parameters:

Parameter	Type	Values	Comments
<disk model#>	literal ASCII		Returned in order of drive number (0=0M, 1=0S, 2=1M, 3=1S, etc); a blank field is returned for an empty slot. When operating in 'nb' mode, disks in banks A and B are treated as a single module.

disk_model

disk_serial – Get disk serial numbers (query only)

Query syntax: disk_serial? ;
Query response: !disk_serial ? <return code> : <disk serial#> : <disk seriall#> : ……;;

Purpose: Returns a list of serial numbers in currently selected disk module.

Monitor-only parameters:

Parameter	Type	Values	Comments
<disk serial#>	literal ASCII		Returned in order of drive number (0=0M, 1=0S, 2=1M, 3=1S, etc); A blank field is returned for an empty slot. When operating in 'nb' mode, disks in banks A and B are treated as a single module.

disk_serial

disk_size – Get disk sizes (query only)

Query syntax: disk_size? ;
Query response: !disk_size ? <return code> : <disk size> : <disk size> : ……;

Purpose: Returns individual capacities of currently selected module.

Monitor-only parameters:

Parameter	Type	Values	Comments
<disk size>	int	bytes	Returned in order of drive number (0=0M, 1=0S, 2=1M, 3=1S, etc); A blank field is returned for an empty slot. When operating in 'nb' mode, disks in banks A and B are treated as a single module.

disk_size

disk_state –Set/get Disk Module Status (DMS): last significant disk operation

Command syntax: disk_state = <DMS> ;
Command response: !disk_state = <return code> : <DMS> ;

Query syntax: disk_state? ;
Query response: !disk_state? <return code> : <active bank> : <active-bank DMS> : <inactive bank> : <inactive-bank DMS> ;

Purpose: Set/get Disk Module Status (DMS), which logs the last significant operation that happened on the disk module.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<DMS> (disk Module Status)	char	recorded | played | erased | unknown | error	none	To be used only if automatically-set DMS parameter is to be overwritten. Requires a preceding 'protect=off' and affects only the active module. Current value of 'disk_state_mask' is ignored.

Monitor-only parameters:

Parameter	Type	Values	Comments
<active bank>	char	A | B	Currently selected bank; '-' if disk module is judged faulty; 'nb' if operating in non-bank mode (≥ 2.8)
<active-bank DMS>	char	recorded | played | erased | unknown | error	recorded – last significant operation was record or a record-like function (net2disk or file2disk). played – last significant operation was playback; disk2net and disk2file do not affect DMS. erased – last significant operation was erase or conditioning, either from 'reset=erase' or SSErase. unknown – last significant operation was performed with version of *dimino* or *SSErase* prior to implementation of the DMS function. error – error occurred; for example, an interrupted conditioning attempt or a failure during one of the significant operations above
<inactive bank>	char	B | A | -	Unselected bank, if module is mounted and ready; if no module or faulty module, '-' is returned.
<inactive-bank DMS>	char	recorded | played | erased | unknown | error	See above.

Notes:

1. Normally, the setting of the DMS parameter happens automatically whenever a record, play or erase command is issued. However, the <disk_state=...> command is provided to manually overwrite the current DMS parameter. This command requires a preceding 'protect=off' and affects only the active module. A 'disk_state=…" command ignores the current value of the disk_state_mask (see 'disk_state_mask' command).
2. The DMS logs the last significant operation that occurred on a disk module. It is designed to distinguish between disk modules waiting to be correlated, have been correlated, or have no data (erased) and ready to be recorded. The DMS is saved on the disk module in the same area as the permanent VSN so that the DMS from both active and inactive disk banks are accessible. Commands scan_check, data_check, disk2net, and disk2file, do not affect DMS.
3. The 'disk_state' and 'disk_state_mask' commands were requested by NRAO and are designed primarily for use at a correlator.
4. If no modules are inserted, an error code 6 is returned.
5. When operating in 'nb' mode, disks in banks A and B are treated as a single module; no inactive bank information is returned in the reply.

disk_state

disk_state_mask – Set mask to enable changes in DMS

Command syntax: disk_state_mask = <erase_mask_enable> : <play_mask_enable> : <record_mask_enable>;
Command response: !disk_state_mask = <return code> : <erase_mask_enable> : <play_mask_enable> : <record_mask_enable> ;

Query syntax: disk_state_mask? ;
Query response: !disk_state_mask? <return code> : <erase_mask_enable> : <play_mask_enable> : <record_mask_enable> ;

Purpose: Set mask to enable changes in DMS.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<erase_mask_enable>	int	0 | 1	1	0 – disable an erase operation from modifying the DMS. 1 – enable erase operation to modify the DMS.
<play_mask_enable>	int	0 | 1	1	0 – disable a play operation from modifying the DMS. 1 – enable play operation to modify the DMS.
<record_mask_enable>	int	0 | 1	1	0 – disable a record operation from modifying the DMS. 1 – enable record operation to modify the DMS.

Notes:

1. The disk_state_mask is intended to prevent accidental changes in the DMS. When a module is at a station, the disk_state_mask setting of 1:0:1 would disable a play operation from modifying the DMS. Likewise, at a correlator one might want to disable the record_mask_enable.

disk_state_mask

disk2file – Transfer data from Mark 5 or FlexBuff/Mark6 to file (*jive5ab* ≥ 2.7.0)

Command syntax: disk2file = [<destination filename>] : [<start byte#>] : [<end byte#>] : [<option>] ;
Command response: !disk2file = <return code> ;

Query syntax: disk2file? ;
Query response: !disk2file ? <return code> : <status> : <destination filename> : <start byte#> : <current byte#> : <end byte#> : <option> ;

Purpose: Transfer data between start-scan and stop-scan pointers from Mark 5 to file.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<dest filename>	literal ASCII	no spaces allowed	See Comments	Default <dest filename> is as specified in Section 6 (i.e. '<scan label>_bm=<bit mask>.m5b'). Filename must include path if path is not default (see Note 5).
<start byte#>	int | null		See Notes	Absolute byte#; if null, defaults to start-scan pointer. See Notes 1 and 2.
<end byte#>	int | null		See Notes	Absolute end byte#; if preceded by '+', increment from <start byte#> by specified value; if null, defaults to stop-scan ponter. See Notes 1 and 2.
<option>	char	n | w | a	n	n – create file; error if existing file w –erase existing file, if any; create new file. a – create file if necessary, or append to existing file

Monitor-only parameters:

Parameter	Type	Values	Comments
<dest filename>			Destination filename (returned even if filename was defaulted in corresponding 'disk2file' command)
<status>	char	active | inactive	Current status of transfer
<current byte#>	int		Current byte number being transferred

Notes:

1. The 'scan_set' command is a convenient way to set the <start byte#> and <stop byte#>.
2. If <start byte#> and <end byte#> are null, the range of data defined by 'scan_set' will be transferred.
3. To abort data transfer: The 'reset=abort' command may be used to abort an active disk2file data transfer. See 'reset' command for details.
4. When <status> is 'inactive', a 'disk2file?' query returns the <dest filename> of the last transferred scan, if any.
5. Default path is the Linux default, which is the directory from which *dimino* or *Mark 5B* was started.

disk2file

disk2net – Transfer data from Mark 5 or FlexBuff/Mark6 to network (*jive5ab* ≥ 2.7.0)

Command syntax: disk2net = connect : <target hostname> ;
disk2net = on : [<start byte#>] : [<end byte#>] ;
disk2net = disconnect ;
Command response: !disk2net = <return code>;

Query syntax: disk2net? ;
Query response: !disk2net ? <return code> : <status> : <target hostname> : <start byte#> : <current byte#> : <end byte#> ;

Purpose: Transfer data between start-scan and stop-scan pointers from Mark 5 to network.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | disconnect		'connect' – connect to socket on receiving Mark 5 system 'on' – start data transfer 'disconnect' – disconnect socket See Notes.
<target hostname>	char		localhost or previously set name	Required only on if <control>='connect'..
<start byte#>	[+] int | null		See Note 1	Absolute byte# or relative offset; if null, defaults to start-scan pointer. See Note 1. See Note 10.
<end byte#>	int | null		See Note 1	Absolute end byte#; if preceded by '+', increment from <start byte#> by specified value; if null, defaults to start-scan pointer. See Note 1.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	connected | active | inactive	Current status of transfer
<target hostname>	char		
<current byte#>	int		Current byte number being transferred

Notes:

1. The '<scan_set> command is a convenient way to set the <start byte#> and <stop byte#>.
2. If <start byte#> and <end byte#> are null, the scan defined by 'scan_set' will be transferred.
3. To set up connection: First, issue 'open' to the *receiving* system ('net2disk=open' or 'net2out=open' to Mark 5, or Net2file as standalone program; then issue 'connect' to the *sending* system ('in2net=connect:..' or 'disk2net=connect:…' to Mark 5).
4. To start data transfer: Issue 'on' to *sending* system ('in2net=on' or 'disk2net=on' to Mark 5). A 'disk2net' transfer will stop automatically after the specified number of bytes are sent.
5. To stop data transfer: Issue 'off' to the sending system ('in2net=off' to Mark 5). After each transfer has been stopped or completed, another transfer may be initiated (see Note 2).
6. To close connection: First, issue 'disconnect' to the sender ('in2net=disconnect' or 'disk2net=disconnect' to Mark 5). A 'disk2net=disconnect' command issued before the specified number of bytes are transferred will abort the transfer and close the

disk2net

connection. Then, 'close' the receiver ('net2disk=close' or 'net2out=close' to Mark 5; Net2file ends). Net2file ends automatically on a 'disconnect'. After a 'net2disk' transfer, the data on disk are not ready for use until after a 'net2disk=close' command has been issued.

7. To abort data transfer: The 'reset=abort' command may be used to abort an active disk2net data transfer. A subsequent 'disk2net=disconnect' must be issued to close the socket and put the Mark 5B back into idle. See 'reset' command for details.
8. Only one data transfer activity may be active at any given time. That is, among 'record=on', 'in2net=..', 'disk2net=..', 'net2disk=..', 'net2out=..', 'disk2file=..', 'file2disk=..', 'data_check' and 'scan_check', only one may be active at any given time.
9. Note that the network protocol parameters are set by the 'net_protocol' command.
10. *jive5ab* 2.6.2 and later extend 'disk2net = on : ..' to support a relative start byte number by prefixing the byte number with a literal '+'. The start byte number for the transfer then becomes the sum of the start byte set via 'scan_set = ..' and `<start byte#>`. This is necessary to be able to support resuming a previously interrupted transfer.

disk2net

DOT – Get DOT clock information (query only)

Query syntax: DOT? ;
Query response: !DOT ? <return code> : <current DOT reading> : <sync status> : <FHG status> : <current OS time> : <DOT-OS difference> ;

Purpose: Get DOT clock information

Monitor-only parameters:

Parameter	Type	Values	Comments
<current DOT reading>	time		Current value of DOT clock. 1970/1/1 00h00m00s as long as not set using **DOT_set! See Note 4.**
<sync status>	char	not_synced | syncerr_eq_0 | syncerr_le_3 | syncerr_gt_3	'not_synced' – DOT 1pps generator has not yet been sync'ed. See Note 1. 'syncerr_eq_0' - DOT 1pps tick is exactly coincident with selected external 1pps tick. See Note 2. 'syncerr_le_3' – DOT 1pps tick within +/-2 clock cycles of selected external 1pps tick. 'syncerr_gt_3' – DOT 1pps tick more +/-3 clock cycles from selected external 1pps tick
<FHG status>	char	FHG_off | FHG_on	'FHG_off' – Frame Header Generator is not running (<current DOT reading> is software estimate) 'FHG_on' – FHG is running (<current DOT reading> is read from hardware FHG). See Note 3.
<current OS time>	time		Corresponding OS time
<DOT-OS difference>	time		<current DOT reading> minus <current OS time>

Notes:

1. A <sync status> of 'not_synced' indicates that the DOT hardware 1pps generator has not been sync'ed to an external 1pps tick. All other <sync status> returns indicate that the DOT 1pps generator has been sync'ed with a 'DOT_set' command. See also Note 2 of 'DOT_set' command.
2. The <syncerr_eq/le/gt_n> status returns are only relevant provided the selected external 1pps that was used to sync the DOT clock remains connected, selected and active and indicates quantitatively the current relationship between the DOT 1pps and the external 1pps to which the DOT clock was originally sync'ed. Note that if the external 1pps tick is asynchronous to the data clock (e.g. comes from a GPS receiver), one would not expect that exact synchronization would be maintained. Caution: the 1pps tick on the VSI-80 connector from a VSI Mark 4 formatter has a slow leading edge and may not always pass the <exact sync> test, though it should always pass the <approx sync> test.
3. DOT 1pps ticks are always counted by dimino to keep higher-order time; when data collection is inactive, the sub-second part of <current DOT reading> is estimated by software measurement of the time interval from the last DOT 1pps second tick. The hardware Frame Header Generator (FHG) runs only during active data collection and creates the Disk Frame Headers inserted periodically into the data stream transmitted to the StreamStor card; during this time, the <current DOT reading> is read directly from the FHG. The FHG is set up according to the parameters of each recording and is started at the beginning of each recording. The time resolution of the FHG is the Mark 5B disk-frame-header period, which is given by 80/(total-rate in Mbps) milliseconds, where the total data rate is determine by three parameters: clock rate (set by 'clock_set' command), bit-stream mask and decimation ratio (both set by 'mode' command); for example, for a total data rate of 1024Mbps, the time resolution of the FHG is ~78 *micro*seconds. For more details see memo 'Data Input Module Mark 5B I/O Board Theory of Operation'.
4. *jive5ab* keeps the DOT at January 1st, 1970 00h00m00s as long as the DOT clock has not been set up correctly (`1pps_source` + `clock_set` + `DOT_set`). This is in direct contrast with DIMino. *jive5ab* does NOT try to estimate an initial DOT at start-up. A restart of *jive5ab* necessitates re-setting the DOT clock. If, during an observation a really excessive <DOT – OS difference> is measured (million of seconds), it is most likely that the DOT has not been properly initialized. *jive5ab* >= 2.5.0 will not start a recording at all if the DOT has not been set. See also Note 9 of `DOT_set`.

DOT

DOT_inc – Increment DOT clock

Command syntax: DOT_inc = <inc> ;
Command response: ! DOT_inc = <return code> ;

Query syntax: DOT_inc? ;
Query response: ! DOT_inc? <return code> : <inc> ;

Purpose: Increment DOT clock time by specified number of seconds

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<inc>	int		0	Number of seconds to increment DOT clock (may be positive or negative). >0 will advance the DOT clock setting; <0 will retard the DOT clock setting

Monitor-only parameters:

Parameter	Type	Values	Comments
<inc>	int		

Notes:

1. The DOT_inc command should be used to adjust an error in the DOT clock only after the DOT clock has been synchronized to an external 1pps tick with the 'DOT_set' command.
2. A 'DOT_inc' issued when the DOT clock is not running (`1pps_source` + `clock_set` + `DOT_set`) causes an error.

DOT_inc

DOT_set – Set DOT clock on next external 1pps tick

Command syntax: dot_set = <time> : [<option>] ;
Command response: !dot_set = <return code> ;Query syntax: dot_set? ;
Query response: !dot_set ? <return code> : <time> : <option> : <time offset> ;

Purpose: Set initial value of Mark 5B DOT clock on next tick of selected external 1pps source

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<time>	time	null | time	OS time	DOT clock can only be set to an integer second value. See Note 1. If null, sets DOT clock according to current OS time.
<option>	char	null | force	null	If 'force', 1pps generator will be re-synced even though the <DOT_synced> status returned by 'DOT?' already indicates it is sync'ed. See Note 2. If null, 1pps generator will be synced only if <DOT_synced> status indicates it is not already sync'ed.

Monitor-only parameters:

Parameter	Type	Values	Comments
<time>	time		Includes all inferred higher-order time (e.g., year, DOY, etc) that was not explicit in command value of <time>
<option>	char		
<time offset>	time		Estimated interval between time from receipt of 'DOT_set' command to 1pps tick that set the DOT clock

Notes:

1. If <time> does not specify higher-order time (i.e. year, day, etc), the current values from the OS time are used. Of course, the DOT clock should be carefully checked with the 'DOT?' query after setting to verify that it is properly set.
2. Because the Mark 5B keeps higher-order time (above 1sec) in software, higher-order time will be lost *dimino* is restarted or the system is re-booted, but the DOT hardware 1pps generator will not lose sync as long as power and data clock (on the VSI 80-pin connector) are maintained; a re-issued 'DOT_set' command will set higher-order time without disturbing the existing 1pps synchronization unless the 'force' option is specified to force re-synchronization of the hardware 1pps generator.
3. After the 'DOT_set' command is issued, a 'DOT?' query should be subsequently issued to verify the proper time setting. If necessary, the 'DOT_inc' command may be used to adjust the DOT clock to the correct second.
4. If the DOT clock is set from OS time, care must be taken that the OS clock is reasonably well aligned with the external 1pps tick (to within a few tens of milliseconds, at worst). Otherwise, there is no requirement on the OS timekeeping apart from Note 2 above.
5. Following the setting of the DOT clock, the DOT clock keeps time based strictly on the selected CLOCK as specified in the 'clock_set' command and runs completely independently of the OS time; furthermore, the external 1pps is not used for any further operations other than cross-checking against the internally generated DOT1PPS interrupt and, in principle, can be disconnected.
6. Normally the DOT_set operation is only done once at the beginning of an experiment. See also Note 1 for 'DOT?' query.
7. Any change in 'clock_set' parameters enacted after a 'DOT_set' command will cause the value of the DOT clock to become indeterminate.
8. A 'DOT_set' command issued before a 'clock_set' command will cause an error.
9. *jive5ab* keeps the DOT at January 1st, 1970 00h00m00s as long as the DOT clock has not been set up correctly (`1pps_source` + `clock_set` + `DOT_set`). This is in direct contrast with DIMino. *jive5ab* does NOT try to estimate an initial DOT at start-up. A restart of *jive5ab* necessitates re-setting the DOT clock. Once a DOT has been set, *jive5ab* increases its value by exactly 1 second every 1PPS interrupt of the selected `1pps_source`.

DOT_set

DTS_id – Get system information (query only)

Query syntax: DTS_id? ;
Query response: !DTS_id ? <return code> : <system type> : <software revision date> : <media type> : <serial number> : <#DIM ports> : <#DOM ports> : <command set revision> : <Input design revision> : <Output design revision> ;

Purpose: Get Mark 5 system information

Monitor-only parameters:

Parameter	Type	Values	Comments
<system type>	char	mark5A | mark5b | Mark5C | StreamStor | -	'Mark5A-C' – case sensitivity apart these speak for themselves (Mark5B+ returns mark5b as well) 'StreamStor' – only StreamStor card detected (new in `jive5ab` 2.8.2) see Notes '-' – generic system, FlexBuff, Mark6 or just-some-computer
<software revision date>	time		~~Date stamp on current version of *dimino* source code (*.c) files~~ Timestamp of compilation of `jive5ab`
<media type>	int	1	Per VSI-S spec: 1 – magnetic disk [0 – magnetic tape; 2 – real-time (non-recording)]
<serial number>	ASCII		System serial number; generally is in the form 'mark5-xx' where xx is the system serial number
<#DIM ports>	int	1	Number of DIM ports in this DTS
<#DOM ports>	int	1	Number of DOM ports in this DTS
<command set revision>	char		Mark 5AB or C ~~DIM~~ command set revision level corresponding to this software release (e.g. '1.0')
<DIM design revision>	int		Revision level of DIM FPGA design on Mark 5B I/O board

Notes:

1. Since version 2.8.2 `jive5ab` supports 'crippled' Mark5s gracefully. A crippled Mark5 is a system where only a StreamStor card is detected and no I/O board (5A, 5B) or daughter board (5B+, 5C). These systems might be found at a correlator site where only disk playback is required. On such a system *all* disk based transfers but 'record=' and 'in2*=' remain functional.

DTS_id

error – Get error number/message (query only)

Query syntax: error? ;
Query response: !error ? <return code> : <error#> : <error message> : <error time> [: <last error time> : <# occurrences>] ;

Purpose: Get error number, message [and time] causing bit 1 of 'status' query return to be set

Monitor-only parameters:

Parameter	Type	Values	Comments
<error#>	int		Error number associated with 'status'query return bit 1
<error message>	literal ASCII		Associate error message, if any
<error time>	time		System time when this error first occurred
<last error time>	time		If the error occurred more than once, system time when this specific error last occurred (since 2.6.0)
<# occurrences>	int		If the error occurred more than once, the number of times this specific error occurred (since 2.6.0)

Notes:

1. ~~Most errors are 'remembered' (even if printed with debug) and printed (and cleared) by either a 'status?' or 'error?' query. Thus, errors may be remembered even after they have been corrected.~~
2. *jive5ab* maintains a queue of errors. If the queue is non-empty at the time of a '`status?`' query, bit 1 of the status word will be set. An '`error?`' query will remove the oldest error message permanently from the queue.

error

evlbi – Get e-VLBI transfer packet reception and re-ordering statistics

Command syntax: evlbi = [<statistic> : <statistic> : <statistic> : ...];
Command response: !evlbi = <return code> : [<statistic value> : <statistic value> : ...];

Query syntax: evlbi? ;
Query response: !evlbi ? <return code> : <statistic value> : ... ;

Purpose: Return specific or default set of packet statistics on a receiving end of an UDPs/UDPsnor/VTP or UDT based transfer

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<statistic>	char	See notes	none	The parameters form a format string. Recognized specifiers are replaced with the current value they describe.

Monitor-only parameters:

Parameter	Type	Values	Comments
<statistic value>	number	see notes	evlbi? behaves like "evlbi=" with a default set of statistics that are returned

Notes:

1. When the net_protocol is set to udp, udps (synonym for udp), udpsnor (*jive5ab* ≥ 2.8), udt or vtp, the receiving *jive5ab* will collect packet reception statistics, to some extent following IETF metrics[1]. One notable deviation is that out of performance considerations, *jive5ab* will not collect out-of-order statistics for packets that are re-ordered by more than 32 packets.
2. Much like how the C-library strftime(3) has time fields available, *jive5ab* has packet statistics counters available. The command is a printf-style "format string" and *jive5ab* will replace special formatters with their current value as per this table:

%t	Total amount of packets received		
%l	Amount of packets lost (count)	%L	Amount of packets lost (percentage)
%o	Amount of packets out-of-order (count)	%O	Amount of packets out-of-order (percentage)
%d	Amount of discarded packets (count)	%D	Amount of discarded packets (percentage)
%r	Total amount of re-orderdering measured	%R	Average amount of re-ordering per packet
%u	Time stamp, unix format + added millisecond fraction	%U	Time stamp YYYY-MM-DD HHhMMmSS.SSSs

3. "`evlbi?`" is an alias for "`evlbi = total : %u : ooo : %o : disc : %d : lost : %l : extent : %R ;`"
4. For udpsnor the numbers returned are aggregated values of up to eight independent senders

[1] "Packet reordering metrics" - http://www.ietf.org/rfc/rfc4737.txt

evlbi

file_check – Check recorded data between start and end of file (query only)

Query syntax: file_check? [<strict>] : [<#bytes to read>] : <file> ;
Query response: !file_check ? <return code> : <data type> : <ntrack> : <start time> : <scan length> : <total recording rate> : <#missing bytes> [: <data array size>];

Purpose: Offers 'scan_check?' functionality for files-on-disk. This command was introduced in *jive5ab* 2.5.1.

Monitor arguments:

Parameter	Type	Allowed values	Default	Comments
<strict>	int	0 | 1	1	0 – Enable less strict checking: • no CRC checks on frame headers • no check for invalid last digit in MarkIV time stamp or time stamp consistency with data rate • no consistency check between frame number+data rate and VLBA time stamp in Mark5B header 1 – Everything has to be good for data format to be detected
<#bytes to read>	int	> 0	1000000	Amount of data to find frames in. The default ~1MB may be too low for high data rate detection (>>2Gbps)
<file>	ASCII			Non-optional argument: the name of the file to check; there is no 'file_set=' analogous to 'scan_set='

Monitor-only parameters:

Parameter	Type	Values	Comments
<data type>	char	tvg | SS | ? vdif (legacy) | mark5b | [st :] mark4 | vlba	tvg – undecimated 32-bit-wide tvg data (see Note 2) SS – raw StreamStor test pattern data 'st :' – this extra field is inserted if straight-through MarkIV or VLBA is detected
<ntrack>	int		Number of tracks detected ('?' if VDIF)
<start time>	time		Time tag at first frame header in scan. See Note 3.
<scan length>	time		
<total recording rate>	real	Mbps	
<#missing bytes>	int	See Note 4	Should always be =0 for normally recorded data. >0 indicates #bytes that have been dropped somewhere within scan <0 indicates #bytes that have been added somewhere within scan
[<data array size>]	int		Parameter is only returned if the detected format is simple VDIF and reports the VDIF data array length.

Notes:

1. The 'file_check' query essentially executes a 'data_check' starting at the start-of-file, followed by a 'data_check' just prior to the end-of-file. This allows information about the selected file to be conveniently determined.
2. Only tvg data that were recorded with a bit-stream mask of 0xffffffff and no decimation will be recognized.
3. Regarding the <start time> value returned by the 'data_check?' and, 'file_check?' queries: The year and DOY reported in <start time> represent the most recent date consistent with the 3-digit <date code> in the frame header time tag (modulo 1000 value of Modified Julian Day as defined in VLBA tape-format header); this algorithm reports the proper year and DOY provided the data were taken no more than 1000 days ago.
4. The <#missing bytes> parameter is calculated as the difference the expected number of bytes between two samples of recorded data based on embedded time tags and the actual observed number of bytes between the same time tags. The reported number is the *total* number of bytes missing (or added) between the two sample points.

file_check

file2disk – Transfer data from file to Mark 5B

Command syntax: file2disk = <source filename> : [<start byte#>] : [<end byte#>] : [<scan label>] : [<<bit-stream mask> ;
Command response: !file2disk = <return code>;

Query syntax: file2disk? ;
Query response: !file2disk ? <return code> : <status> : <source filename> : <start byte#> : <current byte#> : <end byte#> : <scan#> : <scan label> : <bit-stream mask> ;

Purpose: Initiate data transfer from file to Mark 5 data disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<source filename>	ASCII	no spaces allowed	'save.data' or last value	If not in standardized filename format (see Section 6), must specify at least <scan label> and recommend specifying <bit-stream mask> as well. See Note 1. Filename must include path if not default (see Note 5).
<start byte#>	int		0	Absolute byte number; if unspecified, assumed to be zero
<end byte#>	int		0	If =0, will copy to end of file
<scan label>	ASCII	64 chars max	Extracted from <source filename>	Required if <source filename> is not in standardized format (see Section 6). Example: 'exp53_ef_scan123'
<bit-stream mask>	hex		0	Should be specified if <scan label> is specified. See Note 1.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive	Current status of transfer
<current byte#>	int		Current source byte# being transferred
<scan#>	int		Sequential scan number on disk module
<bit-stream mask>	hex		Bit-stream mask

Notes:

1. If <source filename> is in the standardized format for Mark 5B (see Section 6), *dimino* will parse the constituent fields to determine <experiment name>, <station code>, <scan name> and <bit-stream mask>. If source filename does not include a scan label in the proper format, <scan label> must be specified. If <scan label> is specified, it is recommended that <bit-stream mask> also be specified so that the Mark 5B directory entry can be properly completed.
2. The data in the source file must be in Mark 5B data format.
3. To abort data transfer: The 'reset=abort' command may be used to abort an active file2disk data transfer. See 'reset' command for details.
4. When <status> is 'inactive', a 'file2disk?' query returns <source filename> of the last transferred scan, if any.
5. Default path is the Linux default, which is the directory from which *dimino* or *Mark 5B* was started.

file2disk

file2net – Transfer data from file on disk to network

Command syntax: file2net = connect : <target hostname> : <filename>;
file2net = on : [<start byte#>] : [<end byte#>] ;
file2net = disconnect ;
Command response: !file2net = <return code>;

Query syntax: file2net? ;
Query response: !file2net ? <return code> : <status> : <target hostname> : <start byte#> : <current byte#> : <end byte#> ;

Purpose: Transfer data between start byte and end byte from file to network.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | disconnect		'connect' – connect to socket on receiving Mark 5 system 'on' – start data transfer 'disconnect' – disconnect socket See Notes.
<target hostname>	char		localhost or previously set name	Required only on if <control>='connect'..
<start byte#>	[+] int | null		See Note 1	Absolute byte# or relative offset; if null, defaults to start-scan pointer. See Note 1. See Note 10.
<end byte#>	int | null		See Note 1	Absolute end byte#; if preceded by '+', increment from <start byte#> by specified value; if null, defaults to start-scan pointer. See Note 1.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	connected | active | inactive	Current status of transfer
<target hostname>	char		
<current byte#>	int		Current byte number being transferred

Notes:

11. The '<scan_set> command is a convenient way to set the <start byte#> and <stop byte#>.
12. If <start byte#> and <end byte#> are null, the scan defined by 'scan_set' will be transferred.
13. To set up connection: First, issue 'open' to the *receiving* system ('net2disk=open' or 'net2out=open' to Mark 5, or Net2file as standalone program; then issue 'connect' to the *sending* system ('in2net=connect:..' or 'disk2net=connect:…' to Mark 5).
14. To start data transfer: Issue 'on' to *sending* system ('in2net=on' or 'disk2net=on' to Mark 5). A 'disk2net' transfer will stop automatically after the specified number of bytes are sent.
15. To stop data transfer: Issue 'off' to the sending system ('in2net=off' to Mark 5). After each transfer has been stopped or completed, another transfer may be initiated (see Note 2).

file2net

fill2net/file – Transfer fill pattern from host to network or file on disk

Command syntax: fill2[net|file] = connect : <hostname | filename> [: <start> : <inc> : <real-time>] ;
fill2[net|file] = on [: <nword>] ;
fill2[net|file] = disconnect ;
Command response: !fill2[net|file] = <return code>;

Query syntax: fill2[net|file]? ;
Query response: !fill2net ? <return code> : <status> : <hostname> : <byte#> ;
!fill2file ? <return code> : <status> : <filename> ;

Purpose: Send dynamically generated synthetic data frames with valid time stamps in the headers over the network or to a file.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | disconnect		'connect' – connect to socket on receiving Mark 5 system 'on' – start data transfer 'disconnect' – disconnect socket (cf. 'in2net = '; see documentation)
<hostname | filename>	char		localhost or previously set name	Required only on if <control>='connect'. Interpreted as <filename> for 'fill2file', as <hostname> when 'fill2net' is used.
<start>	int | hex		0x11223344	Fill pattern start value. Only valid if <control>='connect'
<inc>	int		0	Each frame's fill pattern value will be 'current_value + <inc>'. Only valid if <control>='connect'
<real-time>	int		0	If non-0, real-time mode enabled, 0 to disable. Only valid if <control>='connect'. See Note 3.
<nword>	int		100000	The number of 8-byte (!) words of fill pattern to generate. '-1' is 2^{64}-1, or ~2^{67} bytes (i.e. ∞)

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	connected | active | inactive	Current status of transfer
<hostname | filename>	char		Current destination of the fill pattern, either a <filename> from 'fill2file' or <hostname> from 'fill2net'
<byte#>	int		Amount of bytes generated so far

Notes:

1. If a valid data format has been set using 'mode=', data frames of this format will be generated. The first time stamp will be frame number 0 in the current UTC second of the O/S. If no data format has been configured ('mode = none ;'), then header-less blocks of data will be put on the network.
2. The data content of the frames/blocks can be a configurable constant or incrementing value for easy identification. Each time a new block (no format) or frame (valid data format) is generated, its contents will be filled with the current fillpattern value. After that, the fillpattern value will be modified by adding the value of <inc> to it. The fillpattern value is initialized with the value from <start>.
3. If <real-time> mode has been enabled, the sending system will try to generate data frames at a rate as close as it can to the configured data format's data rate. If <real-time> is '0' (disabled), the system will generate-and-send data as fast as the hardware will allow.

fill2net, fill2file

fill2disk/vbs – Record fill pattern on Mark5 disk pack or FlexBuff/Mark6 (*jive5ab* ≥ 2.8)

Command syntax: fill2[disk|vbs] = <on/off> : <scan label> [: <start> : <inc> : <real-time>] ;
Command response: !fill2[disk|vbs] = <return code>;

Query syntax: fill2[disk|vbs]? ;
Query response: !fill2[disk|vbs] ? <return code> : <status> : <scan label> : <byte#> ; (*jive5ab* = 2.8.0)
!fill2[disk|vbs] ? <return code> : <status> : <scan number> : <scan label> : <byte#> ; (*jive5ab* ≥ 2.8.1)

Purpose: Record dynamically generated synthetic data with valid time stamps on Mark5 disk pack or FlexBuff/Mark6.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	on | off		'on' – start recording 'off' – stop recording
<scan label>	char			A valid scan label
<start>	int | hex		0x11223344	Fill pattern start value
<inc>	int		0	Each frame's fill pattern value will be 'current_value + <inc>'
<real-time>	int		1	If non-0, real-time mode enabled, 0 to disable. See Note 3.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive	Current status of recording
<scan number>	int		Current scan number. Added in *jive5ab* 2.8.1 to comply with standard Mark5 `record?` reply format
<scan label>	char		Current scan name, only returned if <status> == active
<byte#>	int		Amount of bytes generated so far

Notes:

1. `fill2disk/fill2vbs` behave like `record=on/record=off` with no physical data source required. Combined with `tstat?` this can be used to do performance measuring or just generating some data to test `scan_check?` or any other data related operation.
2. If a valid data format has been set using 'mode=', data frames of this format will be generated. The first time stamp will be frame number 0 in the current UTC second of the O/S. If no data format has been configured ('mode = none ;'), then header-less blocks of data will be put on the network. **Note: fill2vbs *requires* a valid data format and will refuse to record if none has been set.**
3. The data content of the frames/blocks can be a configurable constant or incrementing value for easy identification. Each time a new block (no format) or frame (valid data format) is generated, its contents will be filled with the current fillpattern value. After that, the fillpattern value will be modified by adding the value of <inc> to it. The fillpattern value is initialized with the value from <start>.
4. If <real-time> mode has been enabled, the sending system will try to generate data frames at a rate as close as it can to the configured data format's data rate. If <real-time> is '0' (disabled), the system will generate-and-send data as fast as the hardware will allow.

fill2disk, fill2vbs

get_stats – Get disk performance statistics (query only)

Query syntax: get_stats? ;

Query response: !get_stats ? <return code> : < drive number> : <bin 0 count> : <bin 1 count> :….: <bin 7 count> : <replaced-block count> ~~: <SMART status>~~ ;

Purpose: Get detailed performance statistics on individual Mark 5 data disks

Monitor-only parameters:

Parameter	Type	Values	Comments
<drive number>	int		0=0M, 1=0S, 2=1M, 3=1S,….,14=7M, 15=7S
<bin 0 count>	int		Number of drive transactions falling in its bin 0 (see 'start_stats' command for explanation)
<bin 1 count>	int		Number of drive transactions falling in its bin 1
<bin 2 count>	int		Number of drive transactions falling in its bin 2
<bin 3 count>	int		Number of drive transactions falling in its bin 3
<bin 4 count>	int		Number of drive transactions falling in its bin 4
<bin 5 count>	int		Number of drive transactions falling in its bin 5
<bin 6 count>	int		Number of drive transactions falling in its bin 6
<bin 7 count>	int		Number of drive transactions falling in its bin 7
<replaced-block count>	int		Number of 65KB (actually 0xFFF8 bytes) data blocks unavailable on readback from this drive; these blocks have been replaced with fill pattern with even parity. See 'replaced_blks?' query for more information.
~~<SMART status>~~	~~char~~		jive5ab does **NOT** return this field. See Note 6.

Notes:

1. Each subsequent 'get_stats' query returns current performance statistics for the next mounted drive; recycles through mounted drives. Bin counts are not cleared. See details in Notes on 'start_stats' command.
2. The 'get_stats' query may not be issued during active recording or readback.
3. Drive statistics and replaced-block counts are cleared and re-started whenever a new disk module is mounted or a 'start_stats' command is issued.
4. The 8 bin counts in the 8 bins correspond to drive-response (transaction completion) times, with response time increasing from left to right. A good disk will have large numbers in bins 0 and 1 and small numbers (or 0) in the last few bins. See 'start_stats for additional information.
5. When operating in 'nb' mode, disks in banks A and B are treated as a single module.
6. DIMino returns the SMART status of the disk (OK, Fault) if it is SMART capable or the string NotSMART in case the drive is, well, not SMART capable. This field is not mentioned in the MIT Haystack Mark5B command set v1.12 after which `jive5ab` was modelled. It was chosen to stick to the documentation and not return an undocumented field.

get_stats

group_def – define/inspect aliases for set(s) of disks

Command syntax: group_def = <control> : <GRP> [: <pattern> [: <pattern> ....]] ;
Command response: !group_def = <return code>;

Query syntax: group_def? [<GRP>] ;
Query response: !group_def ? 0 : <GRP> [: <GRP>*] ;
!group_def ? 0 : <pattern> [: <pattern>*] ;

Purpose: Manage or query aliases of groups of patterns for disks (a "group definition").

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	define delete		'define' – add or replace a definition of the alias <GRP> to be the list of <pattern>s given 'delete' – remove the alias named <GRP>
<GRP>	char			Name of the alias to define or delete
<pattern>	char			Pattern (including shell wild cards) or other alias – see 'set_disks' for details

Monitor-only parameters:

Parameter	Type	Values	Comments
<GRP>	char		Optional name of group to list the definition for

Notes:

1. Using 'group_def=' it is possible to define an alias for a (sub)set of directories/disks. This alias can subsequently be used in 'set_disks=' to (quickly) select the indicated (sub)set of disks. The pattern(s) associated with an alias/group are resolved recursively and only expanded when used in a 'set_disks=' command. Since the 'set_disks=' command re-evaluates the actual mount points, the group definition pattern(s) are matched against disks mounted *at that time*.

group_def

in2file – Transfer data directly from Mark 5 input to file on disk

Command syntax: in2file = <control> [: <file name>,<file option>];
Command response: !in2file = <return code> ;

Query syntax: in2file? ;
Query response: !in2file ? <return code> : <last file name> : <status> [: <#bytes written> : <transfer sub status>] ;

Purpose: Control direct data transfer from Mark 5 input to file on disk; bypass (Mark5) disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | off | disconnect		'connect' – connect to socket on receiving Mark 5 system; initially, data transfer is off. 'on' – start/resume data transfer 'off' – pause data transfer 'disconnect' – stop transfer, close file
<file name>	char			Required only on first 'connect'; otherwise ignored
<file option>	char	n | w | a		Create new file ('n'), overwrite potential existing file ('w') or append to potential existing file ('a'). See Note 1.

Monitor-only parameters:

Parameter	Type	Values	Comments
<last file name>	char		File name given at last succesful 'in2file = connect : ..'
<status>	char	active | inactive	
<#bytes written>	int		Number of bytes written to the file since 'in2file = on ;'. Returned if <status> = 'active'
<transfer sub status>	char		Current sub state, 'WAIT', 'RUN', 'CONNECTED' or a combination thereof. Returned if <status> = 'active'

Notes:

1. In contrast to Mark5A/DIMino, *jive5ab* sometimes decided to aggretate/encode the file-open option ('n', 'w' or 'a') in the file name field. This is done by suffixing the file name with a construction "**,<file-open option>**". This was e.g. necessary for transfers where two files were included and for transfer modes not supported by Mark5A/DIMino. In transfers where Mark5A/DIMino documentation decides that the file-open mode character is a separate field, *jive5ab* follows this.
2. This transfer is mostly useful for debugging; to write a short section of raw telescope data to a file on disk. Unless the hard disks underlying the file storage are fast enough, 'bad things' might happen, see Note 3.
3. If the data rate is too fast for the file system to handle, the StreamStor FIFO will eventually overflow. An overflowing FIFO crashes the StreamStor firmware. *jive5ab* will **actively discard** data to keep the FIFO fill level under 60% in order to keep the firmware from crashing. If data is discarded, *jive5ab* will report on the terminal how much.

in2file

in2fork – Duplicate data from Mark 5 input to Mark 5 disks and network

Command syntax: in2fork = <control> [: <host> : <scan label>] ;
Command response: !in2fork = <return code> ;

Query syntax: in2fork ? ;
Query response: !in2fork ? <return code> : <last host>"f" : <status> [: <#bytes written> : <transfer sub status>] ;

Purpose: Control data transfer that duplicates Mark 5 input to Mark5 disks and a remote network destination

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | off | disconnect		'connect' – connect to socket on receiving Mark 5 system, create new scan on Mark 5 disks; data transfer is off. 'on' – start/resume data transfer 'off' – pause data transfer 'disconnect' – stop transfer, close file and scan on Mark 5 disks
<host>	char			Name or IPv4 address of the receiving system. Required only on first 'connect', ignored otherwise
<scan label>	char			Required only on first 'connect', ignored otherwise

Monitor-only parameters:

Parameter	Type	Values	Comments
<last host>	char		Host name given at last succesful 'in2fork = connect : ..' or 'in2net = connect : ..'. The suffix "f" is added to indicate that this was 'in2fork' rather than 'in2net'
<status>	char	active | inactive	
<#bytes written>	int		Number of bytes written to the file since 'in2fork = on ;'. Returned if <status> = 'active'
<transfer sub status>	char		Current sub state, 'WAIT', 'RUN', 'CONNECTED' or a combination thereof. Returned if <status> = 'active'

Notes:

1. Older StreamStor cards (notably the V100 and VXF2) are not capable to perform this duplication at data rates > 512Mbps. The firmware will corrupt the recording on disk when used above this data rate. **This directly called 'in2fork' transfer does not protect against data corruption.** If *jive5ab* is run in buffering mode (see command line arguments, Section **2**) it *does* prevent data corruption upon 'record=on' by automatically disabling the duplication if the data rate is too high for the detected hardware. **This transfer does not do that!**
2. If the data rate is too fast for the network to handle, the StreamStor FIFO will eventually overflow. An overflowing FIFO crashes the StreamStor firmware. *jive5ab* will **actively discard** data to keep the FIFO fill level under 60% in order to keep the firmware from crashing. If data is discarded, *jive5ab* will report on the terminal how much.

in2fork

in2mem – Transfer data directly from Mark 5 input to *jive5ab* internal buffer

Command syntax: in2mem = <control> ;
Command response: !in2mem = <return code> ;

Query syntax: in2mem? ;
Query response: !in2mem ? <return code> : <status> : <#bytes read> ;

Purpose: Control direct data transfer from Mark 5 input to *jive5ab* internal buffer; bypass disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	on | off		'on' – start copying data from I/O board into circular buffer in memory 'off' – end data transfer

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	inactive | active	
<#bytes read>	int		#bytes copied since the start of this transfer. Only returned if <status> = 'active'

Notes:

1. If data is written to the shared memory circular buffer in *jive5ab*, this data can be picked up from a different runtime for further processing, see e.g. 'mem2net = .. ;', 'mem2file = ..; '
2. The shared memory buffer can be read by multiple 'mem2*' transfers at the same time. Depending on the input data rate ('recording' data rate), the number of parallel 'mem2*' transfers and the system's resources (CPU, memory, disk) data may or may not be lost.

in2mem

in2memfork – Duplicate data from Mark 5 input to Mark 5 disks and *jive5ab* internal buffer

Command syntax: in2memfork = <control> [: <scan label>] ;
Command response: !in2memfork = <return code> ;

Query syntax: in2memfork ? ;
Query response: !in2memfork ? <return code> : <status> [: <#bytes written> : <transfer sub status>] ;

Purpose: Control data transfer that duplicates Mark 5 input to Mark5 disks and *jive5ab* internal buffer

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	on | off		'on' – create scan on Mark 5 disks and start data transfer immediately 'off' – stop data transfer
<scan label>	char			Required only on first 'connect'; otherwise ignored

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive	
<#bytes written>	int		Number of bytes written to the file since 'in2file = on ;'. Returned if <status> = 'active'
<transfer sub status>	char		Current sub state, 'WAIT', 'RUN', 'CONNECTED' or a combination thereof. Returned if <status> = 'active'

Notes:

1. This transfer underlies the *jive5ab* buffering mode (see command line options, section **2**) but is **most definitely not** it. This transfer does **NOT** prevent against data corruption if used with too high a data rate for the system's StreamStor card. **Do read the notes for 'in2fork'.**
2. If the data rate is too high for the system to handle, the StreamStor FIFO will eventually overflow. An overflowing FIFO crashes the StreamStor firmware. *jive5ab* will **actively discard** data to keep the FIFO fill level under 60% in order to keep the firmware from crashing. If data is discarded, *jive5ab* will report on the terminal how much.

in2memfork

in2net – Transfer data directly from Mark 5 input to network

Command syntax: in2net = <control> : <remote hostname> ;
Command response: !in2net = <return code> ;

Query syntax: in2net? ;
Query response: !in2net ? <return code> : <status> : <remote hostname> : <#bytes received> : <#bytes in buffer> ;

Purpose: Control direct data transfer from Mark 5 input to network; bypass disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | off | disconnect		'connect' – connect to socket on receiving Mark 5 system; initially, data transfer is off. 'on' – start data transfer 'off' – end data transfer 'disconnect' – disconnect socket See Notes with 'disk2net'
<remote hostname>	char		localhost	Required only on first 'connect'; otherwise ignored

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	inactive | connected | sending	
<remote hostname>			
<#bytes received>	int		#bytes received at the Input since 'connect' and while status is 'sending'
<#bytes in buffer>	int		#bytes remaining in buffer, waiting to be sent

Notes:

1. ~~Important: Due to current software problem, a scratch disk is required in Bank A for in2net operation; will be fixed in a future update.~~ *jive5ab* does not have this limitation; in2net can be run without any diskpacks present.
2. See Notes with 'disk2net' command for usage rules and restrictions.
3. If the data rate is too fast for the network to handle, the FIFO will eventually overflow; this will be reported by either a 'status?' query or an 'in2net?' query with an error message. An overflowing FIFO crashes the Streamstor firmware. *jive5ab* will data to keep the FIFO fill level under 60% in order to keep the firmware from crashing. If data is discarded, *jive5ab* will report on the terminal how much.
4. After 'in2net=off', but before 'in2net=disconnect', <#bytes received> shows the approximate total #bytes transferred from the input source; the #bytes currently sent out through the network is ~<#bytes received> minus <#bytes in buffer>. As <#bytes in buffer> drains to zero (as remaining data is sent out over the network), <#bytes received> becomes somewhat more precise.
5. If 'in2net=disconnect' is issued while <#bytes in buffer> is >0, data will be lost.
6. ~~For operation in special disk FIFO mode, see Section 7.~~ *jive5ab* has no disk FIFO mode but has the *fork* transfer, see Section 4.
7. Note that the network protocol parameters are set by the 'net_protocol' command.

in2net

ipd – Set packet spacing (inter-packet delay)

Command syntax: ipd = <packet spacing> ;
Command response: !ipd = <return code> ;

Query syntax: ipd? ;
Query response: !ipd ? <return code> : <packet spacing> ;

Purpose: Set packet spacing for UDP-based transfers

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<packet spacing>	int	≥ -1 [ns | us]	0	packet spacing in units of microseconds (no unit) or micro/nanoseconds if suffixed by the unit **us** or **ns**. Support for the **us** and **ns** suffixes appeared in *jive5ab* 2.5.0.

Monitor-only parameters:

Parameter	Type	Values	Comments
<packet spacing>	int/float	current packet spacing	Always returned in units of microseconds. See Note 7.

Notes:

4. When using UDP/IPv4 based transfers typically it is important to rate-control the sender in order not to incur excessive packet loss. Packet loss may originate from overwriting the socket buffer in the sender if packets are ingested at too high a rate. A typical case is when the data source bandwidth >> network bandwidth, as is sometimes with file transfers; data can be read from file faster than transferred over the network. Secondly, long-haul networks tend to respond badly to bursty traffic, especially if there are sections in the end-to-end link which have a different nominal bandwidth. It is not uncommon to have a trans-continental point-to-point connection which is built up of sections of 10Gbps, 2.5Gbps and/or other bandwidths. In general the best results with UDP/IPv4 based transfers is if the sender injects packets at a steady rate. *jive5ab* supports control over this parameter via this command: packets will be injected into the UDP transfer every <packet spacing> amount of microseconds.
5. The special setting "-1" means "automatic", or "theoretical" mode. From the current data format and track bit rate the total data rate is computed. Together with the configured MTU value this yields an optimal, theoretical, value for the packet spacing.
6. A value of "0" means that the packets are sent back-to-back.
7. This setting does not apply to the UDT protocol, even though it runs over UDP. In *jive5ab* >= 2.6.0, transfers supporting the UDT protocol should honour this value to implement rate limiting.
8. To give an idea: for 1Gbps links ipd ≈ 60-70us, at 4Gbps ipd ≈ 17us.
9. *jive5ab* implements this packet scheduling by busy-waiting. Depending on the amount of CPUs (notably the discrimination between >1 CPU or not) and the load on your system, the optimal ipd value for your system might differ from the auto/theoretical value. Some experimentation with this parameter may be necessary to find it.
10. Since *jive5ab* 2.5.0 support for ipd's of less than 1us is implemented. Upon query, if ipd < 1us, a float is returned in stead of an integer value. E.g."0.4" would be returned for and ipd that was set to 400ns.

layout – Get current User Directory format (query only)

Query syntax: layout? ;
Query response: !layout ? <return code> : <UD layout> ;

Purpose: Get exact details of the layout of the User Directory on Mark5 disk pack. This command appeared in *jive5ab* 2.5.0

Monitor-only parameters:

Parameter	Type	Values	Comments
<UD layout>	char		Canonical *jive5ab* User Directory format identifier. See Notes.

Notes:

1. The format of the User Directory ('Scan Directory') as written on the Mark5 disk packs has seen a number of versions through the years; some of them dependant on actual Conduant SDK version. *jive5ab* supports all layouts on all platforms for read back (e.g. 'scan_set='). Due to StreamStor firmware limitations not all disk packs can be appended to on all other systems. One notable example is that it is impossible to append to an SDK9-formatted disk pack on an SDK8-based Mark5. It is therefore of paramount importance to erase a disk pack on the actual system that will be recorded with.
2. Each time a new disk pack is mounted and detected (an activated disk pack at startup of *jive5ab* also counts as 'new mount') the layout foundon that disk pack is reported on *jive5ab*'s terminal. This command is a convenient way of retrieving this information remotely.
3. The canonical *jive5ab* User Directory format identifier is a human- as well as machine readable string. Examples are: `Mark5A8DisksSDK8`, `Mark5B16DisksSDK9`. Please refer to Mark5 Memo #100[2] for intricate details plus full description.

[2] http://www.haystack.mit.edu/tech/vlbi/mark5/mark5_memos/100.pdf

layout

mode – Set data recording/playback mode (Mark5A, A+)

Command syntax: mode = <data mode> : <data submode> : [<output data mode> : <output submode>] ;
Command response: !mode = <return code> ;

Query syntax: mode? ;
Query response: !mode ? <return code> : <data mode> : <data submode> : <output mode> : <output submode> : <sync status> : <#sync attempts> ;

Purpose: Set the recording and playback mode of the Mark 5 I/O card or *jive5ab's* internal data format.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<data mode>	char	mark4 | vlba st | tvg | mark5a+n none 'magic mode'	st	'mark4' or 'vlba': strips and restores parity bits. 'st' ('straight-through') mode records 32 input 'tracks' directly 'tvg' – takes data from internal TVG – see Note 8. A null field is special case for correlator. See Note 5. For Mark 5A+ operation, 'n' is track-map # to be used; see Note 14. 'none' – unknown format/don't care, see Section 6 'magic mode'; see Note 15
<data submode>	char	8 | 16 | 32 | 64 | mark4 | vlba	See Note 2 and Note 14	8,16,32,64 is relevant only for 'mark4', 'vlba' and 'mark5a+' modes and corresponds to number of tracks. 'mark4' and 'vlba' relevant only for 'st' mode. Not relevant when <data mode> is 'tvg'. A null field is special case for correlator. See Note 5.
<output mode>	char	mark4 | vlba | st	<data mode>	Optional: For correlator or diagnostic use only: Forces the Output Section of the Mark 5A I/O board into specified mode and submode independently of the Input Section – see Note 5.
<output submode>	char	8 | 16 | 32 | 64 | mark4 | vlba	<data submode>	Optional: For correlator or diagnostic use only – see Note 5.

Monitor-only parameters:

Parameter	Type	Values	Comments
<sync status>	char	s | -	's' indicates Output Section of I/O board is sync'ed; '-' indicates not sync'ed. See Note 9
<#sync attempts>	int		Number of sync attempts by output section. Relevant only for 'mark4' and 'vlba' modes only. See Note 10.

mode/5A, A+

Notes:

1. The 'mode=' command sets both the input and output modes to be the same unless overridden by <output data mode> and <output submode> parameters or the format is not supported by the hardware but is by *jive5ab (see section 7 of this document)*.
2. Power-on default <data mode>:<data submode> is 'st:mark4'. For <data mode> of 'st', default <data submode> is 'mark4; for <data mode> of 'mark4' or 'vlba', default <data submode> is '32'.
3. In 'mark4' or 'vlba' mode, the Mark 5A strips parity on record and restores it on playback to save storage space. If the number of tracks is 8, 16 or 64, the Mark 5 I/O does the necessary multiplexing/demultiplexing to always fully utilize all FPDP 32 bit streams driving the disk array. In 'st' ('straight-through') mode, the input data are recorded and played back with no processing.
4. **In 'mark4:xx' mode, the station ID (set by jumpers in the Mark 4 DAS rack) must be an even number**. Attempting to record in 'mark4' mode with an odd station ID will result in an error. This is due to the fact that, with parity stripped, an odd station ID considerably complicates the job of properly recovering synchronization during playback, and is therefore not allowed.
5. At a correlator, where there is normally nothing connected to the Mark 5A input, it is suggested that the desired playback mode be specified in <output mode> and <output submode> and that <data mode> and <data submode> both be null fields. This will cause the input section of the I/O board to be set to default ('st:mark4') mode and prevents spurious error messages from appearing regarding the input station ID.
6. The only reason to distinguish between 'st:mark4' and 'st:vlba' modes is to allow the 'play_rate' command' to properly set the internal clock generator for a specified data rate; the setting is slightly different for the Mark4 and VLBA cases.
7. The tracks expected from a Mark4 or VLBA formatter in the various modes are as follows:

	Recorded formatter track#'s	**FPDP bit streams**
'mark4:8' or 'vlba:8'		Trk 2 to FPDP streams 0,8,16,24; trk 4 to 1,9,17,25; etc.
'mark4:16' or 'vlba:16'	2-33 even (headstack 1)	Trk 2 to FPDP streams 0,16; trk 4 to 1,17; etc.
'mark4:32' or 'vlba:32'	2-33 all (headstack 1)	Correspond to FPDP bit streams 0-31, respectively
'mark4:64' or 'vlba:64'	2-33 (headstacks 1 and 2)	Trks 2 from both hdstks mux'ed to FPDP bit stream 0, etc.
'st' (any submode)	2-33 all (headstack 1)	Correspond to FPDP bit streams 0-31, respectively

8. In all modes except 'tvg' mode, the data clock is provided by the external data source. In 'tvg' mode, the clock-rate is set by 'play_rate' command.
9. The 'sync status' parameter is relevant only in output mode 'mark4' or 'vlba' where parity must be restored. If 'sync'ed', the I/O board has properly synchronized to the data frames and is properly de-multiplexing and restoring parity.
10. The '# of sync attempts' returned value in the 'mode=' command counts the number of sync attempts the Mark 5A I/O board output section had to make before parity-stripped data ('mark4' or 'vlba') was re-sync'ed, as necessary for parity re-insertion. A large number indicates a problem, perhaps in the output clock or the data itself. The counter is reset to zero on a subsequent 'mode=' command.
11. If in 'tvg' mode, TVG is operated at clock-rate set by 'play_rate' command.
12. When <data mode> is 'mark4' or 'vlba', the NRZM output coding present on the data received from the formatter is converted to NRZL for transmission to the FPDP bus, and hence for recording on disk or transmission over a network. On output, the inverse operation (conversion back to NRZM) is performed, so that the output data on the Mark5A I/O Panel are in the NRZM format for input to the correlator. When operating in <st> or <tvg> mode, no coding conversions are done.
13. When operating in tvg mode, the 32-bit-wide tvg pattern is written directly to the disk with no tape-frame headers or synchronization information of any sort. Furthermore, the tvg runs continuously with no forced resets unless an external 1pps signal is connected to J11 on the Mark 5A I/O board, in which case the tvg is asynchronously reset on each 1pps tick.
14. Operation in Mark 5A+ mode allows a Mark 5B disk module to be read on the Mark 5A and create VLBA output tracks; the Mark 5B data are read, transformed into VLBA track format (including the addition of parity bits), and VLBA-format headers are inserted. The Mark 5A must have the proper Xilinx code and software upgrades installed for this mode to work. The track mapping option ('n' in 'mark5a+n'), as well as the number of output tracks, must be specified. Details are given in Mark 5 memo #39 available at http://www.haystack.edu/tech/vlbi/mark5/memo.html. Note that the VLBA auxiliary data field will be identically zero for all Mark 5A+ playback.
15. 'magic mode' is a one-string-sets-all version of the mode command. The string is parsed and sets the data frame format, number of tracks and track bitrate all in one go. The string follows the Walter Brisken/DiFX mk5access library canonical format as described in Section 7.1 of this manual.

mode/5A, 5A+

mode – Set data recording mode (Mark5B, B+ /DIM)

Command syntax: mode = <data source> : <bit-stream mask> : [<decimation ratio>] : [<FPDP mode>] ;
Command response: !mode = <return code> ;

Query syntax: mode? ;
Query response: !mode ? <return code> : <data source> : <bit-stream mask> : <decimation ratio> : <FPDP mode> ;

Purpose: Set the recording mode of the Mark 5B DIM

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<data source>	char	ext | tvg | ramp |none| 'magic mode'	ext	'ext' – data on VSI 80-pin connector 'tvg' – internal Test Vector Generator; see Note 1 'ramp' – internal ramp generator; see Note 2 'none' – unknown format/don't care, see section 6. 'magic mode', see Note 4
<bit-stream mask>	hex64	See Note 1	0xffffffff	5B format: up to 64 bitstreams!
<decimation ratio>	int	1, 2, 4, 8, 16	1	Specifies ratio of data clock to recorded sample rate; if <data source> is 'tvg' or 'ramp', value of '1' should always be used - see Notes 1 and 2
<FPDP mode>	int	1 | 2	See Note 3	*For diagnostic use only*: Sets FPDP mode (FPDP1 or FPDP2). See Note 3

Monitor-only parameters:

Parameter	Type	Values	Comments
<data source>	char		
<bit-stream mask>	hex64		
<decimation ratio>	int		
<FPDP mode>	int	1 | 2	

Notes:

1. As per the VSI-H specification, the tvg pattern resets at every second tick. The bit-stream mask selects which bits of the tvg pattern are recorded; a decimation value other than '1' should not be used except for special diagnostic testing as the resulting recorded data may not be recognized as tvg pattern by 'data_check' or 'scan_check'. The bit-stream mask has been extended to support 64 bit-streams, necessary for 4Gbps Mark5B or non-dechannelized VDIF format.
2. A 'ramp' pattern replaces the tvg with a 32-bit counter that starts at zero on the next second tick and increments each clock tick for 100 seconds before resetting to zero again. The bit-stream mask selects which bits of the ramp pattern are recorded; a decimation value other than '1' should not be used except for special diagnostic testing as the resulting recorded data may not be recognized as a ramp pattern by 'data_check' or 'scan_check'.
3. Mark 5B only supports FPDP1; any attempt to set to FPDP2 will cause an error. Mark 5B+ always defaults to FPDP2, but may be forced to FPDP1 for test purposes; maximum aggregate date rate for FPDP1 is 1024Mbps - an attempt to record at 2048Mbps in FPDP1 will cause error.
4. 'magic mode' is a one-string-sets-all version of the mode command. The string is parsed and sets the data frame format, number of tracks and track bitrate all in one go. The string follows the Walter Brisken/DiFX mk5access library canonical format as described in Section 7.1 of this manual.

mode/5B, 5B+ DIM

mode – Set data recording/playback mode (Mark5B DOM, Mark5C, generic)

Command syntax: mode = <data mode> : <data submode> : [<extra info>] ;
Command response: !mode = <return code> ;

Query syntax: mode? ;
Query response: !mode ? <return code> : <data mode> : <data submode> : <extra info>

Purpose: Set the *jive5ab* internal format. This is a union of the Mark5A and Mark5B/DIM and Mark5C "mode=" commands. Nothing will be sent to the I/O board because, typically, on these systems there isn't any to program.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<data mode>	char	mark4 |vlba|st tvg |ramp unk|mark5b ext none 'magic mode'	st	'mark4' or 'vlba': expect parity bit stripped MarkIV or VLBA data. 'st' ('straight-through') mode records 32 input 'tracks' directly 'tvg' , 'ramp' expect Mark5A or Mark5B (ramp) I/O board generated TVG pattern 'unk' – on Mark5C this translates to 'none', 'mark5b' + 'unk' only supported on Mark5C 'ext' – expect Mark5B formatted data 'none' – unknown format/don't care, see Section 6 'magic mode', see Note 1
<data submode>	char	8 | 16 | 32 | 64 | mark4 | vlba|int hex64		8,16,32,64 is relevant only for 'mark4', 'vlba' and modes and corresponds to number of tracks. 'int' is VDIF number of bitstreams; see section 6.1. 'hex64' is the 64-bit bit-stream mask for 'ext' Mark5B formats (supports up to 4Gbps Mark5B)
<extra info>	int	modulo 8	(empty)	'vdif' or 'legacyvdif' - VDIF payload (data array) size, must be multiple of eight

Monitor-only parameters:

Parameter	Type	Values	Comments
<data mode>	char		reflects *jive5ab*'s thoughts on what the data format is
<data submode>	mixed		filled in as appropriate for <data mode>
<extra info>	mixed		filled in as appropriate for <data mode>

For format specific notes, refer to the respective Mark5A, Mark5B/DIM "mode" commands. Which notes apply is, obviously, a function of which format is attempted to set.

Notes:

1. 'magic mode' is a one-string-sets-all version of the mode command. The string is parsed and sets the data frame format, number of tracks and track bitrate all in one go. The string follows the Walter Brisken/DiFX mk5access library canonical format as described in Section 7.1 of this manual.

mode/5B DOM, Mark5C, generic

mem2file – Transfer data from *jive5ab* internal buffer to file on disk

Command syntax: mem2file = <control> : <filename> : [<#buffer buffer>] : [<file option>] ;
Command response: !mem2file = <return code> ;

Query syntax: mem2file? ;
Query response: !mem2file ? <return code> : <status> : <#bytes written> ;

Purpose: Control data transfer from *jive5ab* internal buffer to file on disk

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	on | stop | off		'on' – start data transfer, subject to availability, see Note 1. 'off' – stop transfer immediately, discarding bytes currently in buffer 'stop' – initiate stop of transfer, flush all buffered bytes to disk. See Note 3
<filename>	char			Required only on first 'on'; otherwise ignored
<#buffer bytes>	int		256MB	Size of snapshot buffer between *jive5ab* internal buffer and file writer. See Note 2
<file option>	char	n | w | a	n	Create new file ('n'), truncate existing file ('w') or append to existing file ('a')

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	inactive | active | flushing | done	
<connection status>	char		Current connection state, 'WAIT', 'RUN', 'CONNECTED' or a combination thereof. Only returned if <status> != 'inactive'

Notes:

1. This transfer can be started independently from a '*2mem' transfer. After 'mem2file = on ;' still no data may be transferred; data flow will start as soon as data appears in the internal buffer. If a '*2mem' transfer is already writing data in the internal buffer, this transfer will start immediately after 'mem2file = on : .. ;'
2. In order to support writing to slow disks, a separate 'snapshot' buffer is used by this transfer. This allows for the real-time capture of <#buffer bytes> from the internal buffer whilst the file may be written to at a much lower speed. After the snapshot buffer is full, no data will be read from the *jive5ab* internal buffer until space becomes available due to bytes being written to disk.
3. A 'mem2file = stop' will instruct the transfer to stop reading from *jive5ab*'s internal buffer. The transfer remains alive, flushing all currently buffered bytes in the snapshot buffer to disk. The command will return immediately with <return code> equal "1" ("action initiated but not completed"). Subsequent queries will return a <status> of 'flushing' until all bytes have been flushed. Then <status> will transition to 'done'. Issue 'mem2file = off ;' to clear the status to 'idle'.

mem2file

mem2net – Transfer data from *jive5ab* internal buffer to network

Command syntax: mem2net = <control> : <remote hostname> ;
Command response: !mem2net = <return code> ;

Query syntax: mem2net? ;
Query response: !mem2net ? <return code> : <status | remote hostname> : <connection status> ;

Purpose: Control data transfer from *jive5ab* internal buffer to network

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	connect | on | disconnect		'connect' – connect to socket on receiving Mark 5 system; initially, data transfer is off. 'on' – start data transfer, subject to availability, see Note 1 'disconnect' – disconnect socket See Notes with 'disk2net'
<remote hostname>	char		localhost	Required only on first 'connect'; otherwise ignored

Monitor-only parameters:

Parameter	Type	Values	Comments
<status | remote hostname>	char	inactive | <remote hostname>	
<connection status>	char		Current connection state, 'WAIT', 'RUN', 'CONNECTED' or a combination thereof. Only returned if <status> != 'inactive'

Notes:

1. This transfer can be started independently from a '*2mem' transfer. After 'mem2net = on ;' still no data may be transferred; data flow will start as soon as data appears in the internal buffer. If a '*2mem' transfer is already writing data in the internal buffer, this transfer will start immediately after 'mem2net = on ;'

mem2net

mem2time – Decode data from *jive5ab* internal buffer into queryable time stamp

Command syntax: mem2time = <control> ;
Command response: !mem2time = <return code> ;

Query syntax: mem2time? ;
Query response: !mem2time ? <return code> : inactive ;
!mem2time ? <return code> : O/S : <current O/S time> : data : <current data time> : <delta O/S – data> ;

Purpose: Monitor data stream coming into memory by continuously decoding data frames and retaining the last time stamp

Settable parameters:

Parameter	Type	Allowed values	Comments
<control>	char	open | close	'open' – start data transfer (subject to availability, see Note 1) and decoding (subject to format, see Note 2), 'close' – stop transfer

Monitor-only parameters:

Parameter	Type	Comments
<current O/S time>	time	Current O/S time
<current data time>	time	Time stamp of last succesfully decoded data frame
<delta O/S - data>	float "s"	Time difference between O/S and data, in seconds. The unit string "s" is appended.

Notes:

1. This transfer can be started independently from a '*2mem' transfer. After 'mem2time = open ;' still no data may be decoded; decoding will start as soon as data appears in the internal buffer. If a '*2mem' transfer is already writing data in the internal buffer, this transfer will start immediately after 'mem2time = open ;'
2. For this transfer to succesfully decode the incoming data stream, the runtime in which this transfer is started should have its "mode = " be set to the expected data format. Refer to "mode = .." documentation for setting the mode in a convenient way. Note that the runtimes also support setting a 'hardware' format. E.g. to set Mark5B data format "mode = ext : 0xffffffff ; clock_set = 32 : int;" could be used – equivalent to how a hardware Mark5B would be configured.
3. If data is being decoded, the last decoded data time stamp can be retrieved using 'mem2time?'. Repeated querying of 'mem2time?' would allow an inquisitive application to monitor the data stream.

mem2time

mtu – Set network Maximum Transmission Unit (packet) size

Command syntax: mtu = <MTU> ;
Command response: !mtu = <return code> ;

Query syntax: mtu? ;
Query response: !mtu ? <return code> : <MTU> ;

Purpose: Set network MTU for subsequent UDP based data transfers

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<MTU>	int	64 - 9000	1500	See Notes

Monitor-only parameters:

Parameter	Type	Values	Comments
<MTU>	int	<mtu>	Current value of MTU

Notes:

1. The actual MTU value is only used for determining the (maximum) packet size that *jive5ab* can send in a subsequent UDP/IPv4 network transfer. *jive5ab* uses this value as a maximum value, ***unrelated*** to the MTU actually configured on the ethernet device! In certain cases this may lead to a failure of data passing through the interface, notably if *jive5ab's* idea of the MTU is > the MTU configured on the device.
2. The limits mentioned in the allowed values are enforced by *jive5ab*; these limits are typical (hard) ethernet limits.
3. From the MTU and the actual network protocol and configured data format (and optional configured channel dropping) *jive5ab* computes the size of the packet payload section, taking various constraints into account. Example constraints are: payload size must be multiple of eight, an integral number of packets fitting into a <workbuf size> (see 'net_protocol', p.50), losing a compressed packet does not loose sync in the data stream to name but a few. These constraints are not solved until a transfer is actually started. To review the values actually used in a running transfer see the "constraints?" query, p. .
4. The packets on the wire are guaranteed to have a size ≤ the set MTU.
5. This setting also applies to the UDT protocol because it is based on UDP and as such it can be told to honour this size.

net2disk – Transfer data from network to disks

Command syntax: net2disk = <control> : <scan label> [: ~~<bit-stream mask>~~ <ip | host>];
Command response: !net2disk = <return code> ;

Query syntax: net2disk? ;
Query response: !net2disk ? <return code> : <status> : <scan#> : <scan label> : <bit-stream mask> ;

Purpose: Enable data transfer from network to local disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	open | close		'open' or 'close' socket
<scan label>	literal ASCII			Scan label to be assigned to this data; if not specified, defaults to 'EXP_STN_net2disk' See Section 6 for format of scan label.
~~<bit-stream mask>~~ <ip/host>	~~hex~~ ASCII		~~0~~	~~<bit-stream mask> associated with data. See Note 1.~~ Host name or IPv4 address, see Note 5.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive | waiting	Current status of transfer
<scan#>	int		Sequential scan number on disk module
<scan label>	ASCII		Assigned scan label
~~<bit-stream mask>~~ <nbytes>	~~hex~~ int		~~Assigned bit-stream mask~~ Number of bytes written to disk (FIFO, in reality)

Notes:

1. ~~The <bit-stream> mask should always specified so that the Mark 5B directory entry can be properly completed.~~ See Note 5.
2. See Notes with 'disk2net' command for usage rules and restrictions.
3. When <status> is 'inactive', a 'net2disk?' query returns <scan label> of the last transferred scan, if any.
4. Note that the network protocol parameters are set by the 'net_protocol' command.
5. The <bit-stream mask> is not used by *jive5ab*. This parameter is now an optional IPv4 address or host name which will be used to connect to in case the reverse-TCP protocol is selected (see "net_protocol"); it reverses client/server roles.

net2disk

net2file – Transfer data from network to file on disk

Command syntax: net2file = <control> : <file name>,<file option> [: <strictness>] ;
Command response: !net2file = <return code> [: <file size>] ;

Query syntax: net2file? ;
Query response: !net2file ? <return code> : <status> : <#bytes written> ;

Purpose: Control data transfer from network to file on disk.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	open | close		'open' – set up a receiver for incoming data, using current net_protocol and net_port values. *jive5ab* >= 2.6.2 returns file size in the reply to implement m5copy resume capability 'close' – stop the transfer; close connection and file
<file name>	char			Required only on first 'open'; otherwise ignored
<file option>	char	n | w | a	n	Create new file ('n'), truncate existing file ('w') or append to existing file ('a'). See Note 1.
<strictness>	int	[1,] 2	0	Strictness level of decoding incoming frames. See Note 2.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	inactive | active	
<#bytes written>	int		Amount of bytes written to disk

Notes:

1. In contrast to Mark5A/DIMino, *jive5ab* sometimes decided to aggretate/encode the file-open option ('n', 'w' or 'a') in the file name field. This is done by suffixing the file name with a construction "**,<file-open option>**". This was e.g. necessary for transfers where two files were included and for transfer modes not supported by Mark5A/DIMino. In transfers where Mark5A/DIMino documentation decides that the file-open mode character is a separate field, *jive5ab* follows this.
2. If a valid data format has been set via "mode=" *jive5ab* can be instructed to decode incoming data frames and filter only the valid ones. The strictness parameter determines how strict the filtering will be done. Future releases may support finer grained control than on/off. Currently, the value of '1' is allowed by the code but will have no effect; only a value >1 is currently meaningful

strict	Filtering characteristics
0, 1	Look for syncword to find data frames. Check for Mark5B consistency. Allow DBE Mark5B frames; they're broken (no VLBA time stamp)
2	All of strict = 0 plus CRC checks for data formats that support that. Header must strictly comply to format's definition.

net2file

net2mem – Transfer data from network to *jive5ab* internal buffer

Command syntax: net2mem = <control> ;
Command response: !net2mem = <return code> ;

Query syntax: net2mem? ;
Query response: !net2mem ? <return code> : <status> ;

Purpose: Control data transfer from network to *jive5ab* internal buffer

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	open | close		'open' – set up a receiver for incoming data using current net_protocol and net_port values 'close' – stop the transfer; close connection

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	inactive | active	

Notes:
1. -

net2mem

net2out – Transfer data directly from network to Mark 5 output

Command syntax: net2out = <control> ;
Command response: !net2out = <return code> ;

Query syntax: net2out? ;
Query response: !net2out ? <return code> : <status> : <nowbyte> ;

Purpose: Enable data transfer from network to Mark 5 output; bypass disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	open | close	-	'open' or 'close' socket

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive | waiting | paused	active – connected and data flowing inactive – no socket open; doing nothing waiting – socket open and waiting for a connection paused – socket connected but no data waiting to be read See Note 3.
<nowbyte>	int	-	Total number of bytes transferred to the output since 'open'; the 'open' command resets <nowbyte> to 0.

Notes:

1. See Notes with 'disk2net' command for usage rules and restrictions.
2. Note that the network protocol parameters are set by the 'net_protocol' command.
3. Note that none of the status returns necessarily indicate an error – depends on context. At modest data-transfer speed, <status> may be "paused" (indicating no data waiting to be read) most of the time even if <nowbyte> is incrementing; incrementing <nowbyte> indicates that data are flowing.

net2out

net_port – Set IPv4 port and optional local IPv4 address for the data channel (*jive5ab* ≥ 3.0.0)

Command syntax: net_port = [<ip|host>@]<port> ;
Command response: !mtu = <return code> ;

Query syntax: net_port? ;
Query response: !net_port ? <return code> : [<ip|host>@]<port> ;

Purpose: Set the port number and optional local address for network transfers to connect/listen, and send/receive data.Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<ip|host>@	char		none	Resolvable host name or dotted-quad IPv4 address (see Notes)
<port>	int	0- 65536	2630	See Notes

Monitor-only parameters:

Parameter	Type	Values	Comments
<ip|host>@	char	<ip|host>	Current value of the local IPv4 or hostname configured. Only reported if one is set.
<port>	int	<port>	Current value of the data port

Notes:

1. The `*2net` and `net2*` transfers involve a network client and server component respectively. The sender (client) connects to a PROTOCOL:IPv4:PORT combination, the receiver (server) listens on PROTOCOL:*:PORT (all interfaces on the machine). This command can be used to program the PORT part of the connection; the PROTOCOL and IPv4 parts are set by 'net_protocol=' and '*2net = connect : ...' commands. The compiled-in default for this value is 2630 and is compatible with MIT Haystack Mark5A, DIMino and drs.
2. This parameter also dictates which port number UDP data streams will be captured from for recording on Mark6, FlexBuff or during real-time e-VLBI. Even though UDP is connectionless, the receiver (server) has to listen on "udp:*:port" for incoming UDP data packets. Because this parameter is stored per runtime, data streams sent to different ports must be captured in different runtimes.
3. The limits mentioned in the allowed values are governed by internet standards. The port number is a 16-bit unsigned integer value limiting the physical values from 0 to 65535 (2^{16}-1). Besides these numerical limits, the internet standard qualifies some numerical port ranges differently. E.g. for a process to be allowed to start a server on a port number <= 1024, root privilege is required. Also it is important to realize that port selection may not be arbitrary: many ports are registered to specific protocols. The default 2630 was meant to be registered at the Internet Assigned Numbers Authority (IANA) (ICANN since 2001) but that never materialized. The current list of registered ports can be found at the following URL: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.
4. Since *jive5ab* 3.0.0 the `net_port` command supports specifying an optional **<ip|host>@** prefix before the port number. This is mainly useful for FlexBuff/Mark6 systems to force *jive5ab*, after `record=on:...` to open a listening socket only on that address, in stead of listening on all interfaces, which is the default. Using this feature, multiple parallel streams can easily be recorded by different runtimes (see Section 6). The **<ip>** address may be a multicast address, after which *jive5ab* joins that group.
5. Sending a `net_port=` command with only a `<port>` parameter resets the currently defined local address to all interfaces.

net_port

net_protocol – Set network data-transfer protocol and I/O block sizes for all transfers

Command syntax: net_protocol = <protocol> : [<socbuf size>] : [<workbuf size>] : [<nbuf>] ;
Command response: !net_protocol = <return code> ;

Query syntax: net_protocol? ;
Query response: !net_protocol? <return code> : <protocol> : <socbuf size> : <workbuf size> : <nbuf> ;

Purpose: Set network data-transfer protocol and I/O block sizes for all transfers

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<protocol>	char	tcp | unix rtcp|udp|udps| pudp|udt|udpsnor	tcp	Select which network protocol to use for net2* and *2net transfers. See Note 5. The udpsnor protocol was added in *jive5ab* 2.8. See Note 9.
<socbuf size>	int	bytes (see Note 3)	Linux socket buffer size	Used as receive buffer size in ~~'net2out' and 'net2disk'~~ net2*. Used as send buffer size in ~~'in2net' and 'disk2net'~~ *2net. Defaults to ~~0~~ 4MB ~~which causes use of Linux OS defaults buffer size~~.
<workbuf size>	int	bytes (see Note 3)	131072 or last value set	~~buffer size to send to data sockets in 'in2net' and 'disk2net'~~. The basic unit of I/O in any transfer. See Note 4. Also very important for Mark6/FlexBuff, see Note 7.
<nbuf>	int	1~~-16~~	8	~~Number of blocks, each of size <workbuf size>, allocated in a circular FIFO; see Note 2~~ The usage of this parameter has changed completely. See Note 6. Also important for Mark6/FlexBuff, see Note 8.

Notes:

1. Query returns protocol and buffer sizes currently in force.
2. ~~<nbuf> times <workbuf size> must not exceed 134,217,728 bytes.~~ See Note 8.
3. For your convenience the suffixes "k" (value x 1024 bytes) and "M" (value x 1024k bytes) are supported
4. The <workbuf size>, if supplied in a command, will be taken as a hint and at least be made a multiple of eight. It is the basic unit of I/O in **any** transfer. E.g. for "disk2net" the data will be read from the disk pack in chunks of size <workbuf size>. Disk pack I/O transfer speed reaches it maximum at a size of 4MB. With (very) small <workbuf size> blocks the overhead will be huge and performance severely affected.
5. *jive5ab* supports various network protocols. For the unix protocol the format of the destination in the "connect" string is the path to a local file. For all other protocols it is an IPv4 dotted quad notation address or a resolvable internet host name.

tcp, unix	the standard UNIX network protocols
udp, udps, udpsnor	these are synonyms – udps means: UDP + 8-byte sequence number ("net_protocol=udps" + "mode=vdif" ⇒ VTP) for udpsnor see Note 9.
pudp	"plain UDP" for vanilla, UNIX style, UDP protocol without 8-byte sequence number
rtcp	reverse TCP – identical to TCP but with reversed connection direction. Use in case of firewall issues: *2net becomes server, net2* becomes client.
udt	High performance reliable protocol layered on top of UDP. See http://udt.sourceforge.net/

6. The processing chains mentioned in Section 4 of this manual yield their data from one processing step to the next through a queue. The depths of these queues are fixed at compile time, typically rendering this parameter meaningless but in one – important – place. When the

net_protocol

UDPs (a.k.a. VTP, UDP-with-sequence number) protocol is used, this protocol keeps <nbuf> <workbuf size>'d blocks in cache as 'read ahead' buffer such that it can deal with reordering.

7. On Mark6 and FlexBuff the <workbuf size> value sets the size of the blocks. On Mark6 a block of size <workbuf size> is written to the next available file; on FlexBuff, files of size <workbuf size> will be written. MIT Haystack c-plane/d-plane uses 10000000 ($10x10^6$ bytes) byte blocks. Useful values for Mark6 recordings are between 8 – 16MB. For FlexBuff a minimum file size of 128MB is enforced by *jive5ab*.
8. For a number of transfers, *jive5ab* will pre-allocate <nbuf> * <workbuf size> bytes when a transfer is started. It is therefore advised to keep the value of this product within reasonable limits – pre-allocating too little (say 8 buffers of 16kB) or too much (64 buffers of 512MB (=32GB!)) should be avoided. Reasonable values typically depend on the target data rate, the system's speed and amount of installed memory. Some experimentation may be required to fine-tune the combination of <nbuf> and <workbuf size> although this is expected to be necessary only in edge cases e.g. pushing a system close to its performance limits.
9. In order to support recording multi-stream FiLa10G or RDBE data, the ***udpsnor*** protocol was introduced in *jive5ab* 2.8. "udpsnor" is short for "UDP with sequence number, no reordering". FiLa10G and RDBE output VDIF frames with a 64-bit sequence number prepended. Throwing away the sequence numbers would be an option, but *jive5ab* ≥ 2.8 recording ***udpsnor*** keeps sequence number statistics from up to eight (8) individual senders. All data received on the configured UDP port number (see `net_port=`) will be recorded in order of delivery to *jive5ab* in stead of it attempting to reorder the frames based on sequence number. Thus udpsnor is great for local recording: it is simpler but still gives valuable and accurate packet statistics on-the-fly (see documentation for the `evlbi?` query).

net_protocol

OS_rev – Get details of operating system (query only)

Query syntax: OS_rev? ;
Query response: !OS_rev? <return code> : <OS field1> : <OS field2>: ….. : <OS fieldn> ;

Purpose: Get detailed information about operating system.

Monitor-only parameters:

Parameter	Type	Values	Comments
<OSfield1> through <OSfieldn>	literal ASCII		Primarily for diagnostic purposes. The character stream returned from OS, which is very long, is divided into 32-character fields separated by colons to stay within Field System limits. See Notes.

Notes:

1. ‘OS_rev?’ is a replacement for the old ‘OS_rev1?’ and ‘OS_rev2?’ queries; all three of these queries are now synonyms.

packet – Set/get packet acceptance criteria

Command syntax: packet = <DPOFST> : <DFOFST> : <length> : <PSN Mode> : <PSNOFST> ;
Command response: !packet = <return code>;

Query syntax: packet? ;
Query response: !packet? <return code> : <DPOFST> : <DFOFST> : <length> : <PSN Mode> : <PSNOFST> ;

Purpose: Set / get the packet acceptance criteria.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
< DPOFST >	int	≥ 0	0	payload byte offset from beginning of payload to first recorded data
< DFOFST >	int	≥ 0	0	payload byte offset to beginning of recording
< length >	int	> 0	5008	number of bytes to record per packet (see Note 1)
< PSN Mode >	int	0 | 1 | 2	0	Packet Serial Number (PSN) monitor mode (see Note 2)
< PSNOFST >	int	≥ 0	0	payload byte offset from beginning of payload to PSN (for PSN monitor mode 1 or 2)

M onitor-only parameters:

Parameter	Type	Values	Comments
< DPOFST >	int	≥ 0	payload byte offset from beginning of payload to first recorded data
< DFOFST >	int	≥ 0	payload byte offset to beginning of recording
< length >	int	> 0	number of bytes to record per packet (see Note 1)
< PSN Mode >	int	0 | 1 | 2	Packet Serial Number (PSN) monitor mode (see Note 2)
< PSNOFST >	int	≥ 0	payload byte offset from beginning of payload to PSN (for PSN monitor mode 1 or 2)

Notes:

1. The length of data to be recorded must be a multiple of 8 bytes.
2. PSN-monitor 0 mode will disable packet serial number checking and record all data in the order received. PSN-monitor mode 1 will replace invalid packets with the specified fill pattern and guarantee order. PSN-monitor mode 2 will prevent packets from being written to disk if the most significant bit is set.

packet

personality – Set/get personality (available on 5A/5B with *jive5ab* ≥ 2.8)

Command syntax: personality = <type> : <root > ;
Command response: !personality = <return code> ;

Query syntax: personality? ;
Query response: !personality? <return code> : < type >: < root > ;

Purpose: Set / get the application personality (i.e., the emulation mode).

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
< type >	char	mark5C | ~~file~~	NULL	Mark5C– Normal operating mode (default) ~~file – write data to the OS system disk, not streamstor (see Note 3)~~ Supported by *jive5ab* indirectly, see Note 5 ***Must*** be left empty when issued to a Mark5A or Mark5B system (*jive5ab* ≥ 2.8)
< root >	char	bank | nonbank ~~non bank~~	NULL	~~For <type> file this is the root file system path on where to store the incoming scans. (see Note 3)~~ Sent to Mark5C: for <type> Mark5C sets up the StreamStor disk module to store data in either bank or non-bank mode. Sent to Mark5A/Mark5B: sets up StreamStor to store data in either bank or non-bank mode (*jive5ab* ≥ 2.8)

Monitor-only parameters:

Parameter	Type	Values	Comments
< type >	char	mark5C ~~| file~~	Mark5C– Normal operating mode (default), empty on Mark5A, Mark5B (*jive5ab* ≥ 2.8) ~~file – write data to the OS system disk, not streamstor (see Note 3)~~
< root >	char	~~system path~~ bank | non bank	~~For <type> file this is the root file system path on where to store the incoming scans. (see Note 3)~~ Reply from Mark5C: for <type> Mark5C indicates if the system is running in bank or non-bank mode Reply from Mark5A/Mark5B: indicates if the system is running in bank or non-bank mode (*jive5ab* ≥ 2.8)

Notes:

1. ~~A personality is here defined as a set of functions bound to the commands and queries described in this document. A possible implementation of this is for witch personality to have a setup() function and a shutdown() function that are called to initialize and clean up from a personality change, and a set of functions that are mapped to the commands and queries. Nothing is to preclude the various personalities from sharing a subset of functionality. The implementation of *drs* should make it easy to add new personalities to the program.~~
2. This command cannot be issued while a delayed completion operation is in effect or while data is being recorded on any type of medium.
3. ~~The file personality causes incoming data received through the system NIC to be written to a file system (e.g., RAID Array). The optional parameter should be a directory specifying the root of the file system to write. Files written to this directory will have systematically determined file names bearing close resemblance to scan names on Mark5 Modules.. Additionally, a file containing the equivalent of a Mark5 scan list will be created in the specified directory~~.
4. Implementation of this command is optional; its absence will not imply non-conformance with the Mark5C software specification.
5. *jive5ab* >= 2.6.0 implements the 'file' personality as generic packet recorder using the FlexBuff/Mark6 recording model. For this to be enabled on Mark5C it is sufficient to issue 'set_disks=' and 'record=on : ...;' (besides configuring the 'mode=') in a non-default runtime (see Section **6**, and **6.2** specifically).

personality

play – Play data from from current play pointer position

Command syntax: play = <play arm/on/off> : [<start play pointer>] : [<ROT start>];
Command response: !play = <return code>;

Query syntax: play? ;
Query response: !play ? <return code> : <status> ;

Purpose: Initiate playback from disk data at current or specified play-pointer position.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<play arm/on/off>	char	arm | on | off	off	'arm' - causes Mark5A to pre-fill buffer and prepare to play at position specified by field 2 – see Note 2. 'on' – causes playback to start at position specified by field 2. If field 2 is null, starts playback from current play pointer; Field 2 should be null for 'play=on' command following a successful 'play=arm'. 'off' = stops playback (if active) and unconditionally updates playback pointer to current play position or, if field 2 is non-null, to the position specified. See also Note 1. Cannot be issued while 'record' is 'on' (error). In all play modes, all 64 output tracks are active; if fewer than 64 tracks were recorded, the recorded track set is duplicated to unused output tracks; see Note 2.
<start play pointer>	int	>=0	current play pntr	Absolute byte number in recorded data stream; if null field, maintains current value; if <start play pointer> and <ROT start> are both null fields, play starts at current play pointer value.
<ROT start>	int	#sysclks		For use with Mark 4 correlator only: cause play to start after specified number of sysclk periods (counting from beginning of year); sysclk frequency is normally 32MHz, so these can be very large numbers.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	arming | armed | on | off | halted | waiting	'arming' – arming is in progress 'armed' – system is armed and ready to start play 'on' – playback active 'off' – playback inactive 'halted' – playback stopped due to reaching end-of-media or (end-of-scan when playback initiated by 'scan_play') 'waiting' – delayed start of playback (special mode for correlator only)

Notes:

1. After play is turned 'on', the user should periodically query 'status' for details; if playback stops on its own accord (due to end-of-media, etc.), this will be reflected in the response to the 'status' query as 'halted', and a 'play' query will show the status as well; a subsequent command to turn play 'off' or 'on' will reset the relevant bits (9-8) in the 'status' response.
2. The 'play=arm' command causes the Mark 5A to prefill its buffers according to the prescribed position so that playing will start almost instantaneously after a subsequent 'play=on' command is issued; this is intended primarily for use at a correlator. The amount of time need to prefill the buffer can range from a few tens of msec to a few seconds. If all disks are good and all data have been recorded properly, the time will be relatively short; however, if difficulties with disks or recorded data are encountered during the prefill period, up

to several seconds may be required. A 'play?' query should be issued to verify the system is armed for playback before issuing a 'play=on' command. A 'play=on' without a preceding 'play=arm' will begin play, but after an indeterminate delay.

3. During playback initiated by a 'scan_play' command, a 'play?' query will indicated the playback status.
4. When playing back in a mode with fewer than 64 tracks, groups of tracks are duplicated so that all 64 track outputs are always active, as follows:

mode	Primary playback tracks	Duplicated playback tracks
'mark4:8' or 'vlba:8'	2-16 even (headstack 1)	Duplicated to 3-17 odd, 18-32 even, 19-33 even on hdstk1; hdstk2 is duplicate of hdstk1
'mark4:16' or 'vlba:16' - 16	2-33 even (headstack 1)	Duplicated to 2-33 even on hdstk1; hdstk2 is duplicate of hdstk1
'mark4:32' or 'vlba:32' - 32 tks	2-33 all (headstack 1)	Headstack 1 output is duplicated to Headstack 2
'mark4:64' or 'vlba:64' - 64 tks	2-33 (headstacks 1 and 2)	None
'st' (any submode) – 32 tks	2-33 all (headstack 1)	Headstack 1 output is duplicated to Headstack 2
tvg	equivalent to tracks 2-33	Headstack 1 output is duplicated to Headstack 2

Playback of 'Mark5A+n:k' data is to same set of output tracks as for 'vlba:k' data.

5. Note that record/play pointers may have values as large as ~$2x10^{13}$ (~44 bits), so pointer arithmetic must be handled appropriately.
6. Playback clock rate is set by the 'play_rate' command
7. When playing, the playback pointer will update to show the approximate position. If the playback pointer is noted not to be incrementing, an error flag is set in the 'status?' query which can be used as a first order check of proper playback.

play_rate – Set playback data rate; set tvg rate

Command syntax: play_rate = <play rate reference> : <rate> ;
Command response: !play_rate = <return code> ;

Query syntax: play_rate? ;
Query response: !play_rate ? <return code> : <track data rate> : <track clock rate> : <clockgen freq> ;

Purpose: Set the playback rate (specified as <track data rate>, <track clock rate> or <clock generator frequency>. Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<play rate reference>	char	data | clock | clockgen | ext	clock	'data' – set output track data rate (not including parity) to specified value. 'clock' – set output track clock rate (including parity) to specified value. 'clockgen' – set clock generator chip to specified frequency; max is 40 MHz.. 'ext' – external clock select See also Notes below.
<rate>	real	MHz	8	>0 – set rate to specified value; freq resolution of clock generator chip is ~20 mHz. If in 'tvg' mode, sets on-board clock generator rate regardless of value of <play rate reference>; see Notes.

Monitor-only parameters:

Parameter	Type	Values	Comments
<track data rate>	real	Mbps	Track data rate (without parity); =0 if external clock selected
<track clock rate>	real	MHz	Track clock rate; see Note 1 for relationship to <track data rate>
<clockgen freq>	real	MHz	Internal clock generator frequency; see Note 1 for relationship to <track data rate>

Notes:

1. For a given operating mode, the relationships between <track data rate>, <track clock rate> and <clockgen freq> are as follows:

mode:submode	<track data rate> (Mbps)	Typical 'standard' values of <track data rate>	Corresponding <track clock rate> (MHz)	Corresponding <clockgen> frq (MHz)	Total playback data rate (Mbps)
st:mark4	*f*	2 | 4 | 8 | 16	9/8**f*	9/8**f*	32*9/8**f*
st:vlba	*f*	2 | 4 | 8	9/8**f*	9/8**f*	32*9/8**f*
mark4:8	*f*	2 | 4 | 8 | 16	9/8**f*	9/8**f*	8**f*
mark4:16	*f*	2 | 4 | 8 | 16	9/8**f*	9/8**f*	16**f*
mark4:32	*f*	2 | 4 | 8 | 16	9/8**f*	9/8**f*	32**f*
mark4:64	*f*	2 | 4 | 8 | 16	9/8**f*	2*9/8**f*	64**f*
vlba:8 *or* mark5a+n:8	*f*	2 | 4 | 8	1.008*9/8**f*	1.008*9/8**f*	1.008*8**f*
vlba:16 *or* mark5a+n:16	*f*	2 | 4 | 8	1.008*9/8**f*	1.008*9/8**f*	1.008*16**f*
vlba:32 *or* mark5a+n:32	*f*	2 | 4 | 8	1.008*9/8**f*	1.008*9/8**f*	1.008*32**f*
vlba:64 *or* mark5a+n:64	*f*	2 | 4 | 8	1.008*9/8**f*	2*1.008*9/8**f*	1.008*64**f*
tvg (see Note 4)	*f*	any up to 40	*f*	*f*	32**f*

play_rate

2. Upon a 'mode' change, the *jive5ab* software automatically makes any necessary adjustments to the clock generator to meet the current <track data rate> value (e.g. as returned by a 'play_rate?' query).
3. The value of the 'play_rate' parameters has no effect when recording data from an external source; the recording rate is strictly determined by the operating mode and input clock frequency. However, when using the 'rtime?' query to determine the remaining available recording time, the 'play_rate' parameters must correspond to the input data rate.
4. When recording or playing tvg data, <rate> record/playback aggregate rate is always 32*<rate>; set <rate> equal to 32 for tvg record/playback at 1024Mbps.
5. The maximum clock generator rate is 40 MHz, which results in corresponding maximum <track data rates> and <track clock rates> as follows:

data mode:submode	**<track data rate> (Mbps)**	**<track clock rate> (MHz)**
st:mark4	35.56	40
st:vlba	35.27	40
mark4:8	35.56	40
mark4:16	35.56	40
mark4:32	35.56	40
mark4:64	17.78	20
vlba:8 *or* mark5a+n:8	35.27	40
vlba:16 *or* mark5a+n:16	35.27	40
vlba:32 *or* mark5a+n:32	35.27	40
vlba:64 *or* mark5a+n:64	17.64	20
tvg	40	40

6. *jive5ab* supports the play_rate on Mark5B/DOM and generic systems to allow it to specify Mark4/VLBA data format. Use in conjunction with "mode=mark4:..." or "mode=vlba:..." to set the complete data format.

play_rate

pointers – Get current value of record, start-scan and stop-scan pointers (query only)

Query syntax: pointers? ;
Query response: !pointers? <record pointer> : <start-scan pointer> : <stop-scan pointer>;

Purpose: Get current value of record, start-scan and stop-scan pointers

Monitor-only parameters:

Parameter	Type	Values	Comments
<record pointer>	int	bytes	If stopped, returns position at which 'record=on' command will begin recording (always appends to existing); if recording, returns current record position.
<start-scan pointer>	int	bytes	Current value of <start-scan pointer>
<stop-scan pointer>	int	bytes	Current value of <stop-scan pointer>; '-' if undefined.

Notes:

1. Note that the returned byte numbers may have values as large as ~$2x10^{13}$ (~44 bits), so pointer arithmetic must be handled appropriately.
2. When recording, the <record pointer> will be updated to show the approximate current recording position. If the record pointer is noted not to be incrementing during recording, an error flag is set in the 'status?' query which can be used as a first order check of proper operation.

pointers

position – Get current value of record and play pointers (query only)

Query syntax: position? ;
Query response: !position? <return code> : <record pointer> : <play pointer>;

Purpose: Get current value of record and play pointers

Monitor-only parameters:

Parameter	Type	Values	Comments
<record pointer>	int	bytes	If stopped, returns position at which 'record=on' command will begin recording (always appends to existing); if recording, returns current record position.
<play pointer>	int	bytes	Current value of <play pointer>

Notes:

1. Note that the returned byte numbers may have values as large as ~$2x10^{13}$ (~44 bits), so pointer arithmetic must be handled appropriately.
2. When recording, the <record pointer> will be updated to show the approximate current recording position. If the record pointer is noted not to be incrementing during recording, an error flag is set in the 'status?' query which can be used as a first order check of proper operation.

position

protect – Set write protection for active module

Command syntax: protect = <on | off> ;
Command response: !protect = <return code> ;

Query syntax: protect? ;
Query response: !protect? <return code> : <protect on/off>;

Purpose: Set write protection on/off for active disk module

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<off>	char	on | off	off	

Notes:
1. A 'protect=on' command prevents any additional writing to module.
2. A 'protect=off" command allows writing to a module.
3. A 'protect=off' command is required to *immediately* precede a 'reset=erase', 'reset=erase_last_scan' or 'VSN=…' command, even if protection is already off. This protects the module from any accidental erasure or rewriting of the VSN.

protect

record – Turn recording on|off; assign scan label (5A, 5B/DIM, 5C)

Command syntax: record = <record on/off> : <scan label/name> : [<experiment name>] : [<station code>] ;
Command response: !record = <return code> ;

Query syntax: record? ;
Query response: !record ? <return code> : <status>: <scan#> : <scan label> ;

Purpose: Turn recording on|off; assign scan name, experiment name and station code

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<record on/off>	char	on | off		'on' automatically appends to the end of the existing recording. 'off' stops recording and leaves system in 'idle' mode.
<scan name>	ASCII	32 chars max		Relevant only if record is 'on'. If in <scan label> format, field is parsed for <exp name>, <station code> and <scan name>. Otherwise, interpreted as <scan name>, in which case <experiment name> and <station code> should be specified separately. If <scan name> is duplicate of already-recorded scan, a suffix will be added to the <scan name> part of the <scan label> -- see Note 6.
<experiment name>	ASCII	8 chars max		Experiment name; ignored if <record on/off> is 'off'
<station code>	ASCII	8 chars max		Station code; ignored if <record on/off> is 'off'

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	on | off | halted | throttled | overflow | waiting	'halted' indicates end-of-media was encountered while recording. 'throttled', 'overflow' and 'waiting' are all error conditions.
<scan#>	int		Sequential scan number; starts at 1 for first recorded scan.
<scan label>	ASCII		Scan label – see Notes 5 & 6. See Section 6 for definition of scan label.

Notes:

1. After record is turned 'on', the user should periodically query 'status' for details; if recording stops on its own accord (due to end-of-media, etc.), this will be reflected in the response to the 'status' query as 'recording stopped', and a 'record' query will show the status as 'halted'; a subsequent command to turn record 'off' or 'on' will reset the relevant bits (5-4) in the 'status' response.
2. When recording, the record pointer will update to show the approximate position. If the record pointer is noted not to be incrementing, an error flag is set in the 'status?' query which can be used as a first order check of proper recording.
3. When <status> is 'off', a 'record?' query returns the <scan label> of the last recorded scan, if any.
4. Typical causes for status errors:
    a. "throttled" – data rate from Mark 5B I/O card is too fast for disks to keep up (flag received by I/O board from StreamStor card)
    b. "overflow" – FIFO overflow on Mark 5B I/O card
    c. "waiting" – CLOCK has stopped or is faulty

record (5A, 5B/DIM, 5C)

5. The <scan label> field is created in the standardized format specified in Section 6, namely '<exp name>_<station code>_<scan name>'. If <experiment name> and/or <station code> are null, they will be replaced with 'EXP' and 'STN', respectively.
6. An attempt to record a scan with a duplicate scan name on the same disk module will cause a trailing alphabetical character ('a-z', then 'A-Z') to be automatically appended to the scan name (example: '312-1245a'). If more than 52 scans with same user-specified name, the suffix sequence will repeat.
7. Starting from `jive5ab` >= 2.8.2 the 'record=' command will check wether the requested data rate can be handled by the underlying hardware. This defect was found due to a station attempting to record 2Gbps on a plain Mark5B/DIM (a 5B+/DIM is needed for this), resulting in 100% corrupted data.

record (5A, 5B/DIM, 5C)

record – Turn recording on|off; assign scan label; configure Mark6/FlexBuff record setup (G)

Command syntax: record = <on/off> : <scan label/name> : [<experiment name>] : [<station code>] ;
record = mk6 : <recording format> ;
record = nthread : <nReaders> : <nWriters> ;

Command response: !record = <return code> ;

Query syntax: record? [mk6 | nthread] ;

Query response: !record ? <return code> : <status> : <scan#> : <scan label> : <byte count> ;
!record ? <return code> : <recording format> ;
!record ? <return code> : <nReaders> : <nWriters> ;

Purpose: Turn recording on|off; assign scan name, experiment name and station code. Additions were introduced in *jive5ab* 2.6.2.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<record on/off>	char	on | off mk6 nthread		'on' automatically appends to the end of the existing recording. 'off' stops recording and leaves system in 'idle' mode. 'mk6' sets Mark6 or FlexBuff recording mode depending on next argument. See Note 4. 'nthread' sets number of simultaneous network reader and/or disk writer threads. See Note 5.
<scan name>	ASCII	32 chars max		Relevant only if record is 'on'. If in <scan label> format, field is parsed for <exp name>, <station code> and <scan name>. Otherwise, interpreted as <scan name>, in which case <experiment name> and <station code> should be specified separately. If <scan name> is duplicate of already-recorded scan, a suffix will be added to the <scan name> part of the <scan label> -- see Note 3.
<recording format>	int	0|1	0	'0', '1' – only allowed when <record on/off> = 'mk6', set Mark6 or FlexBuff record mode. See Note 4. The default for this value can be set from the command line. The compiled in default is '0'.
<nReaders>	int	1 - <n>	1	1 - <n> - when <record on/off> = 'nthread'. See Note 5.
<experiment name>	ASCII	8 chars max		Experiment name; ignored if <record on/off> is 'off' or 'mk6'
<nWriters>	int	1-<n>	1	1 - <n> when <record on/off> is 'nthread'. See Note 5.
<station code>	ASCII	8 chars max		Station code; ignored if <record on/off> is 'off', 'mk6' or 'nthread'

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	char	active | inactive | on | off	'active' if the system is currently recording, 'inactive' otherwise. **on/off since `jive5ab` 2.8.1**
<recording format>	int	0 | 1	'0', '1', indicating Mark6 ('1') or FlexBuff mode ('0'), if query argument was 'mk6'
<nReaders>	int	1-<n>	configured number of network reader threads if query argument was 'nthread'
<scan#>	int		Sequential scan number; starts at 1 for first recorded scan. Not returned if query argument was 'mk6'
<nWriters>	int		configured number of disk writer threads if query argument was 'nthread'
<scan label>	ASCII		Scan label – see Notes 5 & 6. See Section 6 for definition of scan label. Id.
<byte count>	int		Count of how many bytes have been recorded since `record=on`. Used by m5copy.

record (Generic)

Notes:

1. When <status> is 'off', a 'record?' query returns the <scan label> of the last recorded scan, if any.
2. The <scan label> field is created in the standardized format specified in Section 6, namely '<exp name>_<station code>_<scan name>'. If <experiment name> and/or <station code> are null, they will be replaced with 'EXP' and 'STN', respectively.
3. An attempt to record a scan with a duplicate scan name ~~on the same disk module~~ will cause a trailing alphabetical character ('a-z', then 'A-Z') to be automatically appended to the scan name (example: '312-1245a'). ~~If more than 52 scans with same user-specified name, the suffix sequence will repeat.~~ On FlexBuff/Mark6 this is impossible and an error + refusal to record will happen if an attempt is made to record more than 52 scans with the same label.
4. If the `mk6` value is set to '1', next time a recording is started, the data will be recorded in MIT Haystack d-plane file version 2.0 compatible format. This means one file per disk (on all disks returned by `set_disks?`) by the name of `<scan label>`. If the `mk6` value is '0', data will be recorded in FlexBuff mode: data will be striped across all disks and stored in files of a configurable size (see `net_protocol=`) in a directory per disk, called `<scan label>/`. The data are stored in files named `<scan_label>.<sequence nr>`.
5. Recording on Mark6/FlexBuff requires tuning the number of parallel disk writers to a combination of recorded data rate and number of CPU cores available to *jive5ab*. The default number if disk writers is '1'. On modern Mark6/FlexBuff platforms this should be sufficient for operation up to 8192 Mbps. For 16384 Mbps recording set nWriter to a value in the ball park of 3 (three) or four (4). Configuring less possibly yields in not-enough-performance whilst higher values will result in many threads competing for the same resource, also hurting performance. The number of network readers (nReaders) is settable and queryable but not yet actively *used* in *jive5ab* 2.8!

record (Generic)

recover – Recover record pointer which was reset abnormally during recording

Command syntax: recover = <recovery mode> ;
Command response: !recover = <return code> : <recovery mode>;

Query syntax: recover? ;
Query response: !recover ? <return code> ;

Purpose: Recover record pointer which was reset abnormally during recording.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<recovery mode>	int	0 | 1 | 2	1	0 – attempt to recover data from scan that was terminated abnormally during recording; see Note 1. 1 – attempt to recover from accidental use of 'sstest' or 'WRSpeed Test'; see Note 2. 2 – attempt to recover from StreamStor abnormality; see Note 3.

Notes:

1. A scan terminated abnormally during recording (for example, by a power failure or a keyswitch being accidentally turned to the 'off' position) will not be accessible unless special actions are taken to recover it by forcing the record pointer to the end of recorded data; the scan will be overwritten if a new 'record=on' command is issued before a recovery attempt is made. It is suggested that a 'record=off' command be tried before a 'recover=0' command; this will not cause any harm and might fix the problem by itself. It has also been reported that success with 'recover=0' is demonstrably higher if a 'scan_set' command to select a scan [seemingly any scan, but perhaps preferably to the last (incomplete) scan] is issued before the 'recover=0' is attempted.
2. The utility programs '*sstest*' and '*WRSpeedTest*' will overwrite any existing data near the beginning of a disk module, but will prevent access to user data recorded beyond that point. A 'recover=1' command will attempt to recover the data beyond the overwritten section; the overwritten data are irrecoverable.
3. (Most common) Try recover=2 to recover data that were erased, or if the record pointer has been set to a point near the beginning (often to zero).

recover

replaced_blks – Get number of replaced blocks on playback (query only)

Query syntax: replaced_blks? ;
Query response: !replaced_blks? <return code> : <disk 0> : <disk 1> : <disk 2> : <disk 3> : < disk 4> : <disk 5> : <disk 6> : <disk 7> : <total replaced blks> ;

Purpose: Get number of replaced blocks during playback on disk-by-disk basis.

Monitor-only parameters:

Parameter	Type	Values	Comments
<disk 0>	int		Number of replaced blocks on disk 0
<disk 1>	int		Number of replaced blocks on disk 1
<disk 2>	int		Number of replaced blocks on disk 2
<disk 3>	int		Number of replaced blocks on disk 3
<disk 4>	int		Number of replaced blocks on disk 4
<disk 5>	int		Number of replaced blocks on disk 5
<disk 6>	int		Number of replaced blocks on disk 6
<disk 7>	int		Number of replaced blocks on disk 7
<total replaced blks>	int		Total number of replaced blocks.

Notes:

1. If a disk is unable to provide a requested 65KB (actually 0xfff8 bytes) block of data within the allowed time limits, due to a slow or failed drive, the Mark 5A replaces the requested data block with a data block with even parity that can be detected by as invalid by a correlator. See 'Mark 5A User's Manual' for details.
2. Drive statistics and replaced-block counts are cleared and re-started whenever a new disk module is mounted or a 'start_stats' command is issued. Replaced-block counts restart on each 'play=on' or 'scan_play=on' command.
3. If the case of a totally failed drive, the replaced-block count for that drive will be 0 since the StreamStor ceases to ask for data from that drive, but the <total replaced blks> will be accurate. Statistics gathered from the 'get_stats?' query should be used to help diagnose the failed drive.
4. Replaced-block statistics are updated only after playback has ceased (i.e. replaced-block statistics are not updated during playback).

replaced_blks

reset – Reset Mark 5 unit (command only)

Command syntax: reset = <control> [: <layout>] ;
Command response: !reset = <return code> ;

Purpose: Reset system; mount/dismount disks

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<control>	char	erase | nberase erase_last_scan | abort | condition		'erase'sets record, start-scan and stop-scan pointers to zero (i.e. beginning of media); effectively erasing media; 'nberase' is a 'non-bank' erase that performs the same actions as the 'erase' command except the modules in Banks A and B are initialized as a single unit and recording takes place across the disks in both modules; normally used only for 2Gbps operation with Mark 5B+. 'erase_last_scan' erases the last recorded scan; sets record pointer to end-of-scan just prior to erased scan; sets start-scan and stop-scan pointer to beginning and end, respectively, of scan just prior to erased scan. 'abort' aborts active disk2net, disk2file or file2disk transfers (only) – See Note 2 'condition' starts a disk conditioning cycle (lenghty process!) much like SSErase w/o the need for stopping the server and starting a separate program. Best used togeter with SSErase.py, also written by JIVE. System is always left in 'idle' mode after any reset command. See Note 1.
<layout>	char	legacy | bigblock | <explicit layout>		If <control> = 'erase', and not blank, force UserDirectory layout to write. By default *jive5ab* will automatically choose the layout most appropriate for the system it is running on. See Note 5.

Notes:

1. The former 'reset=mount' and 'reset=dismount' commands are no longer supported; the keyswitches associated with the disk modules are used for all mount and dismount operations.
2. 'reset=abort' returns immediately, but there may be a delay of up to two seconds before the data transfer stops. During this delay, a 'status?' query will show what is happening. ~~The 'reset=abort' command simulates the end of data by setting 'nowbyte=endbyte', which then executes a normal termination.~~ *jive5ab* terminates the transfer immediately but some delay may happen because of O/S or hardware latency. E.g. when doing `disk2file` (especially) the Linux O/S buffers a lot of data. Upon closing the file these bytes must be flushed which will take a variable amount of time (see `disk2file` Notes on how to influence this!). Bytes still in *jive5ab*'s internal buffers are discarded.
3. A 'protect=off' command is required *immediately* prior to a 'reset=erase' or 'reset=erase_last_scan' command, even if protection is already off.
4. The 'reset=nberase' command requires that disk modules are mounted and ready in both banks. Bank A must contain eight disks; bank B may have fewer, though it will normally have the same number. After the 'reset=nberase' command is completed, each module will have recorded on it (until the module is again erased) the following information: 1) bank position of the module (A or B), and 2) VSN of the module in the opposite bank. Each subsequent occasion when the modules are mounted for record or readback operation, the location and identification of the modules is checked; only if the proper modules are mounted in the proper positions will *jive5ab* place the system into non-bank mode or allow any read or write operations.
5. The format of the User Directory ('Scan Directory') as written on the Mark5 disk packs has seen a number of versions through the years; some of them dependant on actual Conduant SDK version. 'legacy' refers to Mark5A, 1024 scans max, 'bigblock' allows 4 Gbps operation on Mark5C (requires firmware > 16.38). Please refer to Mark5 Memo #100[3] for intricate details plus a description of what are allowed values for <explicit layout>. See also the 'layout?' query.

[3] http://www.haystack.mit.edu/tech/vlbi/mark5/mark5_memos/100.pdf

reset

rtime – Get remaining record time on current disk set (query only) (5A)

Query syntax: rtime? ;
Query response: !rtime ? <return code> : <remaining time> : <remaining GB> : <remaining percent> : <data source> : <bit-stream mask> : <decimation ratio> : <total recording rate> ;

Purpose: Get remaining record time of current disk set; assumes recording will be in the mode currently set by the 'mode' command and data rate set by 'play_rate' command.

Monitor-only parameters:

Parameter	Type	Values	Comments
<remaining time>	real	seconds	Approximate remaining record time for current 'mode' and 'play_rate' parameters; Requires that 'play_rate' be set to current record rate – see Notes. See Note 2.
<remaining GB>	real	GB	GB remaining on current disk set (1 GB = 10^9 bytes). See Note 2.
<remaining percent>	real	0-100	Remaining percentage of disk space still available. See Note 2.
<mode>	char	mark4 | vlba | st | tvg	Mode assumed in calculation of <remaining time>. See 'mode' command.
<submode>	char	8 | 16 | 32 | 64 | mark4 | vlba	Submode assumed in calculation of <remaining time>
<track data rate>	real	MHz	Track data rate assumed in calculation of <remaining time>; see 'play_rate' command. See Note 2.
<total recording rate>	real	Mbps	Net recording rate assumed in calculation of <remaining time>, based on current clock frequency and 'mode' parameters. See Note 2.

Notes:

1. Each 'rtime?' query returns an updated estimate during recording; a somewhat more accurate estimate is obtained when recording is stopped and the effects of any slow or bad disks can be more accurately measured.
2. For readability *jive5ab* returns these values with their unit character(s) appended: "s", "GB", "%", "MHz" and "Mbps" respectively.

rtime (5A)

rtime – Get remaining record time on current disk set (query only) (5B/DIM, 5C)

Query syntax: rtime? ;
Query response: !rtime ? <return code> : <remaining time> : <remaining GB> : <remaining percent> : <data source> : <bit-stream mask> : <decimation ratio> : <total recording rate> ;

Purpose: Get remaining record time of current disk set; assumes recording will be in the mode currently set by the 'mode' command and data rate set by 'clock_set' command. On 5C "mode = unk ;" is supported

Monitor-only parameters:

Parameter	Type	Values	Comments
<remaining time>	real	seconds	Approximate remaining record time for current 'mode' and 'clock_set' parameters; Requires that 'clock_set' be set to current record rate – see Notes. See Note 3.
<remaining GB>	real	GB	GB remaining on current disk set (1 GB = 10^9 bytes). See Note 3.
<remaining percent>	real	0-100	Remaining percentage of disk space still available. See Note 3.
<data source>	char	ext | tvg	Assumed to be same as specified in last 'mode' command
<bit-stream mask>	hex		Assumed to be same as specified in last 'mode' command
<decimation ratio>	int		Assumed to be same as specified in last 'mode' command
<total recording rate>	real	Mbps	Net recording rate assumed in calculation of <remaining time>, based on current clock frequency and 'mode' parameters. See Note 3.

Notes:

1. Each 'rtime?' query returns an updated estimate during recording; a somewhat more accurate estimate is obtained when recording is stopped and the effects of any slow or bad disks can be more accurately measured.
2. The difference between the implementation on Mark5B and Mark5C is the fact that on Mark5C the mode can be set to 'unk' – for unknown data. This makes the recording data rate also unknown and as such the remaining time as well.
3. For readability *jive5ab* returns these values with their unit character(s) appended: "s", "GB", "%" and "Mbps" respectively.

rtime (5B/DIM, 5C)

rtime – Get remaining record time on current disk set (query only) (G)

Query syntax: rtime? ;
Query response: !rtime ? <return code> : <remaining time> : <remaining GB> : <remaining percent> : <data format> : <#tracks> : <decimation ratio> : <total recording rate> ;

Purpose: Get remaining record time of accumulated free disk space over all disks currently selected; assumes recording will be in the mode currently set by the 'mode' command. This command appeared in *jive5ab* 2.6.2.

Monitor-only parameters:

Parameter	Type	Values	Comments
<remaining time>	real	seconds	Approximate remaining record time for current 'mode'. See Note 3.
<remaining GB>	real	GB	GB remaining on current disk set (1 GB = 10^9 bytes). See Note 3.
<remaining percent>	real	0-100	Remaining percentage of disk space still available. See Note 3.
<data format>	char	mark4 | vlba | Mark5B | vdif | legacy vdif | tvg | SS | <unknown>	Assumed to be same as specified in last 'mode' command
<#tracks>	int		Number of bit streams (tracks) derived from last 'mode' command
<decimation ratio>	int	0	Currently fixed to "0"
<total recording rate>	real	Mbps	Net recording rate assumed in calculation of <remaining time>, based on current 'mode'. See Note 3.

Notes:

1. Each 'rtime?' query returns an updated estimate during recording; a somewhat more accurate estimate is obtained when recording is stopped and the effects of any slow or bad disks can be more accurately measured.
2. The 'rtime?' implementation in *jive5ab* >= 2.6.2 running on Mark6/FlexBuff returns the free space accumulated over all disks currently selected, as returned by "set_disks?"
3. For readability *jive5ab* returns these values with their unit character(s) appended: "s", "GB", "%" and "Mbps" respectively.

rtime (G)

runtime – Control multiple simultaneous transfer environments

Command syntax: runtime = <name> [: <action>] ;
Command response: !runtime = <return code> ;

Query syntax: runtime? ;
Query response: !runtime ? <return code> : <current> : <#runtimes> [: <name*2*> : ... : <name*N*>] ;

Purpose: Manage transfer environments to allow for multiple, simultaneous, transfers

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<name>	char			Switch to runtime <name>. Create it if runtime <name> does not exist yet. See Notes, specifically Note 2.
<action>	char	new | exists | delete | transient		If given, modifies the default behaviour according to this: 'new' – only succeeds if <name> does not exist yet; an exclusive flag 'transient' – create if not exists and mark <name> for automatic deletion. See Note 6. (*jive5ab* >= 2.7.3) 'exists' – do not create if <name> does not exist; only switch if <name> already exists 'delete' – delete <name> and all its resources. See Note 4.

Monitor-only parameters:

Parameter	Type	Comments
<current>	char	Name of runtime this control connection is associated with
<#runtimes>	int	Total number of runtimes currently defined in the system
<name*N*>	char	Names of the other runtimes; <current> is missing from this list.

Notes:

1. New command connections to *jive5ab*'s control port are automatically associated with the default runtime, runtime "0". See Section **6** of this document. Each runtime is uniquely defined by its name – a meaningless (to *jive5ab*) sequence of characters.
2. Versions < 2.6.3 take the statement from Note 1 very literally; the name can be an empty string too. Thus a literal command 'runtime = ;' would create-and-switch-to a runtime by the name "" (the empty string). It will **NOT** put the connection back in the default runtime. The empty string does not read nicely and the behaviour is counterintuitive. Starting from 2.6.3 *jive5ab* disallows the empty string as <name>.
3. A control's runtime association remains fixed for the duration of the connection, unless it is altered using 'runtime = ... ;'. After a succesfull 'runtime=...;' the association remains fixed again, until the connection is closed or the association is altered using another 'runtime=...;' command.
4. Runtimes are dynamically created on an as-needed basis. Switching to a non-existent runtime is a sufficient 'needed' basis, unless the behaviour of 'runtime=...;' is altered through specification of an <action>.
5. Runtimes should be deleted when done using the specific runtime. If the current runtime is deleted, the system will automatically put the connection back into the default runtime. (See Note 6 for possible automatic runtime deletion in *jive5ab* >= 2.7.3)
6. Starting from version 2.7.3 runtimes can be created with the 'transient' flag. The control connection creating the runtime will be marked as owner of the runtime. If this connection is closed, the runtime and its resources will be automatically deleted from the system.

runtime

scan_check – Check recorded data between start-scan and stop-scan pointers (query only)

Query syntax: scan_check? [<strict> : <#bytes to read>] ;
Query response: !scan_check ? <return code> : <scan#> : <scan label> : <data type> : <ntrack> : <start time> : <scan length> : <total recording rate> : <#missing bytes> [: <data array size>];

Purpose: Check recorded data between the start-scan and stop-scan pointers (e.g. returned by 'pointers?' query). The query arguments <strict> and <#bytes to read> were introduced in *jive5ab* 2.5.1. Mark6/FlexBuff support was added in *jive5ab* 2.6.2.

Monitor arguments:

Parameter	Type	Allowed values	Default	Comments
<strict>	int	0 | 1	1	0 – Enable less strict checking: • no CRC checks on frame headers • no check for invalid last digit in MarkIV time stamp or time stamp consistency with data rate • no consistency check between frame number+data rate and VLBA time stamp in Mark5B header 1 – Everything has to be good for data format to be detected
<#bytes to read>	int		1000000	Amount of data to find frames in. The default ~1MB may be too low for high data rate detection (>>2Gbps)

Monitor-only parameters:

Parameter	Type	Values	Comments
<scan#>	int		Start at 1 for first recorded scan, but see Notes 7, 8.
<scan label>	literal ASCII		
<data type>	char	- | tvg | SS | ? vdif (legacy) | mark5b | [st :] mark4 | vlba	dash – Mark 5B format, but undetermined data type (probably real data) tvg – undecimated 32-bit-wide tvg data (see Note 4) SS – raw StreamStor test pattern data ? – ~~recording not in Mark 5B format (might be Mark 5A, for example); all subsequent return fields are null~~. See Note 6. 'st :' – this extra field is inserted if straight-through MarkIV or VLBA is detected
~~<date code>~~ <ntrack>	int		~~3-digit date code written in first disk frame header~~ Number of tracks detected ('?' if VDIF)
<start time>	time		Time tag at first frame header in scan. See Note 5.
<scan length>	time		
<total recording rate>	real	Mbps	
<#missing bytes>	int	See Note 5	Should always be =0 for normally recorded data. >0 indicates #bytes that have been dropped somewhere within scan <0 indicates #bytes that have been added somewhere within scan
[<data array size>]	int		Parameter is only returned if the detected format is simple VDIF and reports the VDIF data array length.

Notes:

1. The 'scan_check' query will be honored only if record and play are both off and does not affect the start-scan or stop-scan pointers.
2. The 'scan_check' query essentially executes a 'data_check' starting at the start-scan pointer, followed by a 'data_check' just prior to the stop-scan pointer. This allows information about the selected scan to be conveniently determined.

scan_check

3. Only tvg data that were recorded with a bit-stream mask of 0xffffffff and no decimation will be recognized.
4. Regarding the <start time> value returned by the 'data_check?' and, 'scan_check?' queries: The year and DOY reported in <start time> represent the most recent date consistent with the 3-digit <date code> in the frame header time tag (modulo 1000 value of Modified Julian Day as defined in VLBA tape-format header); this algorithm reports the proper year and DOY provided the data were taken no more than 1000 days ago.
5. The <#missing bytes> parameter is calculated as the difference the expected number of bytes between two samples of recorded data based on embedded time tags and the actual observed number of bytes between the same time tags. The reported number is the *total* number of bytes missing (or added) between the two sample points.
6. *jive5ab* extends `scan_check?` to recognize all data formats, not just the recorder's native data type (see Section 7). `scan_check?`'s output may be subtly different depending on wether it is issued on a Mark5B/DIM, Mark6/FlexBuff or any of the other Mark5's.
7. *jive5ab* with Mark6/FlexBuff support does not keep global scan numbers on these systems as was done on the Mark5 disk packs. On Mark6/FlexBuff it remembers and numbers the scans it has recorded since the program's invocation from *0* .. *n-1*. As such, after a restart, scan numbers start back at *0*.
8. There is no autodetection of Mark6/FlexBuff recorded format. If a previously recorded Mark6 or FlexBuff recording is to be `scan_check`'ed it *may* be necessary to set the recording format appropriately (see `record = mk6 : 0|1`).

scan_check

scan_set – Set start-scan and stop-scan pointers for data readback

Command syntax: scan_set = <search string> : [<start scan>] : [<stop scan>] ;
Command response: !scan_set = <return code> ;

Query syntax: scan_set? ;
Query response: !scan_set? <return code> : <scan number> : <scan label> : <start byte#> : <stop byte#> ;

Purpose: Set start-scan and stop-scan pointers for data_check, scan_check, disk2file and disk2net. Mark6/FlexBuff support was added in *jive5ab* 2.6.2. See Note 7 for *jive5ab*'s difference in query response!

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<search string>	int or ASCII	scan number | scan label | 'inc' | 'dec' | 'next'	last recorded scan	First attempts to interpret as scan number (first scan is number 1); if not numeric or no match, attempts to match all or part of existing scan label, case insensitive (see Note 1). 'inc' increments to next scan; cycles back to first scan at end; 'dec' decrements to previous scan. 'next' finds next scan with previous value of <search string>. If null field, defaults to last fully recorded scan.
<start scan>	char | time | int	s | c | e | s+ | <time> | +<time> | -<time> | +<bytes> | -<bytes>	s	s|c|e|s+: Set start scan position to 'start', 'center', 'end' (actually ~1MB before end) of scan, or specified <time> within scan; this is convenient if you want to do a subsequent 'data_check' at a prescribed position. 's+' sets the start-scan pointer to 65536 bytes past the start of the scan. <time>: time within scan: see Notes 2 & 3 +<time>: offset time from beginning of scan (i.e. '+30s' will start 30 seconds from beginning of scan) -<time>: offset time from end of scan (i.e. '-30s' will start 30 seconds before end of scan) +<bytes>: offset number of bytes from beginning of scan. -<bytes>: offset number of bytes from end of scan
<stop scan>	time | int	<time> | +<time> | -<time> | +<bytes> | -<bytes>	end-of scan	<time>: Time at which to end readback; see Notes 2 & 3. If preceded by '+', indicates duration of data (in record-clock time) from <start scan> time. +<time>: offset time from <start scan> position. -<time>: offset time from end-of-scan +<bytes>: offset bytes from <start scan> position -<bytes>: offset bytes from end of scan

Monitor-only parameters: NOTE *jive5ab* returns this Mark5A-style reply on **all** systems! DIMino/drs replies are subtly different.

Parameter	Type	Values	Comments
<scan number>	int		Scan number of scan matching <search string>. '?' on FlexBuff/Mark6 because those have no scan numbers.
<scan label>	time		Scan label of scan matching <search string>
<start byte#>	int		Absolute start byte number of selected data range
<stop byte#>	int		Absolute end byte number of selected data range

scan_set

Notes:

1. If <search string> is all numeric, scan_set will first try to interpret it as a scan number. If it is not all numeric or the scan number does not exist, scan_set will find the first scan label that matches all or part of the corresponding non-null subfields in <search string>; null subfields in <search string> match all scans. All searches start from the first scan except if 'scan_set=next'; if 'scan_set' is already pointing at last scan, then 'scan_set=next' will start search at first scan. Searches are case insensitive.
   Examples:

<search string>	**Matches**
105	Scan #105, if it exists; otherwise, first scan label containing '105' anywhere (e.g. 'grf103_ef_123-1056')
grf103_	First scan label with 1st subfield containing 'grf103'
_EF	First scan label with 2nd subfield containing 'EF' (searches are case insensitive)
__1056	First scan label with 3rd subfield containing '1056'
_ef_1056	First scan label with 2nd subfield containing 'ef' and 3rd subfield containing '1056'

2. When 'record=off' is issued or end-of-media (following a 'record=on') is encountered, the start-scan and stop-scan pointers are set to span the entire just-recorded scan.
3. If the <start scan> or <stop scan> parameter is a <time> value, this time must be specified with sufficient significance to resolve any ambiguity within the scan. For example, '30s' would set the start-scan pointer to start at the first '30s' mark in the scan (regardless of the value of the minute). If a calculated byte position is outside the bounds of a scan, an error code '0', but the default will be retained and an error code will be posted, which can be recovered by an 'error?' or 'status?' query.
4. A 'scan_set=' command is not allowed during active data transfers.
5. The specified values of <start scan> and <stop scan> must be within the target scan.
6. The 'pointers' query can be issued at any time to retrieve the current value of the start-scan and stop-scan pointers.
7. Due to an oversight on *jive5ab*'s authors' behalf, the '`scan_set?`' query response is strangely coherent across all systems. *jive5ab* returns the exact same response format always: it returns the response as documented in the Mark5A Command Set 2.73. The subtle differences between the documented Mark5A, Mark5B/DIM and Mark5C responses to this query had elided the authors and were only brought to their attention in August 2015. To wit: Mark5B/DIM should not return the scan number at all and return the selected start/end as *time stamps* rather than byte numbers. The Mark5C version should be a strange amalgamation between the 5A and 5B/DIM responses: it should not return the scan number (cf. 5B/DIM) but should return the start/end as absolute byte numbers (cf. 5A).

scan_set

set_disks – Select mount points to record on (FlexBuff/Mark6)

Command syntax: set_disks = <pattern1> [: <pattern2> : ... : <patternN>] ;
Command response: !set_disks = <return code> : <#mount points> ;

Query syntax: set_disks? ;
Query response: !set_disks ? <return code> : <#disks> : <disk0> : <disk1> : ... : <diskN> ;

Purpose: Select mount points to next record on with FlexBuff or Mark6. This command appeared in *jive5ab* 2.6.0

Settable parameters:

Parameter	Type	Allowed values	Comments
<patternN>	char	flexbuff |	'flexbuff' – select all active FlexBuff mount points "/mnt/disk[0-9]+"
		mk6 |	'mk6' – select all active Mark6 mount points "/mnt/disks/[1-4]/[0-7]"
		null |	'null' – select no mountpoints succesfully (no matches is an error), useful for testing w/o physical disk writing (jive5ab ≥ 3.0.0)
		[1234]+ |	'1' .. '4' – (or a concatenation thereof) select all disks in Mark6 slot '1', '2' or the combination thereof. ('1234' is equivalent to 'mk6')
		<MSN>	<MSN> - select all disks of mounted Mark6 disk pack with extended Module Serial Number <MSN> (case insensitive)
		shell pattern |	'shell pattern' – any path name, may contain shell wild cards like '*' and '?'
		regex |	'regex' – any non-builtin alias (like 'flexbuff' or '1') that starts with "^" and ends with "$", as in regular expression full string match
		<GRP>	<GRP> – a group (a.k.a. alias) as defined by the 'group_def=' command
			See Notes.

Monitor-only parameters:

Parameter	Type	Comments
<diskN>	char	Full path of selected mount point N. See Note 3.

Notes:

1. When this command executes, *jive5ab* queries the Operating System for currently mounted volumes. Only mounted volumes that are not the root file system are eligible for selection to be recorded on. The command will return the amount of actually selected mount points. A command that resulted in 0 (zero) selected eligible mount points returns an error code '4'. Since *jive5ab* 3.0.0 it is possible to select no mountpoints succesfully, using the special pattern 'null'. This feature can be used to test raw network capture performance by inhibiting writing to disk.
2. The patterns supported by the command are quite flexible. There are a number of built-in aliases, or shortcuts if you will, that ease selection. E.g. the built-in pattern 'mk6' is an alias for the regex-based selection "^/mnt/disks/[1-4]/[0-7]$". The regex form ensures that only disks exactly mounted by Mark6 software are selected. Using a shell pattern like "/mnt/disks/*/*" looks similar but *might* select more/unintended disks than wanted.
3. The query returns the current list of mount points that will be recorded on when a 'record = on : <scan label> ;' is issued.
4. The compiled-in default behaviour of *jive5ab* is to always find-and-select the 'flexbuff' mount points. This behaviour can be altered from the command line, see Section 2, "Running the *jive5ab* program".

set_dis

sp*2* – Configure, start, stop corner turning: split [in/fill/net/file/disk] to [net/file]

Command syntax: sp*2* = station : <VDIF station>
sp*2* = vdifsize : <output VDIF frame size>
sp*2* = bitspersample : <bits per sample>
sp*2* = bitsperchannel : <bits per channel>
spill2* = realtime : [0|1]

sp*2* = net_protocol : <protocol> : <sockbufsz>
sp*2* = mtu : <mtu>
sp*2* = ipd : <ipd>

spif2* = connect : <file> : <corner turning chain> : <outputX> = <dstY> [: ...] ;
sp*2* = connect : <corner turning chain> : <outputX> = <dstY> [: ...] ;

spill2* = on [: <nword> : <start> : <inc>] ;
spin2* = on [: <nbyte>]
sp[id/if]2* = on [: [+]<start byte> : [+]<end byte>]

spin2* = off
sp*2* = disconnect ;

Command response: !sp*2* = <return code>;

Query syntax: sp*2*? [<parameter>];
Query response: !sp*2*? <return code> : <status> | <parameter value> ; *(<status> if no <parameter> given, else <parameter>'s value)*

Purpose: The corner turning transfers allow dechannelization of all supported input data formats into single- or multichannel multi-thread legacy VDIF. See Section **8** (and specifically **8.2**) of this manual. The data format must be properly configured using "mode=" (magic-mode) or "mode=" in combination with "play_rate=" or "clock_set=", depending on MarkIV or Mark5B format.

The most important parameters to set before running are the bits-per-sample and bits-per-channel values; those cannot be inferred from the data format. Usually they are identical, except for VLBA/MarkIV formats with non 1:1 fan-in or fan-out. The VDIF station name is informative only.

When doing splet2net (read from network – corner turn – write to network) there are two sets of network parameters to be considered. The network configuration set by the normal 'net_protocol = ..' and 'mtu = ..' commands is used for the input section. Using 'splet2net = net_protocol : ...', 'splet2net = mtu : ...' (and optionally 'splet2net = ipd : ...') the output network configuration can be set.

The `<corner turning chain>` and `<outputX> = <dstY>` syntaxes are separately described in sections **8.2.1** and **8.2.2** respectively.

sp*2*

Settable parameters, in general: `sp*2* = <control> : <control value>`

Parameter	Type/ value description	Allowed values/types	Comments
<control>	char	station vdifsize bitspersample bitsperchannel net_protocol mtu ipd connect on off disconnect	Control the sub-commands of the sp*2* functions. 'connect', 'on/off' and 'disconnect' are used to set up and tear down the transfer, not unintentionally quite like 'in2net = ' and friends. See the connection set up/teardown notes for 'in2net=' Note that 'off' only applies to 'spin2*' transfers and pauses the transfer, not stop it. 'spin2*=on' resumes the transfer (again, not quite unlike 'in2net=') The other controls can be used to configure the corner turning itself and/or output details
<control value>	VDIF station output VDIF frame size bits per sample/channel protocol, sockbufsz mtu ipd file nword, start, inc realtime nbyte [+]start byte, [+]end byte	char int (>0 | -1) int see 'net_protocol=' see 'mtu=' see 'ipd=' ASCII int, int/hex, int int int, int	Two-letter station code. Override automatic-VDIF frame size selection. '-1' to select automatic selection. See Note 1. Sets the indicated bit widths for decoding input data format Refer to the documentation of the standard 'net_protocol = ' command Id. for 'mtu =' Id for 'ipd =' File name of file to open for reading data from Fill-pattern specifics, refer to 'fill2net = ' command documentation If '0' the system will run as fast as the hardware will, '1' means try to follow data rate from 'mode' Indicates how many bytes to read from the I/O board. Default 2^{64}-1 Absolute start or end byte number; offset from scan_set= or amount if starts with '+' prefix

Monitor-only arguments:

Parameter	Type	Values	Comments
<parameter>	char	station | vdifsize | bitspersample | bitsperchannel | net_protocol | mtu | ipd	Retrieve the value of any of the settable parameters

Monitor-only parameters:

Only one of the two fields below is returned, depending on wether the query had a <parameter> argument or not:

Parameter	Type	Values	Comments
<status>	char	connected | active | inactive	Current status of transfer.
<parameter value>	char		Current value of requested <parameter>

Notes:

1. The corner turner VDIF output section automatically selects an appropriate output VDIF frame size, unless explicitly overriden. The selection mechanism defaults to making one frame per output of the corner turner. If the output is sent to the network using udp or udps (==vtp) as protocol, however, it chooses otherwise. In this case it will compute the largest compatible VDIF frame size that will fit into the configured MTU and breaks up the corner turner output into an integer number of VDIF frames, if such a size exists.

sp*2*

SS_rev[12] – Get StreamStor firmware/software revision levels (query only)

Query syntax: SS_rev? ;
Query response: !SS_rev ? <return code> : <SS field1> : <SS field2>: ….. : <SS fieldn> ;

Purpose: Get information on StreamStor firmware/software revision levels.

Monitor-only parameters:

Parameter	Type	Values	Comments
<SSfield1> through <SSfieldn>	literal ASCII		Primarily for diagnostic purposes. The character stream returned from StreamStor, which is very long, is divided into 32-character fields separated by colons to stay within Field System limits. See Notes.

Notes:

1. 'SS_rev?' is a replacement for the old 'SS_rev1?' and 'SS_rev2?' queries; all three of these queries are now synonyms. *jive5ab* supports the 'SS_rev1?' and 'SS_rev2?' queries only on Mark5A systems.

SS_rev[12]

start_stats – Start gathering disk-performance statistics

Command syntax: start_stats = [<t0> : <t1> :….: <t6>] ;
Command response: !start_stats = <return code> ;

Query syntax: start_stats? ;
Query response: !start_stats ? <return code> : <t0> : <t1> :….: <t6> ;

Purpose: Start gather disk performance statistics

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<tn>	time		0.001125s 0.00225s 0.0045s 0.009s 0.018s 0.036s 0.072s	Clears and restarts gathering of drive statistics. See Notes. Seven optional values define 8 bins corresponding to drive-response (i.e. transaction completion) times; values must increase monotonically; a separate set of bins is maintained for each mounted drive. The count in a bin is incremented according to the following rules, where 't' is drive-response time of a single read or write transaction: Bin 0: t<t0 Bin 1: t0<t<t1 . Bin 6: t5<t<t6 Bin 7: t>t6

Notes:

1. Drive statistics and replaced-block counts are cleared and re-started whenever a new disk module is mounted or a 'start_stats' command is issued. Read drive statistics with 'get_stats' query. Bin values are common for all drives. Each count within a bin represents a transfer of 65528 bytes (2^{16}-8).
2. The 'start_stats' command may not be issued during active recording or readback.

start_stats

status – Get system status (query only)

Query syntax: status? ;
Query response: !status ? <return code> : <status word> [: <error number> : <error message> [: <error time>]] ;

Purpose: Get general system status

Monitor-only parameters:

Parameter	Type	Values	Comments
<status word>	hex	-	Bit 0 – (0x0001) system 'ready' Bit 1 – (0x0002) error message(s) pending; (message may be appended); messages may be queued; error is NOT cleared by this command. See also 'error?' query Bit 2 – (0x0004) not used Bit 3 – (0x0008) one or more 'delayed-completion' commands are pending. Also set whenever any data-transfer activity, such as recording, playing, or transfer to or from disk or net, is active or waiting. Bit 4 – (0x0010) one or more 'delayed-completion' queries are pending Bit 5 – (0x0020) Disk-FIFO mode Bit 6 - (0x0040) record 'on' Bit 7 - (0x0080) media full (recording halted) Bit 8 - (0x0100) readback 'on' Bit 9 - (0x0200) end-of-scan or end-of-media (readback halted) Bit 10 – (0x0400) recording can't keep up; some lost data Bit 11 – (0x0800) not used Bit 12 – (0x1000) disk2file active Bit 13 – (0x2000) file2disk active Bit 14 – (0x4000) disk2net active Bit 15 – (0x8000) net2disk active or waiting Bit 16 – (0x10000) in2net sending (on) Bit 17 – (0x20000) net2out active or waiting Bit 18 – (0x40000) DIM ready to record Bit 19 – (0x80000) DOM ready to play Bits 20-27 are set properly even if a data transfer is in progress. Bit 20 – (0x100000) Bank A selected Bit 21 – (0x200000) Bank A ready Bit 22 – (0x400000) Bank A media full or faulty (not writable) Bit 23 – (0x800000) Bank A write protected Bit 24 – (0x1000000) Bank B selected Bit 25 – (0x2000000) Bank B ready Bit 26 – (0x4000000) Bank B media full or faulty (not writable) Bit 27 – (0x8000000) Bank B write protected
<error number>	int	-	(optional) error number of error pending causing bit 1 to be set – see Note
<error message>	char	-	(optional) error message of error pending causing bit 1 to be set – see Note
<error time>	time	-	(since 2.6.0) system time when the error was added to the queue

Notes:

1. *jive5ab* returns the error details only if an error is pending, like Mark5A/DIMino does. This is undocumented behaviour for Mark5A/DIMino. In *jive5ab* the error is **not** cleared by this `status?` query, in direct contrast to Mark5A/DIMino.

status

task_ID – Set task ID (primarily for correlator use)

Command syntax: task_ID = <task_ID> ;
Command response: !task_ID = <return code> ;

Query syntax: task_ID? ;
Query response: !task_ID ? <return code> : <task_ID> ;

Purpose: Set task ID (primarily for correlator use)

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<task_ID>	int			For use with Mark 4 correlator only: Causes Mark 5 system to listen to only ROT broadcasts with the corresponding 'task ID'. See Notes.

Notes:

1. The 'task_ID' command is used in conjunction with the 'play' command for accurate synchronization of Mark 5 playback-start with correlator ROT clock.
2. On generic systems the task_ID command is recognized and ROT broadcasts are monitored but at the moment none of the data transfers available on those systems observe the actual ROT clock.

task_ID

track_check – Check data on selected track (query only)

Query syntax: track_check? ;
Query response: !track_check ? <return code> : <data mode> : <data submode> : <data time> : <byte offset> : <track frame period> : <track data rate> : <decoded track#> : <#missing bytes>;

Purpose: Check recorded track which, on playback, will output data to track pointed to by current 'track_set' value.

Monitor-only parameters:

Parameter	Type	Values	Comments
<data mode>	char	st | mark4 | vlba | tvg | SS	See 'mode' command for explanation of data modes; 'tvg' corresponds to VSI test pattern;' SS' corresponds to StreamStor test pattern '?' indicates unknown format. **Note** – If a Mark 5B module is present, a 'data_check' is done instead – See Note 6
<data submode>	char	8 | 16 | 32 | 64 | mark4 | vlba	'8|16|32|64' if <data mode> is 'mark4' or 'vlba'; 'mark4|vlba' if <data mode> is 'st'
<data time>	time		Time tag from next 'track' frame header beyond current play pointer. See Note 5 of 'scan_check'.
<byte offset>	int	bytes-	Byte offset from current play pointer to beginning of next 'track' frame header of target track
<track frame period>	time		Time tag difference between adjacent track frames; allows original track data rate to be determined.
<track data rate>	real	MHz	Track data rate of source data from formatter.
<decoded track#>	int		Track# decoded from auxiliary data field of target track; followed by 'D' if track is a 'duplicated' track; followed by'?' if unallowed track# in this position. See Note 3.
<#missing bytes>	int	bytes	Number of missing bytes between last and current 'track_check'; Should be =0 if immediately previous 'track_check' was within same scan Meaningless if immediately previous 'track_check' was in a different scan. See Note 4. See also Note 6 in 'scan_check'

Notes:

1. The 'track_check' query will be honored only if record and play are both off.
2. The 'track_check' query checks data beginning at the current position of the play pointer; the play pointer is not affected.
3. The 'track_check' query targets the first of the two selected 'track_set' tracks and executes the following actions:
   a. Determines the data mode/submode based on the format of the disk data.
   b. If the target track is a track which is actually recorded in this mode/submode (see 'mode' command Notes), several frames of data are collected from the expected position of this track in the disk data. If the target track is not recorded, the data are collected from the position of the recorded track number which, during playback, is duplicated onto the target track (see 'play' command Notes) in this mode/submode.
   c. A 'track frame header' is extracted from the collected data and the embedded <data time> and <track#> information is decoded. Note that the <decoded track#> will match the target track only in the case in which the target track was actually recorded.
4. Further analysis is done to determine the <track frame period> and <#missing bytes>. A 'blank' is returned in the <#missing bytes> field if the # of missing bytes cannot be calculated.

track_check

5. Regarding the 'data time' value returned by the 'data_check?', 'scan_check?' and 'track_check?' queries: The Mark 4 time-tags contain the day-of-year (DOY) but only the final digit of the year; the VLBA time-tags contain, instead, the last 3 digits of the Julian day number (misnamed MJD). To show the year and DOY in the returned values of 'data time' requires some assumptions. For Mark 4, we assume the most recent year consistent with the unit-year and DOY written in the Mark 4 time-tag; this algorithm reports the proper year provided the data were taken no more than 10 years ago. For VLBA, we assume the most recent Julian Day Number (JDN) consistent with the last 3 digits available in the VLBA time-tag; this algorithm reports the proper year provided the data were taken no more than 1000 days ago.
6. When a Mark 5B module is inserted into the Mark 5A (operating in so-called 'Mark 5A+' mode), a 'track_check?' query is meaningless since Mark 5B has no notion of 'tracks'. Instead, a 'track_check?' query performs a 'data_check'. This is done to maintain backward compatibility with existing Mark 4 correlator software.

track_check

trackmask – Configure channel dropping setup

Command syntax: trackmask = <bit mask> ;
Command response: !trackmask = <return code> ;

Query syntax: trackmask? ;
Query response: !trackmask? <return code> : <bit mask> ;

Purpose: Set up channel dropping data compression.

Settable parameters:

Parameter	Type	Comments
<bit mask>	hex	64 bit hexadecimal value 0x... Bit streams with a '1' in their position are kept. See Note 3.

Monitor-only parameters:

Parameter	Type	Comments
<bit mask>	hex	The current active bit mask. '0' means channel dropping is not active.

Notes:

1. The channel dropping mechanism is described in Section **8.1** of this manual.
2. The 'trackmask=' will return '1' ("initiated command but not finished") if the bit mask is not equal to 0. *jive5ab* is computing the compression algorithm and generating the (de)compression code in the background. It is only safe to start a transfer when 'trackmask?' does not return a '1' return code any more.
3. The bit mask resembles the function of the bit stream mask in the Mark5B `mode = ext : 0x...` command. It is vitally important to realize that the trackmask bit mask needs to be 64 bit value. If less than 64 bits are specified in the bit mask, the lower bits of the 64 bit mask will be filled in with the `<bit mask>` parameter, higher order bits are zeroed. As such, when dealing with 32 bit streams (e.g. on the Mark5B), the mask must be replicated twice or more data will be thrown away than expected.

trackmask

track_set – Select tracks for monitoring with DQA or 'track_check'

Command syntax: track_set = <track A> : <track B> ;
Command response: !track_set = <return code> ;

Query syntax: track_set? ;
Query response: !track_set ? <return code> : <track A> : <track B> ;

Purpose: The 'track_set' command serves a two-fold purpose: 1) to select two tracks to be output to the Mark 4 decoder or VLBA DQA and 2) to select the track examined by the 'track_check' query.

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<track A>	int/char	2-33 (hdstk 1) 102-133 (hdstk 2) | inc | dec	15	Track selected to be sent to DQA/decoder channel A; track to be analyzed by 'track_check'. Default is headstack 1; add 100 for headstack 2, if present. Track numbers follow the 'VLBA' convention; i.e. 2-33 for headstack 1, 102-133 for headstack 2. 'inc' increments current value – see Note 3; 'dec' decrements current value. If null field, current value is maintained.
<track B>	int/char	2-33 (hdstk 1) 102-133 (hdstk 2) | inc | dec	16	Track selected to be sent to DQA/decoder channel B. 'inc' increments current value – see Note 3; 'dec' decrements current value. If null field, current value is maintained.

Notes:

1. Note that tracks are duplicated according to the table in the Notes with the 'play' command. Any of the 'primary' or 'duplicated' tracks may be selected to go to the DQA/decoder.
2. <track A> is also used as the track to be examined by the 'track_check' query and should correspond to a track that is actually recorded in the selected data mode (see table with 'record' command).
3. The 'inc' value increments the current selected track value by one; cycles through all 32 tracks on each headstack, then begins again. This is a convenient method of cycling through all tracks during system testing.

track_set

tstat – Get current runtime status and performance

Command syntax: tstat = <ignored> ;
Command response: !tstat = <return code> : <time> : <status> [: <step #1> : <byte count #1> : ...] ;

Query syntax: tstat? ;
Query response: !tstat ? <return code> : <delta-time> : <status> [: <step #1> : <performance #1> : ...] ;

Purpose: The 'tstat' query/command return information of which transfer the runtime the command is sent to is doing (<status>) and how fast (query) or the raw byte counts since start-of-transfer (command) each step in the processing chain is operating.

Settable parameters: none. Any parameters given are ignored; it's only to differentiate between query or command. By providing time stamps and raw byte counts, an inquisitive application can do monitoring and/or differencing ("bytes/second") itself:

Parameter	Type	Comments
<time>	UNIX time stamp	UNIX time stamp when 'tstat=' command was processed, with fractional decimal seconds value added
<status>	ASCII	String describing which transfer is executing. 'idle' if nothing happening
<step #*N*>	ASCII	If <status> is not 'idle', the name of step #*N* in the current processing chain
<byte count #*N*>	ASCII	If <status> is not 'idle', the number of bytes processed by step #*N* in the current processing chain

Monitor-only parameters:

Parameter	Type	Comments
<delta-time>	double	Amount of seconds since last 'tstat?' query was processed
<status>	ASCII	String describing which transfer is executing. 'idle' if nothing happening
<step #*N*>	ASCII	If <status> is not 'idle', the name of step #*N* in the current processing chain (repeat for all steps)
<performance #*N*>	ASCII	If <status> is not 'idle', the processing speed of step #*N* in bytes/second in the current processing chain. The performance is computed by differencing the current byte count for step #*N* and its previous byte count and divide by <delta-time>

Notes:

1. The 'tstat?' query version does not attempt to deal with multiple simultaneous pollers. So if two users and/or applications are polling 'tstat?' in the same runtime, the returned delta-time may not be the poller's intended poll-time interval. For predictable time intervals the application should use the 'tstat=' command version and do the differencing/dividing-by-time-span itself.
2. If <status> is not 'idle', the reply always contains pairs of <name> and <count> (or <performance>). The <name> of a processing step is usally self-explanatory.

TVR – Start TVR testing

Command syntax: TVR = <tvr mask> ;
Command response: ! TVR = <return code>

Query syntax: TVR? ;
Query response: ! TVR? <return code> : <status> : <tvr mask> : <tvr error> ;

Purpose: Start TVR testing

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<tvr mask>	hex		Current bit-stream mask	Defines bit streams to be tested; reset <tvr error> flag to 0.

Monitor-only parameters:

Parameter	Type	Values	Comments
<status>	int	0 | 1	0 – TVR not running 1 – TVR running
<tvr mask>	hex		
<tvr error>	int	0 | 1	0 – no errors detected 1 – at least one error detected; reset <tvr error> flag to 0

Notes:

1. The TVR receives data from the VSI input and can be used only when the ext (VSI) clock has been selected by the 'clock_set' command. After issuing the 'TVR' command, the TVR will start operation on the next DOT 1pps tick. Only the bit-streams specified in the <tvr mask> will be tested; only undecimated TVG data can be tested. The tested bit streams must appear on the VSI input bit-streams in the positions expected for full 32-bit TVG pattern. A 'TVR?' query may be issued anytime after the TVR has started. Any single bit error on any of the selected bit-streams sets the <tvr error> status bit to '1', then and clears the <tvr error> in anticipation of the next 'tvr?' query.

TVR

(un)mount – power up/down bank (as if keying the bank on or off) (jive5ab >= 2.8.2)

Command syntax: mount = <bank> [: <bank>] ;

unmount = <bank> [: <bank>] ;

Command response: !(un)mount = <return code> ;

Purpose: Power one or more banks up or down, as if a bank's key was turned

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
<bank>	char	A|B		Only banks 'A' and 'B' are valid bank labels

Notes:

1. The key on the Mark5 front panel switches power to the corresponding disk pack on or off and brings the disks up (or down) in a controlled fashion. This power management is done by the StreamStor firmware. The firmware responds to a key transition from 'off' to 'on' (bring power to the disk pack) or from 'on' to 'off' to power the disk pack off. This powering on/off can also be triggered via the Conduant StreamStor SDK and since jive5ab 2.8.2 through these VSI/S commands.

(un)mount

version – Get detailed version information of this *jive5ab®* (query only)

Query syntax: version? ;
Query response: !version ? <return code> : <program> : <version> : <#bits> : <release> :
<build info> : <StreamStor SDK path> [: <more gunk>]* ;

Purpose: Get detailed properties of the *jive5ab* binary.

Monitor-only parameters:

Parameter	Type	Values	Comments
<program>	char	jive5ab	If really talking to *jive5ab*™ it will say `jive5ab` here.
<version>	char	x.y[.z[-gunk]]	Numeric version, at least *major.minor*. Sometimes including *patchlevel* and extra *gunk*. See Note 2. Updated ≥ 3.0.0
<#bits>	char	32bit | 64bit	Wether *jive5ab* was compiled as 32-bit or 64-bit binary
<release>	char	dev | release Debug | Release	Wether this was a development or a release build. *jive5ab* ≥ 3.0.0 is built using CMake, which has its own, built-in, configuration management which is now adopted by *jive5ab*.
<build info>	char		Host name and host's date and time at the time when this binary was compiled. See Note 1.
<StreamStor SDK path>	char		The path to Conduant's SDK library that this *jive5ab* was linked with. See Note 3.
<more gunk>	char		In releases ≥ 3.0.0 more optional functionality might be compiled in, the gunk will contain information which.

Notes:

1. Each time *jive5ab* is compiled, the host name and host's date/time are compiled into the binary.
2. *gunk* may be 'FiLa10G' if this *jive5ab* was compiled with the `FILA=1` make command line argument. This compile-time switch makes *jive5ab* deal correctly with the broken 64 bit sequence numbers that the RDBE and FiLa10G output when the output format is set to mark5b on the latter. *jive5ab*'s compiled with this flag will fail to read data from correctly sent sequence numbers, e.g. when FiLa10G's output format is changed to VDIF, or when data from another *jive5ab* is attempted to read. Starting from *jive5ab* 3.0.0, using CMake's optional configuration settings, these are reported separately through the <more gunk> fields.
3. If no StreamStor SDK was available (e.g. on generic hardware) or when compiling with StreamStor support was disabled, this will read 'nossapi' – "no streamstor API".

version

VSN – Write extended-VSN to permanent area

Command syntax: VSN = <VSN> ;
Command response: !VSN = <return code> ;

Query syntax: VSN? ;Query response: !VSN ? <return code> : <extended VSN> : <status> :
[: <disk#> : <original S/N> : <new S/N> : 'Disk serial-number mismatch'] :
<companion extended VSN> : <companion bank>;

Purpose: Write module extended-VSN (volume serial number) to permanent area on active module

Settable parameters:

Parameter	Type	Allowed values	Default	Comments
VSN	char			Permanent 8-character VSN, analogous to tape VSN, which survives 'reset=erase' command and module conditioning (example: 'MPI-0153'). VSN format rules are enforced – see Note 4. The module capacity and maximum data rate for the extended-VSN are calculated and appended to the VSN to create the 'extended-VSN' (example 'MPI-0153/960/1024'). For non-bank-mode modules, see Note 7.

Monitor-only parameters:

Parameter	Type	Allowed values	Comments
<extended VSN>	char		Example: 'MPI-0153/960/1024'; see Notes 4 and 5. For non-bank-mode modules, see Note 7.
<status>	char	OK | Unknown | Fail	OK – disk serial #'s on current set of disks matches serial #'s when VSN was last written. Unknown – disk serial #'s have not been written Fail – current disk serial #'s do not match serial #'s when VSN was last written. See Note 6.
Following parameters are returned only if <status> is 'Fail":			
<disk#>	int	0-7	First disk# in module in which there is a serial-number discrepancy
<original S/N>	char		Serial number of disk in position <disk#> when VSN was written
<new S/N>	char		Serial number of disk now in position <disk#>
'Disk serial-number mismatch'	char		Warning message
<companion extended VSN>	char		If non-bank-mode module, returns VSN of companion non-bank module. See Note 8.
<companion bank>	char	B | A	Bank position of companion non-bank module

Notes:

1. The 'VSN=..' command is normally issued only when the module is first procured or assembled, or when the disk configuration is changed. The serial numbers of the resident disks are noted.
2. The 'VSN?' query compares the serial numbers of the original disks to the serial numbers of the currently-resident disks and reports only the first discrepancy. Issuing a 'VSN=…' command or a 'reset=erase' command will update the disk-serial# list to the currently-resident disks.

VSN

3. A 'protect=off' command is required *immediately* preceding a 'VSN=' command, even if protection is already off.
4. The format of the extended-VSN is "VSN/capacity(GB)/maxdatarate(Mbps)" – example 'MPI-0153/960/1024'. The following rules are enforced by the *dimino*:
   a. VSN – Must be 8 characters in length and in format "ownerID-serial#" (for parallel-ATA modules) or "ownerID+serial#" (for serial-ATA modules, when they become available)
   b. ownerID – 2 to 6 upper-case alphabetic characters (A-Z). The 'ownerID' must be registered with Jon Romney at NRAO (jromney@nrao.edu) to prevent duplicates. Numeric characters are not allowed. Any lower-case characters will automatically be converted to upper case.
   c. serial# - numeric module serial number, with leading zeroes as necessary to make the VSN exactly 8 characters long. Alphabetic characters are not allowed in the serial#.
5. *dimino* and `jive5ab` will compute the capacity of the module in GB and the maximum data rate in Mbps (number of disks times 128 Mbps) and append these to the VSN to create the extended VSN. Module capacity in GB is calculated as capacity of the smallest disk, rounded down to nearest 10GB, and multiplied by the number of disks in the module.
6. ~~The recorded disk serial #'s are updated each time a scan is recorded.~~ This is way too expensive (in disk access) so only Note 2 applies.
7. ~~A "VSN=.." command may not be issued to any module which has been initialized in non-bank mode.~~ `jive5ab` allows writing a VSN always, updating the extended VSN with the capacity and maximum record rate of what is currently active, bank or non-bank mode.
8. ~~When a non-bank-mode pair of modules is mounted and the unit is operating in non-bank mode, a "VSN?" query will return the VSN of both modules as indicated in the return parameters.~~ Because of changes in Note 7, the `vsn?` query will always return the actual VSN of the active module, wether it is bank or non-bank-mode. Use the `bank_set?` query to find the original constituent VSNs of a non-bank-mode pair. See Section **11**.
9. ~~When only a single module of a non-bank-mode module-pair is mounted, a "VSN?" query will return the both the VSN of the mounted module plus the VSN and bank position of its unmounted companion; however, no reading or writing of data will be allowed in this situation.~~ Because of changes in how non-bank-mode is implemented this behaviour is also not true. See Section **11** for all details of what (can) happen(s) if only a single module is inserted and any data access is attempted. The `VSN?` and `bank_set?` queries always work.