

Gets32 Data Acquisition Protocol Overview

Updated: August 20, 2007

Kevin Cahill, October 8, 2003

From Emerging Data Acquisition Protocols

Request For Comment Feb 18, 2000

Introduction

This document defines the Gets32/Sets32 data acquisition protocols. This protocol is a superset of the Retdat/Setdat protocol with the most significant differences including a 32 bit length and offset specification, an expanded data event specification string, and returned timestamps.

The VMS data pools have not and are not likely to implement this protocol. Consequently, front-ends should continue to support the Retdat/Setdat protocol.

Linear addressing beyond 16 bits is likely to be exploited by few front-ends. The expanded data event specification string reduces the delays incurred with soft collection on event by moving the event monitoring to the front-end. The returned timestamps provide the means to correlate data and measure some performance metrics.

This description is intended as a starting point for someone to formally document this protocol.

Protocol Framing

GETS32/SETS32 Requests

The request header is type-coded and carries information for the destination front-end as well as intermediate, consolidating engines. The message header for GETS32 and SETS32 is the same and contains:

typecode (byte)	// type code, presently: (1)
majorVersion (byte)	// protocol version (1)
minorVersion (byte)	// protocol version (0)
acnetTrunk (byte)	// front-end trunk
acnetNode (byte)	// front-end node
orderFlag (byte)	// Gets32 (0), Sets32(1=>forward settings to database)
priority (byte)	// 0=>low, 1=>normal, 2=>high
isRepetitive (byte)	// event is repetitive when non-zero
replySize (int)	// expected reply size
packetCount (short)	// number of packets
classicFTD (short)	// classic FTD, or "-1" when incomplete
dataEventStringLength (short)	// length of the event string
dataEventString (string)	// terse event string

The remainder of the request buffer differs from RETDAT/SETDAT in that lengths and offsets are specified in 32 bits.

dipi (int)	// device index with property index in high byte
ssdn(8 byte)	// subsystem device number
length (int)	// length
offset (int)	// offset

The Java Data Acquisition Engine data pools, its Open Access Client architecture, and its consolidation task POOL32 support the GETS32 and SETS32 protocols.

GETS32/SETS32 Replies

The reply header for GETS32/SETS32 will contain:

globalStatus (short)	// global status
typeCode (byte)	// type codes (1)
majorVersion (byte)	// protocol version (1)
minorVersion (byte)	// protocol version (0)
orderFlag (byte)	// undefined at this time (0)
replySequenceNumber (int)	// incrementing sequence reply number
cycleTimestamp (long)	// timestamp of most recent multicast 0x0f event
collectionTimestamp (long)	// timestamp appropriate for the data collection event
replyTimestamp (long)	// timestamp appropriate for the data return

Return timestamps support correlation. Ideally, data collection on state transition or clock event from dissimilar front-end architectures will return identical cycle and collection timestamps.

Timestamps are expressed in milliseconds since January 1, 1970 00:00:00 GMT.

The cycle time stamp should be calculated from the UCD (Unibus Clock Decoder) module clock reports. In any one second period of 1000 milliseconds, only 15 valid cycle time stamps exist where each of those 15 were reported and time-stamped by the UCD module as Tevatron 0x0f clock event events. Alternately, the cycle sequence number as reported by the UCD as an unsigned integer can be provided along with a zeroed high order integer for the cycleTimestamp. In that case, the receiving software will convert the cycle sequence number into the appropriate timestamp.

The collection time stamp should be calculated from the collection event. All front-ends reporting on a collection of 1000 milliseconds after a super cycle reset should report the same collection time stamp as the time of the super cycle reset as reported by the UCD module plus 1000 milliseconds. In practice, the front-ends, engines, and higher order software may maintain accurate microsecond or nanosecond timestamping that when reduced to milliseconds can differ by a millisecond unit or so.

The reply time stamp should indicate when the reply was ready for network transmit. Generally, the time stamps should follow in time-order of cycle, collection, and reply.

The remainder of the replies from GETS32/SETS32 is identical to RETDAT/SETDAT replies.

Data Events

Presently, valid data events are of four types: immediate, periodic, on event, or on state transition.

The event string specification is the result of the string returned from the toTerseString method of the DataEvent Java class. The elements of the string are comma separated, and the first character in the string is the first character of the four valid types: immediate, periodic, event, and state. Delays are specified as the decimal representation of a longword number of milliseconds of delay. Other decimal strings may be assumed to be converted to an int.

The immediate event string contains no other elements. Examples are limited to: "i".

The periodic event string's next element contains the decimal number of milliseconds between returns. The last element in a periodic string is a boolean string. When true, returns are requested immediately and at the periodic rate. When false, returns may be sent at the periodic rate without an immediate return. This once immediately boolean is a suggestion. Examples include "p,1000,true", "p,66,false", and "p,60000,true" where they describe one Hertz, fifteen Hertz, and once a minute returns respectively.

The clock event string's second element is the hexadecimal representation of a single Tevatron clock event. The third element is the character 'h', 's', or 'e' representing hard, soft, or either clock event respectively. The last element is the decimal number of milliseconds of delay. The string "e,0,h,5000" expresses a delay of 5 seconds after a hard supercycle reset event.

The state event string's second element is the device name of the state device. The third element is a

decimal compare value. The fourth element is the decimal delay in milliseconds, and the last element is the compare operator string. The compare operator strings represent any state announcement ('*'), an announcement with a value equal to the compare value ('='), and an announcement with a value not equal to the compare value ('!='). The string "s,V:CLDRST,9,1000,=" expresses a delay of one second after the collider state device (V:CLDRST) reading announces(=) the inject pbars (9) state. The multicast state transition events announce device indices instead of device names. If the maintenance of state device names to indices is problematic, the state event might be described with a device index instead of name.

Ping Replies

It has been demonstrated that when a consolidating engine is restarted, and engines requesting data from front-ends served by the consolidating engine resend their requests through the consolidating engine, messages may be lost as input buffers on the consolidating engines are overwhelmed by the multitude of incoming, concurrent requests. Despite setting input buffer sizes to the maximum allowed by the operating systems, this behavior is readily observable in our system. The same phenomena is likely to be experienced by front-ends in time as increasing clock event and delay combinations are specified in our data loggers.

There is a slow recovery mechanism in the engines that restarts 'quiet' data acquisition requests every 90 minutes. A faster recovery mechanism is desired.

Repliers should periodically send a zero length, status only reply to multiple reply requests. The status returned should be an ACNET pend. That is 0x0101. The engines will restart requests that have not been replied to within the last 120 seconds. The engines themselves as GETS32 repliers wake every 90 seconds and send a ping reply to any multiple reply request that has not been replied to within the last 20 seconds.