1.0 Version Control

Interface Control Document 4.1 - Generic Pixel Server Communications, Command/Response and Data Stream Interface Description

Version/Change Record

Version	Date Approved	Sections Affected	Reason/Remarks
0.0.1	20020311	All	First Released Draft
0.0.2	20020412		

Created:January 7, 2002
Modified:December 8, 2003

NOAO Detector R&D Group Report

Interface Control Document 1.1 - Data Handling System Interface

Input - Status and Data Stream Description

Nick C. Buchholz (NOAO

ICD_1.1_DHS_Interface.fm/V0.1

The output of the Generic Pixel Server (GPX), the Instrument Control System (ICS) and the Generic Manager of Instruments (GMI) must all be combined at some single point to allow the data handling system to reduce and use the data. This document describes the message interface to this DHS interface program. It includes the Data/Status stream used by a Generic Pixel Server and the status and event messages produced by the ICS and GMI layers.

1.0 Introduction

1.1 Scope

The Data and Status outputs from the **GPX** (ICD 4.0) have so far been left unspecified. It has been agreed to some extent that a message based method for publishing the data and status has significant benefits over a file based technique. This document serves the purpose of isolating the discussion of the data and status streams from the remainder of the ICD 4.0 discussion.

In addition while developing the ICD 4.0 description of the Data and Status outputs, we reached the conclusion that an ICD describing what the DHS Interface can expect to receive from the entire Observation Control System was more appropriate. It is assumed that the GPX will confom to this ICD completely.

The document is divided into the following sections:

- **Section 1.0, "Introduction,"** , this introduction.
- **Section 2.0, "DHS Application Programmer Interface"**
- **Section 3.0, "DHS Interface Message Structure,"** , which describes the communications methods for the **GPX**, including the Data and Status message stream.

The intended audience for this document is:

- Anyone with an interest in the Generic Pixel Server Interface.

1.2 Acronyms and Glossary

1.2.1 Abbreviations and Acronyms

See ICD 4.0 Section 1.2.1

1.2.2 Glossary

See ICD 4.0 Section 1.2.2

1.2.3 Observation Sequence Definitions and Association IDs RULE

1.3 Reference Documents

1. SPE-C-G0037, “Software Design Description”, Gemini 8m Telescopes Project.
2. “ICD/16 — The Parameter Definition Format”, Steve Wampler, Gemini 8m Telescopes Project.
3. WHT-PDF-1, “FITS headers for WHT FITS tapes”, Steve Unger, Guy Rixon & Frank Gribbin, RGO.
4. NOST 100-1.0, “Definition of the Flexible Image Transport System (FITS)”, NASA Office of Standards and Technology.
5. GEN-SPE-ESO-00000-794, “ESO Data Interface Control Document”, Miguel Albrecht, ESO.
6. IEEE Std. 610.12-1990 - “IEEE standard glossary of software engineering terminology”, Standards Coordinating Committee of the IEEE Computer Society, USA, 19901210
7. ANSI/IEEE Std 754-1985 - “IEEE Standard for binary floating-point arithmetic” - Standards Committee of the IEEE Computer Society, USA 19850812
8. xxxx “XDR - Extended data representation Standard” ????
9. ICD 6.0 - “Generic Detector Controller - Command and Data Stream Interface Description”, Nick C. Buchholz(NOAO), Barry M. Starr(NOAO), Version 0.1.2, Dated: 200203011- NOAO Document number ##.##.#
10.

FIGURE 1. Observatory System Reference Model

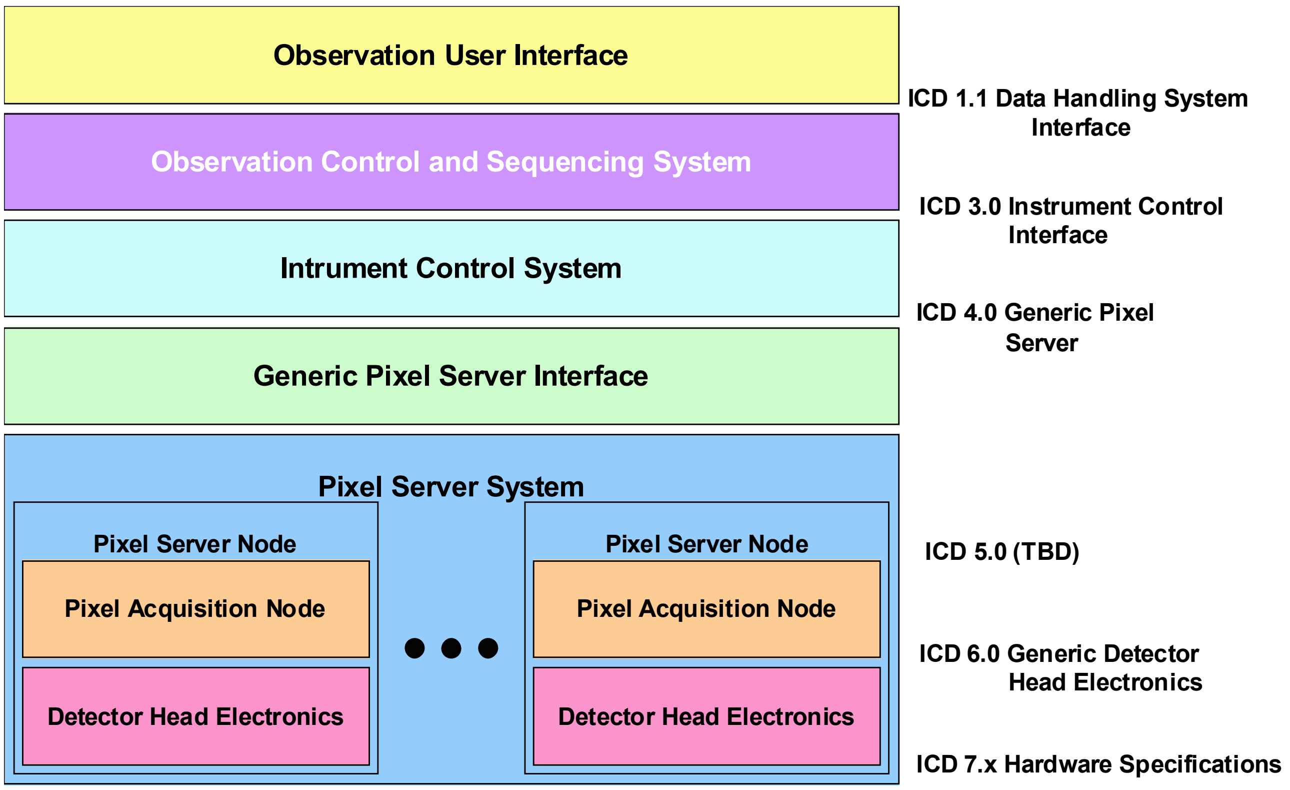
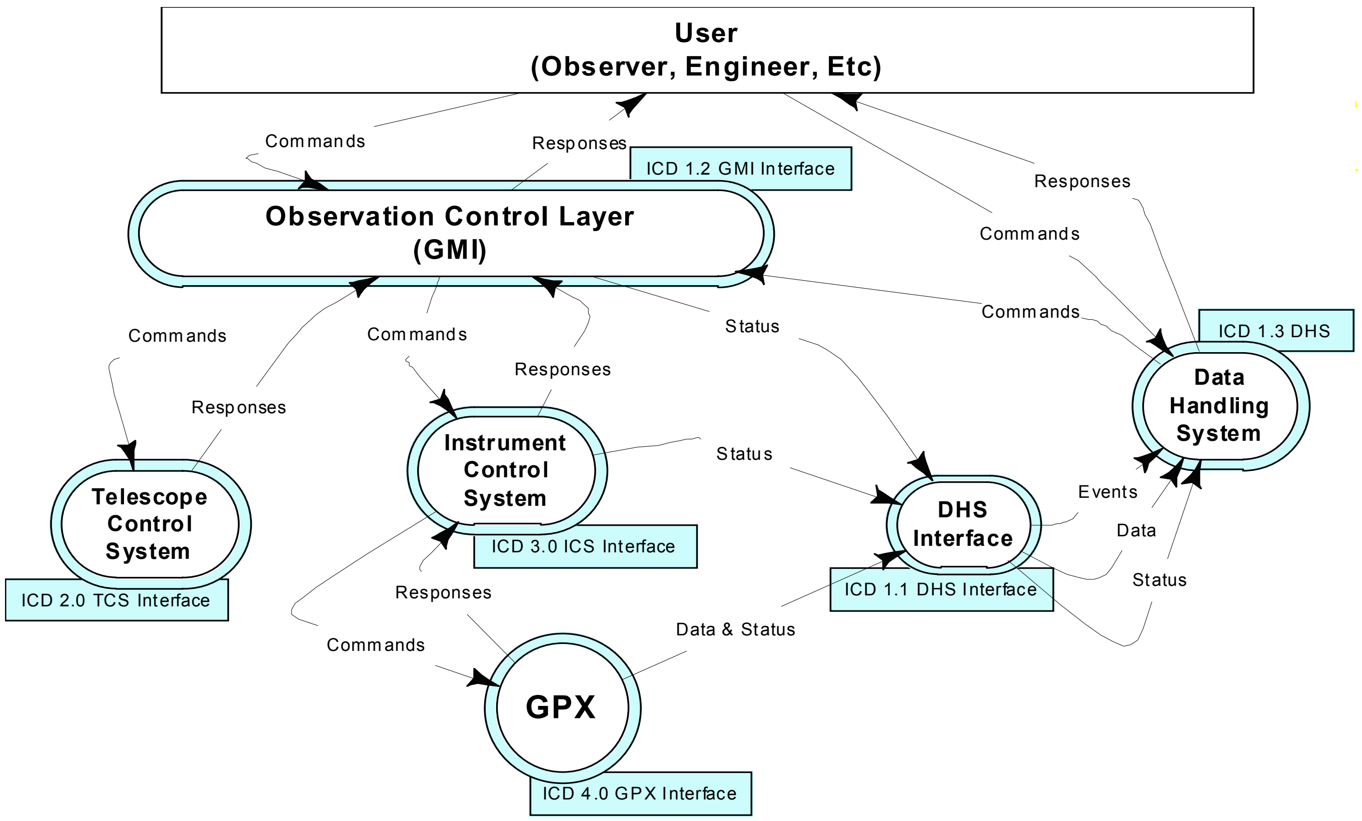

Observation User Interface
ICD 1.1 Data Handling System Interface
Observation Control and Sequencing System
ICD 3.0 Instrument Control Interface
Intrument Control System
ICD 4.0 Generic Pixel Server
Generic Pixel Server Interface
Pixel Server System
Pixel Server Node
Pixel Acquisition Node
Detector Head Electronics
Pixel Server Node
Pixel Acquisition Node
Detector Head Electronics
ICD 5.0 (TBD)
ICD 6.0 Generic Detector Head Electronics
ICD 7.x Hardware Specifications

FIGURE 2. System Context Diagram



2.0 DHS Application Programmer Interface

The DHS interface API will be included entirely within a single shared library loaded at runtime into the data publication process (panSaver) of the GPX. the library may use whatever data transfer method the DHS group deems appropriate. The nature of this protocol will be buried within the DHE API library.

2.1 Standard Calling Sequence for DHS API Functions

Each DHS API function will have a defined parameter set as follows:

*dhsOpen(long *status, char *response, dhsHandle dhsId, { additional parameters});*

2.1.1 Parameter 0 - long *status

This parameter peforms two functions the first it gives a status value which is inherited from earlier routines. Second it provides a means for each routine to report its status to later routines. On entering each routine the routine should check the incoming status value and simply return if the Status indicates a prvious error.

2.1.2 Parameter 1 - char *response

This parameter is a character buffer pointer which provides a means for status information to be returned to the upper levels of the process and eventually the user. The pointer may be null however if it is non-null it must point to a memory buffer which contains at least 4096 bytes. Each routine should detect the presence of this address and fill in the buffer only if the pointer is non-null.

2.1.3 Parameter 2 - dhsHandle dhsId

Except for the dhsOpen command each dhs routine will pass in a handle parameter obtained by doing a dhsOpen. The dhsOpen routine will pass in a dhsHandle pointer (dhsHandle *) in which the rotuine will store the handle to a successfully opended DHS interface or an ERROR(-1) value.

2.2 DHS API Routines

2.2.1 dhsOpen -

*dhsOpen(long *status, char *response, dhsHandle *dhsID)*

2.2.2 dhsClose -

*dhsClose(long *status, char *response, dhsHandle dhsID)*

2.2.3 dhsSendAVBlk -

*dhsSendAVBlk(long *status, char *response, dhsHandle dhsID, char *blkAddr, long blkSize)*

2.2.4 dhsSendData -

*dhsSendData((long *status, char *response, dhsHandle dhsID, char *blkAddr, long blkSize, long dataSize)*

2.2.5 dhsASyncMsg

*dhsASyncMsg(long *status, char *response, dhsHandle dhsID, char *msgAddr, long msgSize)*

2.2.6 dhsASyncResp

*dhsASyncResp(long *status, char *response, dhsHandle dhsID, char *msgID, long msgSize)*

2.2.7 dhsGetExpId

*dhsGetExpID(long *status, char *response, dhsHandle dhsID, long *ExpID, long *idSize)*

2.2.8 dhsStartObs

*dhsStartObs(long *status, char *response, dhsHandle dhsID, long *ObsID, long *idSize)*

2.2.9 dhsEndObs

*dhsEndObs(long *status, char *response, dhsHandle dhsID, long ObsID, long *idSize)*

2.2.10 dhsGetObsId

*dhsGetObsId(long *status, char *response, dhsHandle dhsID, long *ObsID, long *idSize)*

2.2.11 dhsIoctl

*dhsIoctl(long *status, char *response, dhsHandle dhsID, long ObsID, long ExpID, ...)*

3.0 DHS Interface Message Structure

The status and data stream message protocol to the DHS interface is defined in this section. The protocol uses messages passed over a socket stream to send pixel, status and event data to the outside world. The messages are considered to be a stream of bytes which are interpreted by the receiving process/client to be in one of the defined formats.

3.1 Storing messages RULE

The bytes of a message SHALL BE stored in memory by the receiving process with the lowest numbered byte of the message in the lowest memory address.

3.2 Byte order RULE

All messages SHALL use network byte ordering. Bytes in a message are labeled from 0 to N (where N is the length of the message).

3.2.1 Creating messages RECOMMENDATION

The sending software SHOULD build the message using htonl() or htons() or similar routines to convert from host ordering to network ordering. The receiving software SHOULD convert the message byte order into something which is usable locally using ntohl() or ntohs() or similar functions.

BYTE 0	BYTE 1	BYTE 2	BYTE 3
Message Type	*Protocol Version*	*Exposure Identifier*	

BYTE 4	BYTE 5	BYTE 6	BYTE 7
Message Body Length in bytes (unsigned)			

BYTE 8	BYTE 9	BYTE 10	BYTE 11
Message Source IP Address			

3.2.2 Strings RULE

Strings which are embedded in a message SHALL BE inserted with the left most character of the string in the lowest order message byte. i.e. string “ABCD” will appear in message Byte N through N+3 with ‘A’ in byte n and ‘D’ in byte n+3.

3.2.3 Very Long integers RULE

The protocol implementers SHALL make provision for 64 bit integers by using the network ordering decision used for long integers. In the messages the Most Significant Byte of a very long integer SHALL BE sent first. i.e. will be closest to the start of the message.

3.2.4 Floating Point and Double Values **RULE**

Real numbers represented by floats or doubles in ‘C’ SHALL BE represented in the messages in IEEE floating point format (see **[7]**). The byte ordering SHALL BE as defined in the XDR/Network standard.

3.3 Message structure **RULE**

DHS Interface messages SHALL have the following structure:

1. Message Header Bytes - 12 bytes describing the message. Followed by:
2. Message Body Bytes - 1 to 4294967295 bytes containing the body of the message. Followed by:
3. Message Error Check Bytes - 4 bytes which implement a CRC on the message body and header.

3.3.1 Message Header RULE

The message header SHALL have the structure described in Figure 3, “Message Header Structure Description RULE,” on page 8 Below:

FIGURE 3. Message Header Structure Description RULE

3.3.1.a Message Type Field RULE

The first byte (byte 0) of the header is the Message Type Field. This field SHALL contain a one byte descriptor of the type of data to expect in the message. Figure 4, “Message Type Field Descriptions RULE,” on page 8 details the message types available.

FIGURE 4. Message Type Field Descriptions RULE

Message Type Name	Byte Code	Explaination of type
Control Msg	0x01	Control Messages are used to change the behavior of the DHS interface They include commands which modify how data and events are packaged and processed.
Keyword Msg	0x02	Keyword messages are used to change the value of a keyword used by the DHS. They include information which may appear as FITS headers and other information which may be maintained with the data not in the image header
Event Msg	0x03	Event messages are used to announce events which occur in the Observation Control Layer, the GPX, ICS and TCS systems.
Pixel Data Msg	0x04	Pixel Data messages are used to transfer the pixel data from the GPX to the DHS Interface for processing and storage.
Header Block Msg	0x05	Header Block messages are used to transfer FITS header information to be associated with an exposure.
Fixed Data Block	0x06	Fixed Data block messages are used to tranfer infomation to be associated with an Exposure, Observation, DitherSet, Science Field or Science Program. It is intended that this information will be associated without interpretation or modification by the DHS system.

3.3.1.b Protocol Version Field

3.3.1.c Exposure ID Field

3.3.1.d Message Body Length

3.3.1.e Message Source IP Address

3.3.2 Message Body Rule

The message body SHALL BE a number of bytes equal to the Message Body Length in the message header field. The contents of the message body are interpreted by the receiving system in accordance with the Message Type and Protocol Version stored in those message header fields.

3.3.3 Message Error Check Rule

The Message Error Check SHALL BE a four byte CRC value calculated by doing a “exclusive or” operation on all other of all other bytes in the message in groups of four bytes.

4.0 DHS Interface Message Types

4.0.1 Control messages

Control messages are used to perform some sort of control function in the Data Consumer. The following messages may be generated by the GPX and will have to be dealt with by the Data Consumer

4.0.1.a Start Association

4.0.1.b End Association

4.0.1.c Flush

4.0.1.d All Data Sent

4.0.1.e Disconnect

4.0.1.f Reset

4.0.1.g Refresh Status Screen

4.0.1.h Start Logging

4.0.1.i End Logging

4.1 Command/Response Communications Stream Definition — Rule

The Command/Response communications streams used by the Generic Pixel Server take place over a *socket* connection. The control layers which use the **GPX** connect to the **GPX** by connecting to a IP address and socket port number.

4.1.1 Socket Connections — Observation

While it is not necessary that a socket connection be used, it will simplify the system to make this a rule. It is possible to serialize the command response streams in another way i.e. as a result of an RPC call, OS driver call or a serial hardware line. However we feel the *socket connection* is the most reliable and OS independent method to use.

The command/response set only assumes that the underlying communications protocol to the Pixel Server delivers a single ASCII string to the command processor for each command issued.

4.2 Socket Definitions — Rule

The connection to a **GPX** will be by a *set* of socket connections. Assuming the primary communication socket (commands) is on Port *N* (TBD), a second socket on port N+1 would be used for the command response stream and a third, on port N+2, would be used for the Asynchronous Status message stream.

4.3 Status and Data Stream Interface — Rule

The status information and data output from the **GPX** shall be a stream of messages across a socket connection. Any client wishing to use the data produced by the **GPX** may connect to this socket and receive the message stream.

4.3.0.a Data output minimum — **Rule**

Every **GPX** will at a minimum have a mode which is able to produce a **FITS** formatted image on a local disk for engineering and diagnostic purposes. This capability is in addition to the message based Data interface

4.3.0.b Status output minimum — **Rule**

Every **GPX** will at a minimum have a mode which allows the display of status information on a terminal- like display, as lines of status messages.

4.3.0.c Local Client / Remote Client Message formats — Rule

The message formats, for some messages, will differ based on whether the client is connected locally or remotely. Local clients, running on the same machine as the GPX, will receive pixel Data through a shared memory mechanism to avoid data copying where un-neccessary. Remote clients will receive data across the socket.

4.3.0.d Final data stream — Observation

The final data stream from the **GPX** will be partially determined by local protocols.

4.3.0.e Data Stream Minimum — Permission

It is possible for a implementation **GPX** to produce a stream of FITS images to export the data. This stream could then be transformed into the **GPX** standard format or into some local format i.e. Gemini DHS, NOAO PicFeed, IDL, etc. by an auxiliary process.

4.3.1 Keyword messages — Rule

Keyword messages are used to inform the final user of the data as to the value of an attribute of the data.a keyword message consists of

4.3.1.a

4.3.2 Interpretable Header block messages — Rule

Interpretable Header block messages are strings which represent FITS header information to be included with the final image. This info is not examined or acted on by the **GPX** and may be intrepreted and modified as required by the FITS standard or local DHS practices.

4.3.2.a

4.3.3 Inviolate Header block messages — Rule

Inviolate Header block messages are strings which represent FITS header info to be include with the final image. This information is required to be included in the form presented without modification.

4.3.3.a

4.3.4 Event messages

GPX to indicate to the data consumers that a particular event has taken place.

4.3.4.a Readout Start

4.3.4.b Readout End

4.3.4.c Integration Start

4.3.4.d Integration Time Remaining

4.3.4.e Exposure Start

4.3.4.f Exposure End

4.3.4.g System Reset

4.3.5 Pixel Block messages

Pixel Block messages are blocks of pixel data being sent. A short preamble in the message body describes the format and unscrambling needed by the data.

4.3.5.a Message Length is number of 4 byte integers in message

4.3.5.b Preamble gives unscrambling and location information

4.3.5.c

Appendix I. Keywords produced by A GPX system

This table is totally incomplete I wanted to get a more defined basic structure before we went into these details so this can be ignored with the knowledge that most of the information about the state of the GPX will be preserved by the final system

Keyword Name	Usage/Explanation
numArrays	An integer value giving the number of arrays in the focal plane.
arrayDescriptor type rows columns outputsPerArray	A structure describing the characteristics of the array being controlled. The components are: a string giving the type of array, Two integers giving the size in rows and columns and an integer giving the number outputs on the array. Other elements may be needed for certain arrays.
outputArrangement picQueue baseR, baseC strideR, strideC chunkR, chunkC sizeR, sizeC	A structure which outlines how the array outputs are read. This includes a queue descriptor which tells where to place the pixels for processing and information on the structure of the pixel data block transferred. We describe the block of data as chunks of contiguous pixels separated by a intervening pixels from other blocks. The block is described by a number of integers giving the starting row and column of the block, a row and column stride (the number of pixels to skip when storing chunks) the row and column chunk size and the total size of the final block of data in rows and columns
spcDescriptor gain settlingTime offset noiseFoM	A structure which describes the configuration of the signal processor chains in the system. The components of the structure are floating point arrays which describe the gain, settling time, and offset of the signal processing chain. Included is a noise figure of merit (TBD) which will allow the quietest set of chains to be chosen when that is important.
waveForms	A descriptor for the timing waveforms to be run when running the array. These will be an array of bytes which will either describe of define the timing of the array readout. It is expected that each system will have an idiosyncratic way of describing these waveforms.
DacValueN - float	An array of floating point voltage values which are to be loaded into any DAC settable voltages used to control the array. Each system will likely have a unique set of these voltages and an mapping from voltage name to DAC number should be provided in the **GPX**
Min Integration Time	A floating point number giving the minimum integration time achievable by the system
Base Readout time	A floating point number giving the fastest possible readout time for the entire array.
spdRoiDescriptor Row0 Col0 rowSize ColSize	A structure which describes a “speed up” region of interest (ROI). This is provided so a system with a large array can describe a sub array which will be readout to provide faster readout and shorter integration times. (Mostly used for IR systems without an internal cold shutter) The ROI is described by four integers giving the First row and column to be read and the size of the ROI in rows and columns.
binning	An integer value giving the binning factor for the array readout. This may be two values if the row and column binning factors are different.
intTimeSecs	A floating point number giving the desired integration time to use in seconds.
digAvgs	An integer giving the Number of Digital Averages to use while reading out the pixels
numPics	An integer giving the Number of pictures to generate for each gpxStartExp
ROIdescrptors Row0 Col0 rowSize ColSize	A list of structures defining the regions of interest (ROI) to readout and archive. The components of the structure are four values representing the first row and column included in the ROI and the row and column size of the ROI
Outputs to Read	An integer giving the number of outputs on the Array which will be used during the readout.

Keyword Name	Usage/Explanation
PreFlash	A boolean Value determining if the exposure sequence will include a pre-flash step.
waveFormsToRun	A list of the Waveforms to run during this array readout.
shutterState	A boolean Value determining if the shutter is to be opened during the Integration time.
arrayPowerState	A boolean Value determining if the array will be activated/powered-up during the exposure
intraPixelDelay	A floating point number giving the amount of time to allow for settling while reading each pixel
idleProcess	An Integer tag describing how the array will be run during any Idle time in the observing run
Data Disposition -struct disposition - procedure name Arguments - filename, directory image format, data type, data stream/queue	
Pre-Processing Algorithm	
Unscrambling Algorithm	
Image Data Set ID	
coAdds - integer	
fSamples -integer	
binning - integer	
intTimeSecs - float	
digAvgs - integer -	
numPics - integer	
shutterState	
arrayPowerState	
Data Disposition	
Pre-Processing Algorithm	
Unscrambling Algorithm	
Image Data Set ID	
coAdds - integer	
fSamples -integer	
TriggerSouce -	
TriggerTimeOut	
Image Data Set ID	
None	
intTimeSecs - float	
intTimeSecs - float	
current Shutter State	
Row or Y shift	
Column or X shift	
Units to Simulate	
Units to Test	
System Power State	
System Reset Level	

Appendix II. Data stream interface questions

1. Will we need more than 255 message types? *No, this should be enough.*
2. Is one byte sufficient for version information? *Yes, there is a suggestion that no version information be included. We'll include it altough it's not needed.*
3. Should messages be self describing or depend on defined structure? *Defined structure*
4. Is the IP address sufficient for the message source ID? *This is adequate.*
5. Is a 4 Byte length too much? Would 2 bytes be enough? *2 Bytes is not quite enough 4 bytes is probably too large but is a convenient size. Pixels will be sent as a series of 4 byte integers. With a maximum message length in bytes of 4 x (NumPixels) + Preamble size or 4 x (65536) + Preamble. Assuming 65536 pixels per message this will require 256 messages for a 4k x 4k image.*
6. Should messages be grouped by time or by ID tag? *Both! we will have a start association message and each message will include the association ID*
7. Should image ID's be strings or binary? *Strings*
8. Should Association ID's be strings or binary? *Strings*
9. How should pass through data be handled? *As Header Blocks to be interpreted or not as required*
10. What formats should be allowed for output pixels? *8-bit signed or unsigned bytes, 16-bit signed or unsigned integers, 32-bit signed or unsigned Integers, 32-bit IEEE Floating point numbers,*
11. How should pixel ordering be described? *A structure with fixed format*
12. How should pixel type be described? *with text tags: byte, ubyte, short, ushort, int, uint, float*
13. Should the message stream include internal status messages? (i.e. housekeeping channels, state info, etc. *Yes*
14. What Event Classes have already been described?
15. What Events have already been described?
16. Can keywords be grouped into single messages?
17. Do we need a table data format? *Yes*
18. Where should unscrambling take place?Should data be presented in order always? *No may or may not be assembled into an image*

19. Which will be more important efficiency or human readability? *Efficiency for pixels, Human readability for everything else*