The Johns Hopkins University
Applied Physics Laboratory

# TIMED Telemetry Server

## User's Guide

Revision: Version 2.0 – July 14, 2000
Author: Kenneth Heeres

Revision: Version 3.0 – April 18, 2001
Revised by: Paul Lafferty
Includes updates taken from CONTOUR Telemetry Server, User Guide, Version 1.0 - November 9, 2000, by Leeha Herrera

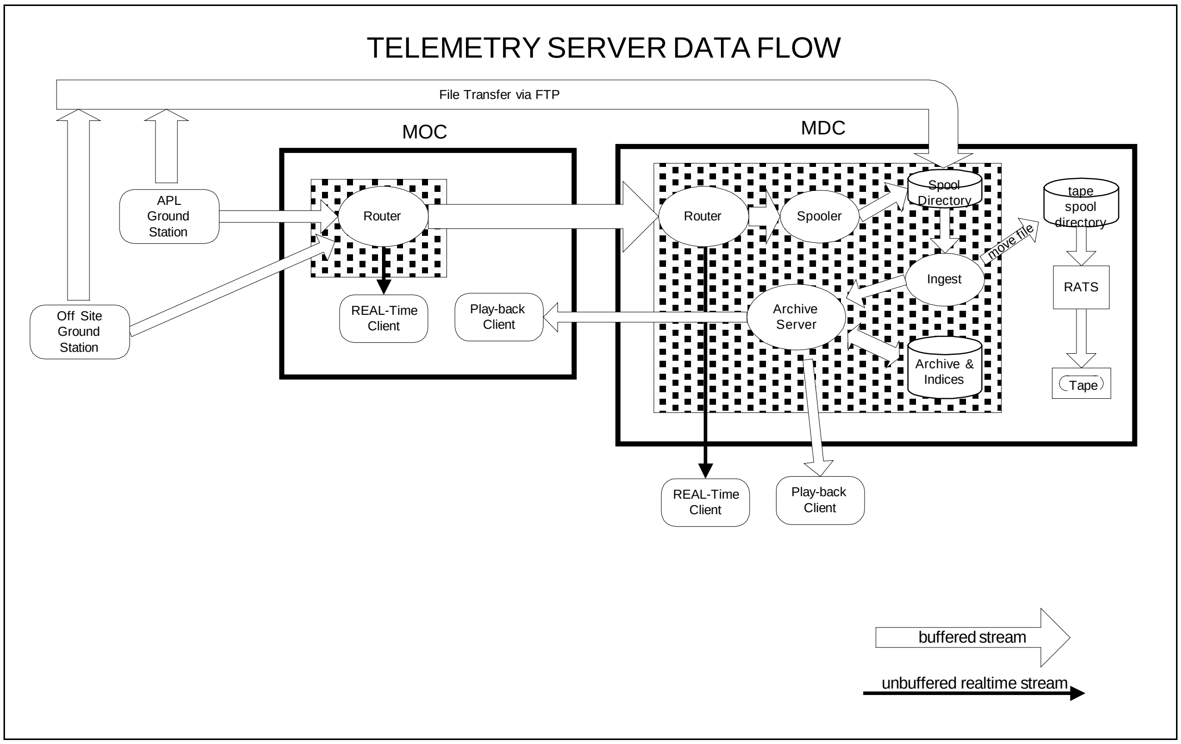
Table of Contents

## Overview

The intended audiences for this guide are end users who want to use the services to obtain telemetry, operators who have to operate the services, and people that are curious about how it works.

It should be sufficient for end users to read this overview and scan the appendices for relevant information.

The system described here is called the Telemetry Server. Its basic function is to provide a TCP/IP socket stream of telemetry data from some source to some client. This stream may be either real-time or playback. The figure illustrates the data flow.




The Telemetry Server consists of several applications:

Router – the router's purpose is to move real-time telemetry from one system to another, i.e. route, and to serve clients real-time telemetry. Sources of telemetry attempt to connect to the router to deliver data and users of data attempt to connect to the router to receive data. An additional service that the router provides is that it converts the data from whatever input format a source delivers the data to a standard format called a STF (Supplemented Telemetry Frame). In most cases the data already received is in that format already.

Spooler – the spooler's purpose is to quickly get telemetry data to a disk. The Telemetry Server supports real-time and playback services. In order to support playback services it needs to index the incoming data and this can take some time. The telemetry server also does not want to lose data so it needs to assure that data it receives is safely stored on the disk. A spooler acts as a buffered client (QpassThru) to a router; data passed to a spooler needs to be written to the disk before it will attempt to read more data.

Ingest – ingest's purpose is to read a file from the spool directory, make a connection to the archive server and start sending it data. Ingest is either spawned by spooler when it starts receiving data or it is run out of a script when a file is FTP'd into the spool directory.

Archive Server – the archive server's purpose is to index the incoming STFs by ground receipt time (GRT) and optionally spacecraft time (SCT), place the data in an organized archive, and provide a playback service of telemetry data to clients. The data may be returned in either GRT or SCT order.

There are several smaller and ancillary applications but the four above are the heart of the system. There can and often are multiple routers in the system. For example to support field operations: there is a router in the field to support local real-time telemetry services and to pass the data on to a router in the Mission Operations Center (MOC). The router in the MOC provides local real-time telemetry services to the MOC and passes data on to a router in the MDC (Mission Data Center). The router in the MDC provides local real-time telemetry services to the MDC and remote real-time telemetry services to POCs (Payload Operations Centers)

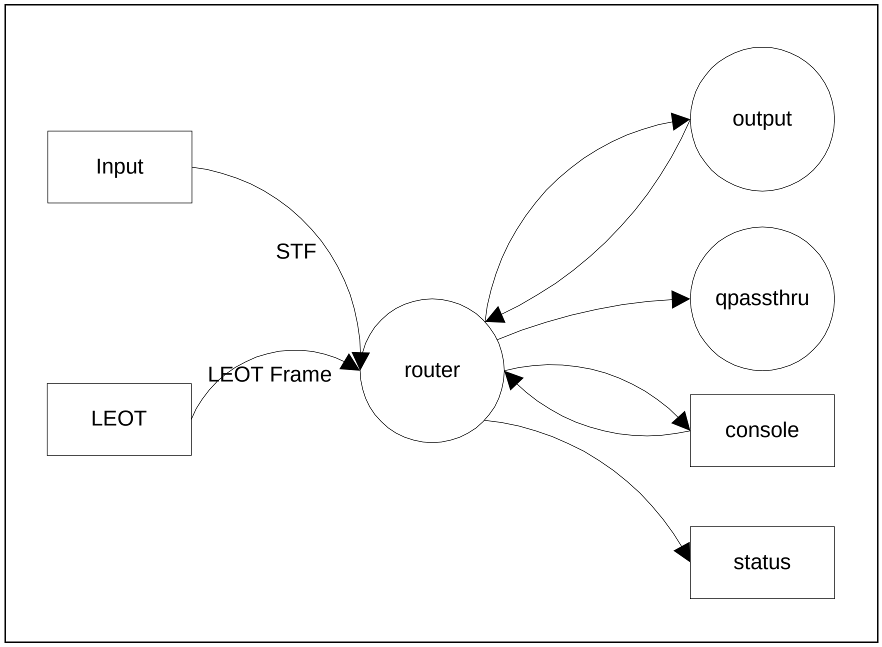
## Main Applications

The main applications of the telemetry server are the router, spooler, and the archive server. They are described below.

### Router

The discussion above described the main functions of the router. The details are provided here. The router is a separate executable. It is written in C++ and can execute in the UNIX (only tested in Solaris) environment. When the router starts, it reads a configuration file called router.ini (see Appendix D) from the local directory. The router also writes to a log file. The location of the log file is configured in the .ini file. The router uses Internet domain TCP/IP sockets to do all of its control and communication.

For TIMED the executable is stored in /tmdc/route, along with the .ini file. The router, with one exception acts as a TCP/IP socket server. It has two types of connections that it receives data from, “Input” and “LEOT”; three types of output connections, “Output”, “Console”, and “Status”; and a special case where it acts as a client to another process, “QPassthru” where it passes data on.




The router, as do all of the applications, have security and performance control features which restrict what systems can access which input or output port. Restrictions are by host name or IP address. Systems can be treated as a group and then restrictions can be applied to a group. For example: a group of machines could be called the SABERPOC and then a restriction could be placed that says there may only be at most 4 output connections to the SABERPOC. These groups can be described in the .ini file or they may be described via console commands. Security can be turned off by using the appropriate .ini or console directive.

## Data Input

The normal input to the router is a STF. A STF is a standard CCSDS Telemetry Frame with Attached Sync Marker and an added information header (see Appendix A. for frame and packet definitions) General information about CCSDS can be found in Reference xx. The information header, called a GRH (Ground Receipt Header), is described in Appendix A. There are several sources of data:

Input – The client is expected to be delivering STFs. If the router cannot handle the data it will block incoming data. There can be up to 10 "Input" clients.

LEOT – Low Earth Orbiting Terminals, these systems produce CCSDS frames with a slightly different header, and the router internally converts these to STFs. The format of the LEOT header is described in Appendix A.3 and the layout of the LEOT data is described in Appendix A.

The router acts as a socket server to sources of data. It listens on specific port numbers for incoming data. The port numbers that are listened to are configured in the .ini file. Different types of data have different port numbers. A source of data needs to know the host that is running the router software and the correct port number for its type of data in order to successfully feed data to the router. A source of data acts as socket client to the router. The protocol used is TCP.

## Data Output

The router supports several different output formats and uses several different protocols. The output formats are:

STF – normally a user of data would not use STFs. These are used for router to router communication.

STP – Supplemented Telemetry Packet, this format (described in Appendix A) contains all of the information of a STF but the data portion consists of only a CCSDS packet, rather than a CCSDS frame.

PTP – POC Telemetry Packet, the format (described in Appendix A) consists of a GRH and a CCSDS packet.

A client can specify the desired output format using directives sent to the router when the connection is first made. Those directives are described in Appendix C.

Various protocols for data output are also available:

Output – this format is provided to give each client a "fair" chance of receiving data. It uses a non-blocking TCP connection. When data are received by the router, it is written to each of the active output clients. If the client would block, i.e. it cannot read the data fast enough, the write is skipped for that client.

QPassThru – this format is provided for router to router connectivity and is meant to be a guaranteed delivery mechanism. It uses a non-blocking TCP connection but maintains a large, configurable, queue of incoming data. The buffer size is dynamic and nominally can contain 30 seconds of data before dropping data. The size, in seconds, of the queue is configurable. If data cannot be delivered to the QpassThru connection, it is dropped. The assumption is that the data will be FTP'd later.

Except in the case of QpassThru, the router acts as a TCP server to users of the data. The clients make a TCP connection to the router using a port number configured in the .ini file, write ASCII directives to the router using directives described in Appendix 0, and then begin to read the data it requested. When the client is through reading the data, it closes the socket. The current maximum number of real-time clients is 25.

## Control/Status

There are two methods of controlling the router: configuration parameters in the .ini file that are read at startup and directives sent to the router's console port. A client can establish a TCP connection to the console port and enter ASCII directives to the router. A convenient way of using this feature is to establish a Telnet connection using the console port number. The port number and the host that can use this connection are configured in the .ini file. There is only one console port.

### Configuration File (.ini)

The configuration file is called router.ini and is normally in the directory where the router was started but can be changed on the command line. *(If the designer/implementor of this had been a UNIX weenie rather than a PC weenie the file would have been called*

*.routerrc).* The parameters that can appear in the configuration files are listed in Appendix D.1.

### Console Directives

The console directives, with the exception of start and stop, are essentially the same as what are in the .ini file. Console directives can be found in Appendix E.

## Status

Status can be found in two ways. The first is to connect to the status port. When you do that router writes it status out the port and disconnects. The second is to enter a status command over a console connection.

## Logging

There are several different logging levels that can be specified:

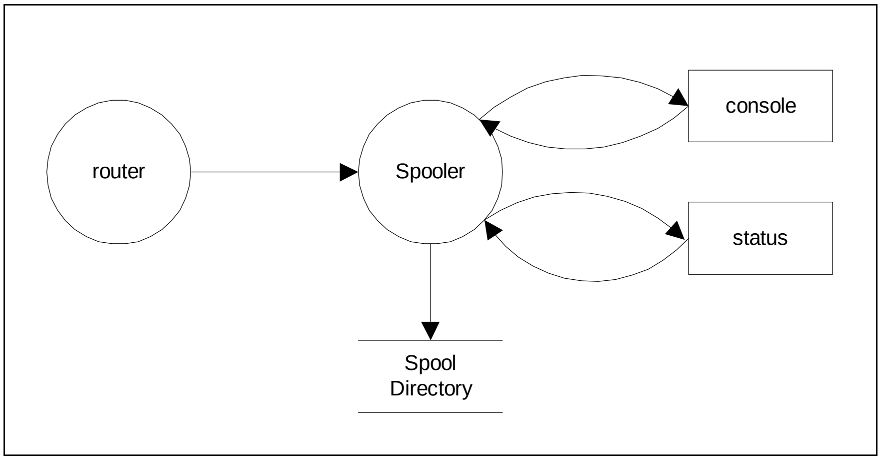
1 debug level 1
2 debug level 2
3 debug level 3
4 debug level 4
5 debug level 5
6 information messages
7 error messages
8 fatal messages that cause router termination

The lower number logging levels give more detail and include the upper numbered levels. That is, if you specify log level = 4, you will see messages from debug level 4, debug level 5, etc.

## Spooler

The spooler receives telemetry data in the form of STFs from a router using a Qpassthru connection. Its job is to place the STFs in a temporary ingest file and spawn an ingest process to send the data in the file to the archive server. The temporary ingest files are named using the following naming convention: TS + date + month + year + "_" + hour + minute.arv. For example "./TS02281994_1725.arv". Each file nominally contains the data received in a one hour period (this time period can be set in the .ini file).

The directory that the spooler uses is configured in the .ini file but is normally /project/timed/mdc/ops/spool. Processes that send telemetry data using FTP also place their data in this spool directory.




## Control/Status/Logging

The same methods for controlling the router are used to control the spooler. The configuration file options, console commands and status listings are in the Appendices. Logging is done as in the router.

## Ingest

The ingest application reads files that are being written by the spooler application to the spool directory and files that have been FTP'd to the spool directory. An instance of the ingest process is started for each file to process. For the case of spooler-generated files, the spooler application spawns an ingest process instance when it opens a temporary file. For the case of FTP'd files, there is an ftp monitoring application which will start ingest process instances when files have been FTP'd to the spool directory. The monitor program is documented in Appendix H.

One ingest process per ingest file reads the data from that temporary file and writes it to an archive server socket connection. After sending the file's contents to the archive server, the ingest process moves the file to the tape_spool directory and the RATS process saves those files to tape.

## Archive Server

The archive server is the most complicated application of the Telemetry Server. It reads data provided by ingest processes via socket connections in the form of STF's, indexes the files by GRT and optionally SCT, creates daily archive and index files, and serves clients requesting playback of data. By default, archiving by SCT time is enabled, and archiving of duplicate STFs is disabled. These defaults can be overridden in the .ini file. Note that only STFs originated by the spacecraft (with Source = = Spacecraft) can be indexed by spacecraft time. The spacecraft source code is specified in ArchiveAccessor.h.

The most natural question to ask is why is playback service combined with archive creation and indexing task. The main reason, I believe, is latency. Playback requires that the system read the index files which are continuously open and being written to by the indexing task. It was determined that the proper performance of the playback function required that the indexing task lock the index file while writing. If they were separate applications this would effectively preclude playback while data is arriving.

The archive server writes STFs to a *.tlm file, where the file is named for the year and day of year. The server creates a new .tlm file each day, and the file can be no larger than 4 Gbytes. This limit is due to the use of 32 bit file byte offset values in the index records. STFs are written to the file in the order they are received.

The archive server is capable of indexing by Ground Receipt Time (GRT) or Spacecraft Time (SCT). It produces one index file of a given type for each day of data. The GRT file has a *.gri extension and the SCT file has a *.sci extension.

The GRT index comprises a doubly-linked list of GRT index objects. The index objects contain pointers to the previous and next index objects, byte offset in file of first byte of STF, source ID (e.g., spacecraft), Front End ID, Virtual Channel, and GRT. The first record in the GRT index file is an index object whose only populated members are the previous and next indices. These are set to indicate the first and last index records in the file.

The following discussion assumes indexing by GRT. As STFs arrive, the server compares the GRT associated with the STF with the time from the previous STF. If the STF has arrived out of time order, the server does a linear search from its current index value, first backward in time, then forward, to find where the index should logically lie. Although new index records (one is created for each STF) are *always* appended to the existing list, their logical position is determined by the use of the previous and next pointers. [ Haven't yet figured out what happens if STFs arrive from previous days or for days in the future ]

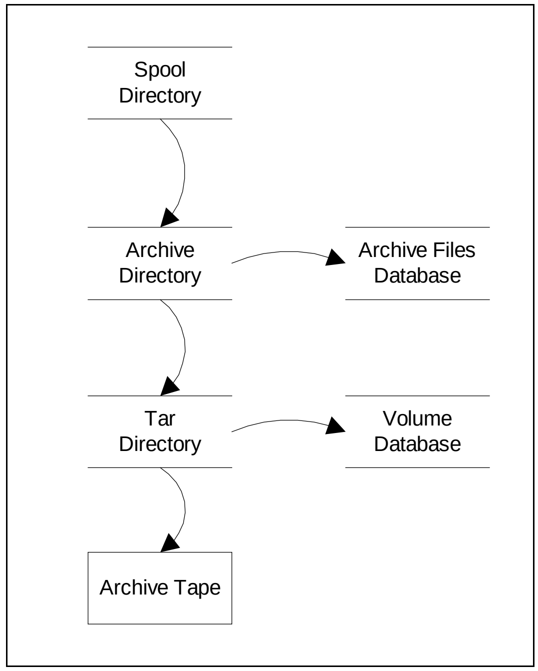
On playback, the user has the option of specifying filtering options over and above what is possible using the indexes alone. For example, the user can specify he wants STPs for a specific APID. The server reads all STFs containing that APID, but only forms STPs from the packets matching the APID.

The spacecraft time index system uses a two-level index. One index is similar to the GR index, containing previous and next pointers, byte offset in file of first byte of STF, spacecraft time, sequence number (i.e., CCSDS packet sequence number for packets with that APID), APID, GRT, packet number (i.e., number of packet within STF), and frame quality flag. Indices are maintained in logical time order, where time order is determined by the combination of spacecraft time, APID, and sequence number.

The first record in the SCT index file is a secondary index, identifying the first packet for each hour of the day. This is followed by a header Packet Time Index Data record that indicates the index of the earliest (in the next index field) and latest (in the previous index field) data in the file. The primary Packet Time Index Data records follow this in the file.

## Ancillary Applications

### RATS (Raw Archive Tape Spooler)




TSM (Telemetry Services Module)

## References

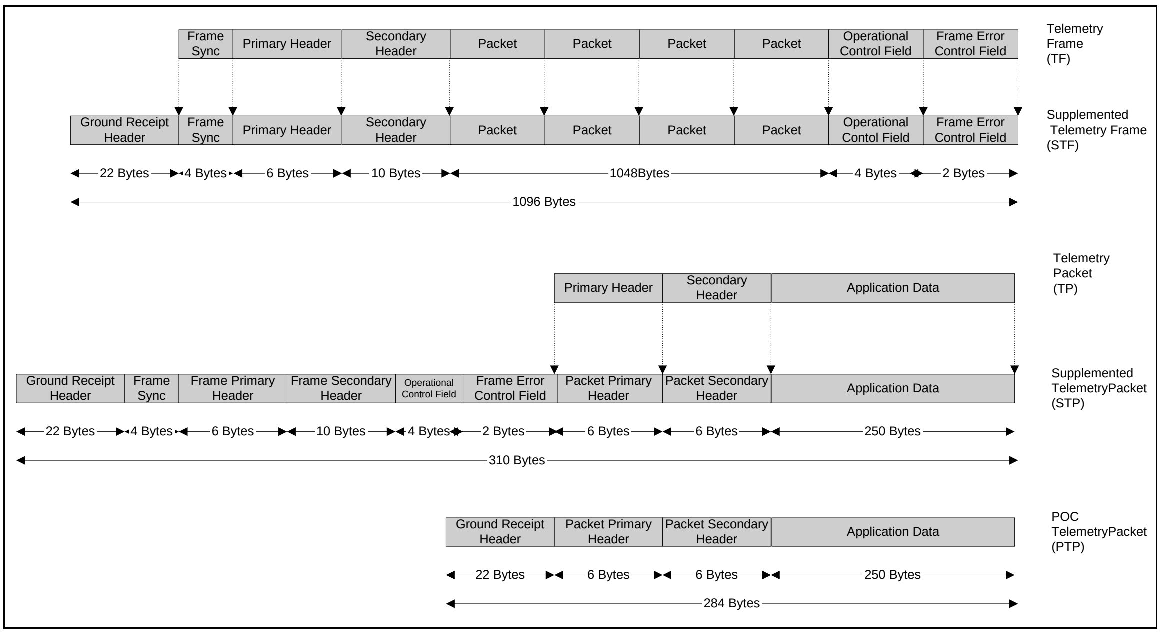
## Glossary

| | |
|---|---|
| CCSDS | Consultative Committee on Space Data Systems |
| FTP | File Transfer Protocol |
| GRH | Ground Receipt Header |
| GRT | Ground Receipt Time |
| GSFC | Goddard Space Flight Center |
| I&T | Integration and Test |
| IP | Internet Protocol |
| JHU/APL | Johns Hopkins University Applied Physics Laboratory |
| LEOT | Low Earth Orbit Terminal |
| MDC | Mission Data Center |
| MOC | Mission Operations Center |
| POC | Payload Operations Center |
| PTP | POC Telemetry Packet |
| RATS | Raw Archive Tape Spooler |
| TIMED | Thermosphere, Ionosphere, Mesosphere, Energetics, Dynamics |
| SCT | Spacecraft Time |
| STF | Supplemented Telemetry Frame |
| STP | Supplemented Telemetry Packet |
| TCP | Transmission Control Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| VAFB | Vandenberg Air Force Base |

# Appendices

## Appendix A. TIMED Telemetry Layouts

### A.1. Frame & Packet Layouts




### A.2. Ground Receipt Header

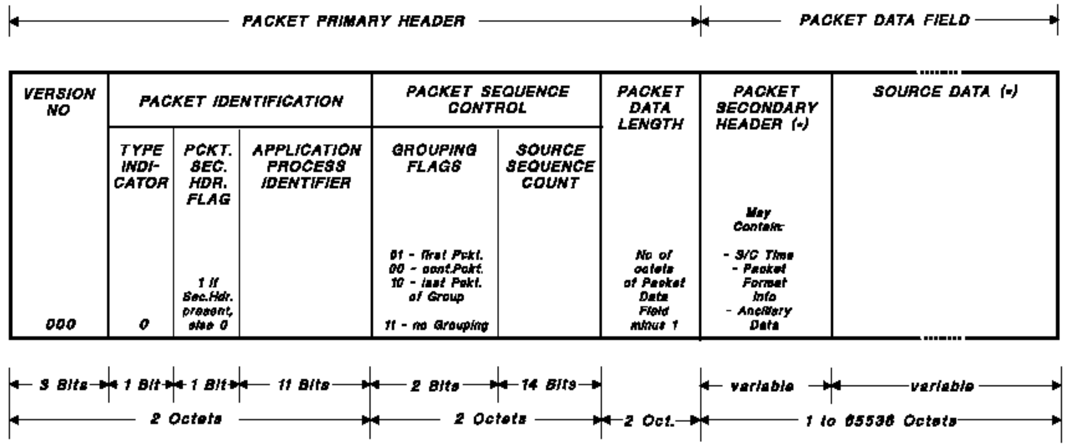
| Field | Offset (b) | Length (b) | Description |
|---|---|---|---|
| Size | 0 | 16 | Size of this object including headers in bytes, unsigned integer in MSB first order (max = 65535) |
| data type | 16 | 8 | type of data object, 1 = STF, 2 = STP, *3 = PTP* |
| *spare bits* | *24* | *8* | *undefined* |
| *GRH Version ID* | *32* | *6* | *version id associated with this GRH format. (Decimal number, where this version = 2)* |
| spacecraft ID | 38 | *10* | CCSDS SCID assigned to TIMED |
| GR Time | 48 | *32* | Ground receipt time in elapsed seconds since 00:00:00 UTC January 6, 1980, in MSB first order |
| GR Time Vernier | 80 | *32* | Microsecond offset from GR Time, in MSB first order |
| Frame Source Type | 112 | *4* | 0001 - Emulator/Mini-MOC<br>0010 - Simulator<br>0011 - Loop-Back<br>0100 - spacecraft |

| Field | Offset (b) | Length (b) | Description |
|---|---|---|---|
| | | | 0101 - GSE<br>0110 - unused<br>0111 - unused<br>1000 - User-Defined<br>1001 - 1111 - unused |
| Frame Source Index | 116 | *4* | for frame source type 0001 (Emulator/Mini-MOC)<br>0001 GUVI Spacecraft Emulator<br>0010 SABER Spacecraft Emulator<br>0011 SEE Spacecraft Emulator<br>0100 TIDI Spacecraft Emulator<br>0101 GNS Mini-MOC 1<br>0110 GNS Mini-MOC 2<br>0111 G&C Mini-MOC 1<br>1000 G&C Mini-MOC 2<br>1001 IEM Mini-MOC 1<br>1010 IEM Mini-MOC 2<br>for frame source type 0010 (Simulator)<br>0001 TOPS<br>0010 Software Simulation<br>for frame source type 0011 (Loop-Back)<br>0001 FE Hardware Simulation<br>for frame source type 0100 (Spacecraft)<br>0001 Spacecraft<br>for frame source type 0101 (GSE)<br>0001 GSE<br>0010 MPCF sc1_rt instance<br>0011 MPCF sc2_rt instance<br>0100 MPCF dev instance<br>0101 MPCF tops instance<br>0110 MPCF iem_mm1_rt instance<br>0111 MPCF iem_mm2_rt instance<br>(where MPCF=MOC/POC Command Filter: sc1_rt, sc2_rt, dev, tops, iem_mm1_rt, and iem_mm2_rt = EPOCH stream names)<br>for frame source type 1000 (User-Defined)<br>0000-1111 - User-Defined |
| Path | 120 | *4* | other path information (0000 for now) |
| Front-end Identifier | 124 | *4* | 0001 - FE1 (*bench-testing*) |
| | | | 0010 - FE2 (I&T)<br>0011 - FE3 (*primary ground station*)<br>0100 - FE4 (spare)<br>0101 - G&C<br>0110 - GPS<br>0111 - MOC<br>1000-1101 - LEO-T or other off-site<br>1110-1111 - unused<br>*(Note that front end assignments will be finalized later.)* |
| R-S decode Flag | 128 | *1* | 0 = disabled |

| Field | Offset (b) | Length (b) | Description |
|---|---|---|---|
| | | | 1 = enabled |
| R-S error status | 129 | *1* | 0 = frame uncorrectable<br>1 = frame correct or corrected |
| R-S error count | 130 | *7* | 0 = no error needed correction<br>1..80 count of corrected errors81..127 unused |
| CRC Flag | 137 | *1* | 0 = CRC disabled<br>1 = CRC enabled |
| CRC Error Flag | 138 | *1* | 0 = CRC failed<br>1 = CRC passed |
| Master Channel Sequence checked | 139 | *1* | 0 = not checked/unknown<br><br>1 = sequence number checked |
| Master Channel Sequence Number Error | 140 | *1* | 0 = sequence number increased by one<br><br>1 = sequence number increased by two or more |
| Frame Sync Mode | 141 | *2* | 00 = search<br>01 = check<br>10 = lock<br>11 = flywheel |
| Frame Quality Flag | 143 | *1* | 0 = data is suspect<br>1= data is correct<br>*used to determine if the frame quality is acceptable for output to client who requests only "good" data; No Frame Error Detected = No RS Error & No CRC Error & No SSR Playback Error* |
| Frame Sync Pattern Errors | 144 | *4* | number of errors detected in Frame Sync pattern |
| Frame Sync bit slips | 148 | *4* | 0000 = no slip<br>1001 = 1 bit late<br>1010 = 2 bits late<br>1011 = 3 bits late<br>1101 = 1 bit early<br>1110 = 2 bits early<br>1111 = 3 bits early |
| Archive Flag | 152 | *1* | 0 = do not archive this data<br>1 = archive this data |
| SSR Playback Error | 153 | *1* | 0 = No error<br>1 = SSR playback error |
| spares | 154 | *22* | *undefined* |

## A.3. LEOT Header

## A.4. CCSDS Packet Header Format




(*) may or may not be required; for details see specifications in text.

## A.5. TIMED CCSDS Packet Primary Header

| Field | Offset(b) | Length(b) | Description |
|---|---|---|---|
| Version Number | 0 | 3 | Packet version number |
| Type | 3 | 1 | 0 designates a telemetry packet |
| SH Flag | 4 | 1 | Secondary header flag |
| Subsystem ID | 5 | 4 | Subsystem identifier<br>0000 C&DH 1<br>0001 C&DH 2<br>0010 AIU 1<br>0011 AIU 2<br>0100 FC 1<br>0101 FC 2<br>0110 GNS 1<br>0111 GNS 2<br>1001 GUVI<br>1010 TIDI<br>1011 SABER<br>1100 SEE |
| Format | 9 | 7 | Data format identifier |
| G Flag | 16 | 2 | Grouping Flag: 0-first packet, 1-intermediate packet, 2-last packet, 4-not part of a group |
| Source Sequence Count | 18 | 14 | Continuous sequence count |
| | 32 | 16 | Number of bytes in the packet data field -1, this includes the secondary header. For TIMED the number is 255 |
| Total | 48 | | |

### A.6. TIMED CCSDS Packet Secondary Header

| Field | Offset(b) | Length(b) | Description |
|---|---|---|---|
| S/C Time | 0 | 32 | GPS unsegmented time in seconds |
| Optional time | 32 | 16 | |
| Total | 48 | | |

## Appendix B. TIMED Application ID's

| Subsystem | Number of Identifiers (Decimal) | Range of Identifiers | |
|---|---|---|---|
| | | (Decimal) | (Hex 11-bit integer) |
| C & DH #1[1] | 128 | 0 thru $2^7$ - 1 | 000 thru 07F |
| C & DH #2 | 128 | $2^7$ thru 2 x $2^7$ - 1 | 080 thru 0FF |
| AIU #1 | 128 | 2 x $2^7$ thru 3 x $2^7$ - 1 | 100 thru 17F |
| AIU #2 | 128 | 3 x $2^7$ thru 4 x $2^7$ - 1 | 180 thru 1FF |
| FC #1 | 128 | 4 x $2^7$ thru 5 x $2^7$ - 1 | 200 thru 27F |
| FC #2 | 128 | 5 x $2^7$ thru 6 x $2^7$ - 1 | 280 thru 2FF |
| GNS #1 | 128 | 6 x $2^7$ thru 7 x $2^7$ - 1 | 300 thru 37F |
| GNS #2 | 128 | 7 x $2^7$ thru 8 x $2^7$ - 1 | 380 thru 3FF |
| Ground System[1] | 128 | 8 x $2^7$ thru 9 x $2^7$ - 1 | 400 thru 47F |
| GUVI | 128 | 9 x $2^7$ thru 10 x $2^7$ - 1 | 480 thru 4FF |
| TIDI | 128 | 10 x $2^7$ thru 11 x $2^7$ - 1 | 500 thru 57F |
| SABER | 128 | 11 x $2^7$ thru 12 x $2^7$ - 1 | 580 thru 5FF |
| SEE | 128 | 12 x $2^7$ thru 13 x $2^7$ - 1 | 600 thru 67F |
| Spare | 128 | 13 x $2^7$ thru 14 x $2^7$ - 1 | 680 thru 6FF |
| Spare | 128 | 14 x $2^7$ thru 15 x $2^7$ - 1 | 700 thru 77F |
| Spare | 128 | 15 x $2^7$ thru 16 x $2^7$ - 2 | 780 thru 7FE |
| Idle Packets[2] | 1 | 16 x $2^7$ - 1 | 7FF |

## Appendix C. Client Directives

The directives listed here are written to the server over the socket after the connection is made. The directives are accepted until the "BEGN" directive is received and then the selected data begins to flow out the socket. The server will no longer attempt to read the socket. This interface is assumed to be under control of a program and thus no time is spent doing fancy parsing.

[1] as received from, or sent to, controlled equipment in the ground system
[2] as contained in VC0 Telemetry Frames

## C.1. Real-time Client Directives

| Directive | Parameters | Description | Defaults |
|---|---|---|---|
| APID | *Number in oct,hex or decimal* | Application Process ID from Packet Primary Header. You can request multiple APIDs, one per directive. You must specify at least one APID or SSYS to receive TP,STP,or PTP. For all APIDs use SSYS=ALL. | (none) |
| BEGN | RT | Start to Send Data | n/a |
| DRTY | (none) | Include data that has been marked as bad in the ground receipt header. Normally this data is not passed on. | n/a |
| EXAPID | *Number in oct,hex or decimal* | Exclude APID from stream. You can request multiple APIDs for exclusion. | (none) |
| FRNT | *decimal number*, "ALL","BEST" | Front-end id from Ground Receipt Header. You can request multiple FRNTs, one per directive. You can get all of the front ends by using the keyword ALL. The ALL option will send duplicate streams for a source if there are multiple input streams from the same source. When BEST is specified the server will automatically switch which Front End the data comes from in an attempt to supply a continuous stream from a particular source. | (none) |
| SRCE | *decimal number, "ALL"* | Frame Source ID – Frame Source Type and Frame Source Index from the Ground Receipt Header. You can request multiple sources, one per directive. You can get all of the sources by specifying ALL. | (none) |
| SSYS | *decimal number*, "ALL" | Requests all APIDs that match the subsystem ID (4 most significant bits of the APID field in the Packet Primary Header). You can request multiple subsystems, one per directive . You must specify at least one APID or SSYS to receive TP,STP or PTP. SSYS=ALL will supply all APIDs. | (none) |
| TYPE | "TP", "STP", "TF", "STF", "PTP" | Specify whether to get Telemetry Packets, Supplemented Telemetry Packets, POC Telemetry Packets, Transfer Frames, or Supplemented Transfer Frames. Only one type may be specified. | (none) |
| VCHN | "0","6","7", "ALL" | Virtual channel ID from the Transfer Frame Primary Header. You can request multiple VCHNs, one per directive. | (none) |
| TLM_PORT | *decimal number* | Port number for remote connection of second socket (required if second socket requested) – This was done for SEE instrument OASYS users, whose protocol prevents it from receiving data on the requesting socket. | (none) |
| TLM_HOST | *decimal number ddd.ddd.ddd.ddd* | Host IP number for remote connection of second socket - if not the same as first socket (host names not allowed) | same as IP of first socket connection |

## C.2. Playback Client Directives

| Directive | Parameters | Description | Defaults |
|---|---|---|---|
| APID | *Number in oct,hex or decimal* | Application Process ID from Packet Primary Header. You can request multiple APIDs, one per directive. You must specify at least one APID or SSYS to receive TP,STP,or PTP. For all APIDs use SSYS=ALL. | (none) |
| BEGN | PB | Start to send data | n/a |

| Directive | Parameters | Description | Defaults |
|---|---|---|---|
| DRTY | (none)<br>or<br>"ONLY" | If directive not given them only good data is sent.<br>If directive specified without parameter then stream will include both good and bad data. If DRTY=ONLY then only data marked as bad will be sent. Quality is defined by the Frame Quality Flag in the ground receipt header. | n/a |
| EXAPID | *Number in oct,hex or decimal* | Exclude APID from stream. You can request multiple APIDs for exclusion. | (none) |
| FRNT | *decimal number,* "ALL" | Front-end id from Ground Receipt Header. You can request multiple FRNTs, one per directive. Only meaningful for ORDR=GR. You can get all of the front ends by using the keyword ALL. The ALL option will send duplicate streams for a source if there are multiple input streams from the same source. Ignored if ORDR=SC. | (none) |
| NOWAIT | *(none)* | Do not wait at end of archive data even if stop time not reached. | Off (see STOP) |
| ORDR | *"SC","GR"* | Spacecraft time or Ground Receipt time order. Prior to launch only ground receipt time ordering will be available. | GR |
| SRCE | *decimal number, "ALL"* | Frame Source ID – Frame Source Type and Frame Source Index from the Ground Receipt Header. You can request multiple SRCEs, one per directive. You can get all of the sources by specifying ALL. | (none) |
| SSYS | *decimal number,* "ALL" | Subsystem ID (4 most significant bits of the APID field in the Packet Primary Header). You can request multiple SSYSs, one per directive. You must specify at least one APID or SSYS to receive TP,STP or PTP. SSYS=ALL will supply all APIDs. | (none) |
| STRT | *yyyy ddd hh:mm:ss* | start time – must be before time of last data in archive | start of current utc day |
| STOP | *yyyy ddd hh:mm:ss* | end time - if end time exceeds the time of the last data in the archive the server will wait for new data to arrive | time of last data in archive |
| TYPE | "TP", "STP", "TF", "STF", "PTP" | Specify whether to get Telemetry Packets, Supplemented Telemetry Packets, POC Telemetry Packets, Transfer Frames, or Supplemented Transfer Frames. Only one type may be specified. | (none) |
| VCHN | "0","6","7", "ALL" | Virtual channel ID from the Transfer Frame Primary Header. You can request multiple VCHNs, one per directive. Ignored if ORDR=SC. | (none) |
| TLM_PORT | *decimal number* | Port number for remote connection of second socket (required if second socket requested) – This was done for SEE instrument OASYS users, whose protocol prevents it from receiving data on the requesting socket. | (none) |
| TLM_HOST | *decimal number ddd.ddd.ddd.ddd* | Host IP number for remote connection of second socket - if not the same as first socket (host names not allowed) | same as IP of first socket connection |

## Appendix D. Configuration Files

Each of the applications has a file that is read at start up and it is used to configure the file. The default name of the file is "application.ini" and its default location is the directory from which the application is started. . If the line in the configuration file begins with a "#" or does not have an "=" sign it is ignored. The code generally looks for the key word and then does an sscanf to get the value. If a line does not contain a key word, it is ignored.

Each application will take a command line argument which is the full path name of an alternate start up file.

### D.1. Router Configuration File

| Keyword | Description |
|---|---|
| CONSOLE_LISTENER_IP | Valid IP address of a console front-end machine. |
| CONSOLE_PORT | The Router's assigned port number for the console listener socket. |
| INPUT_LISTENER_IP | Valid IP address of an Input front-end machine serving supplemented telemetry frames. For all addresses you can use either the IP address or the full domain address. |
| INPUT_PORT | The Router's assigned port number for the Input front-end listener socket that serve supplemented telemetry frames. |
| LEAP_SECONDS | Leap seconds added to the ground receipt time in the conversion from LEO-T messages to supplemented telemetry frames. |
| LEOT_LISTENER_IP | Valid IP address of a LEO-T front-end machine. |
| LEOT_PORT | The Router's assigned port number for the LEO-T's front-end listener socket. |
| LISTENER_BYPASS | Specifies whether the router will validate the IP address of each client connection. |
| LOG_ALIVE_MESSAGE_DELTA | The time in seconds between each logger log alive message. |
| LOG_FILE_PATH | The absolute or relative pathname of the log file, including the ending slash. |
| LOG_FILE_PREFIX | The prefix to the log file name. |
| LOG_LEVEL | Router logging level. Identifies the level of displayed and log messages. The user is allowed to changed the log level for debug messages only. INFO, ERROR, and FATAL messages will be logged always.<br>1 – debug message (many messages)<br>2 – debug message<br>3 – debug message<br>4 – debug message<br>5 – debug message<br>6 – INFOrmational message<br>7 – ERROR message<br>8 – FATAL message |
| LOG_TO_FILE | Specifies whether the Router will log to file at start up. ("YES" or "NO") |
| OUTPUT_LISTENER_IP | Valid IP address of an Output front-end machine. |
| OUTPUT_PORT | The Router's assigned port number for the output real-time client listener socket. |
| PASS_THRU_CONNECTION | Specifies whether the Router attempts to connect to the PassThru client at startup. ("YES" or "NO") |
| PASS_THRU_IP | The IP address of the PassThru client. |

| Keyword | Description |
|---|---|
| PASS_THRU_PORT | The Router's assigned port number for the pass thru client socket. |
| PASS_THRU_QUE_TIME | The maximum time that a STF can be in the passthru queue in seconds. |
| PASS_THRU_RECONNECT_RETRIES | The number of reconnect retries the Router will attempt if the PassThru connection is lost. |
| PASS_THRU_RECONNECT_TIMER | The delta time used in the reconnect retry logic. (seconds) |
| SELECTOR_WAIT_TIMER | Time in seconds the selector class will wait before returning if there is no activity on any socket on the socket list. |
| SPACECRAFT_ID | Spacecraft ID for the TIMED Mission, set in the ground receipt header during the conversion of LEO-T messages to supplemented telemetry frames. Uses "%d" to sscanf for the spacecraft ID. This is not actually checked against the incoming data. |
| STATUS_LISTENER_IP | Valid IP address of an Status front-end machine. |
| STATUS_PORT | The Router's assigned port number for the status listener socket. |
| VERSION | Software version this configuration file is compatible with. If this software version does not match the application software version, the Router will not execute with this configuation file. Uses "%f" to sscanf for the version number. |

# Appendix E. Console Directives

Directives can be entered at the console for most of the applications. These directives are used to control the console and to override the settings from the start up file.

These are the directives that can be entered from a console session. If the directive is not one of the one listed, an error message is logged and reported back to the console. Each directive line is read in and only the `isalnum()` characters ({0-9,'a'-'z','A'-'Z'}), ':', '-', and '.' are passed on.

## E.1. Router Console Directives

| **Console Directive** | **Description** |
|---|---|
| HELP | Display console directives |
| PAUSE | Pause all input and output telemetry message processing |
| RESTART | Restarts all input and output telemetry message processing |
| LOGLEVEL # | Set new log level to #. Does an `sscanf` with "%d" to get the new log level. Log level can be set to a value between 1 and 6. |
| LOGSTOP | Stop all logging to log file. |
| LOGSTART | Open new log file and begin logging to log file |
| CLOSESOCKET # | Close the specified socket descriptor connection. Does a `sscanf` with "%d" to get the socket number. |
| CLOSECONSOLE | Close the active console connection |
| OPENPASSTHRU | Attempt to connect to QPassThru client |
| STATUS | Display status of the Router |
| STOP | Kills the router! |
| ADD GROUP IP name:ip | Add and IP address to a group. Checks for a ':', `strncpy`'s the letters before the ':', and does an `sscanf` with "%s" to get the ip address. |
| MOD GROUP CONN name:# | Modify the number of connections a group can have. Checks for a ':', `strncpy`'s the letters before the ':', and does an `sscanf` with "%d" to get |

| Console Directive | Description |
|---|---|
| | the #. |
| REMOVE GROUP IP name:ip | Remove an IP address from a group. Checks for a ':', strncpy's the letters before the ':', and does an sscanf with "%s" to get the ip address. |
| SET ALL BYPASS {YES,NO} | Bypass security checking on all sockets. Does a strncmp on "YES" or "NO", if neither |
| SET INPUT BYPASS | Bypass security on input sockets |
| SET LEOT BYPASS | Bypass security on LEOT sockets |
| SET STATUS BYPASS | Bypass security on status sockets |
| SET CONSOLE BYPASS | Bypass security on console sockets |
| SET OUTPUT BYPASS | Bypass security on output sockets |
| RESET | Reset counters for dropped packest on QpassThru |
| TIMECHECK {ON,OFF} | Turn on and off time checking for bad data |
| PASSTHRUCONNTIME seconds | Time between reconnection tries |
| PASSTHRUQUETIME seconds | The length of time that data can be held in the passthru queue before they are dropped. |
| | |

### E.2. Spooler Console Directives

| Console Directive | Description |
|---|---|
| HELP | Display console directives |
| PAUSE | Pause all input and output telemetry message processing |
| RESTART | Restarts all input and output telemetry message processing |
| LOGLEVEL # | Set new log level to #. Does an sscanf with "%d" to get the new log level. Log level can be set to a value between 1 and 6. |
| LOGSTOP | Stop all logging to log file. |
| LOGSTART | Open new log file and begin logging to log file |
| CLOSESOCKET # | Close the specified socket descriptor connection. Does a sscanf with "%d" to get the socket number. |
| CLOSECONSOLE | Close the active console connection |
| CLOSESPOOL | Close the current spool file |
| STATUS | Display status of the spooler |
| STOP | Kills the spooler |

### E.3. Archive Server Console Directives

| Console Directive | Description |
|---|---|
| HELP | Display console directives |
| PAUSE | Pause all input and output telemetry message processing |
| RESTART | Restarts all input and output telemetry message processing |
| LOGLEVEL # | Set new log level to #. Does an sscanf with "%d" to get the new log level. Log level can be set to a value between 1 and 6. |
| LOGSTOP | Stop all logging to log file. |
| LOGSTART | Open new log file and begin logging to log file |
| CLOSESOCKET # | Close the specified socket descriptor connection. Does a sscanf with "%d" to get the socket number. |
| CLOSECONSOLE | Close the active console connection |
| STATUS | Display status of the Router |
| STOP | Kills the router! |
| ADD GROUP IP name:ip | Add and IP address to a group. Checks for a ':', strncpy's the letters before the ':', and does an sscanf with "%s" to get the ip address. |

| Console Directive | Description |
|---|---|
| `MOD GROUP CONN name:#` | Modify the number of connections a group can have. Checks for a ':', `strncpy`'s the letters before the ':', and does an `sscanf` with "%d" to get the #. |
| `REMOVE GROUP IP name:ip` | Remove an IP address from a group. Checks for a ':', `strncpy`'s the letters before the ':', and does an `sscanf` with "%s" to get the ip address. |
| `SET ALL BYPASS {YES,NO}` | Bypass security checking on all sockets. Does a `strncmp` on "YES" or "NO", if neither |
| `SET INPUT BYPASS` | Bypass security on input sockets |
| `SET LEOT BYPASS` | Bypass security on LEOT sockets |
| `SET STATUS BYPASS` | Bypass security on status sockets |
| `SET CONSOLE BYPASS` | Bypass security on console sockets |
| `SET OUTPUT BYPASS` | Bypass security on output sockets |
| `TIMECHECK {ON,OFF}` | Turn on and off time checking for bad data |

## Appendix F. Application Status

A status of the running application can be obtained by making a socket connection to the status port or entering a "STATUS" command to an already existing console connection.

### F.1. Router Status Output

```
BEGIN_STATUS
ROUTER IS STARTED
Input Listener, Port(3100) Sock(6)
Leot Listener, Port(3101) Sock(7)
Output Listener, Port(3102) Sock(8)
Console Listener, Port(3110) Sock(9)
Status Listener, Port(3111) Sock(10)
BadTimeDisconnect is ON, DisconnectHost(), DisconnectTime(,)
PassThru, IP(tmdc-ts2.jhuapl.edu:3200) Sock(5)
LastSendTime(05/09/2000,16:11:51) DroppedMSGCount 0 QueLength 0 QueTime 30 ReconnTime 300
Past/present inputs, Total:(24), INPUT:(24), LEOT:(0)
Current inputs, Total:(5), INPUT:(5), LEOT:(0)
Input, IP(128.244.227.165:3100) Sock(11) FE(INPUT)
 Created(05/03/2000,14:22:52) 1stMsg(05/03/2000,14:22:52) LastMsg(05/09/2000,16:11:51)
Input, IP(128.244.149.29:3100) Sock(12) FE(INPUT)
 Created(05/03/2000,14:22:53) 1stMsg(05/03/2000,14:22:53) LastMsg(05/09/2000,16:11:51)
Input, IP(128.244.149.136:3100) Sock(13) FE(INPUT)
 Created(05/03/2000,14:32:44) 1stMsg(05/03/2000,14:33:05) LastMsg(05/09/2000,16:11:51)
Input, IP(128.244.47.246:3100) Sock(14) FE(INPUT)
 Created(05/03/2000,15:48:11) 1stMsg(05/03/2000,17:56:57) LastMsg(05/09/2000,16:11:51)
Input, IP(128.244.149.51:3100) Sock(16) FE(INPUT)
 Created(05/03/2000,15:49:58) 1stMsg(,) LastMsg(,)
Past/present outputs:(36)
Current outputs:(4)
Output, IP(128.244.149.29:3102) Sock(15) Msg(STP)
 Created(05/05/2000,21:06:03) 1stMsg(05/05/2000,21:06:04) LastMsg(05/09/2000,16:11:51)
 Apid: Frnt: 1 7 Srce: 33 81 85 Ssys: 0 1 2 3 4 5 6 7 8 13 14 Vchn: 7 Drty: YES Begn: YES
Output, IP(128.244.47.246:3102) Sock(17) Msg(STP)
 Created(05/08/2000,20:18:03) 1stMsg(05/08/2000,20:18:03) LastMsg(05/09/2000,16:11:51)
 Apid: 1108 Frnt: ALL Srce: 24 81 Ssys: 2 3 4 5 8 Vchn: 7 Drty: NO Begn: YES
Output, IP(128.244.47.29:3102) Sock(20) Msg(STP)
 Created(05/08/2000,18:42:45) 1stMsg(05/08/2000,19:30:40) LastMsg(05/09/2000,16:11:51)
 Apid: 4 Frnt: 1 Srce: ALL Ssys: Vchn: ALL Drty: NO Begn: YES
Output, IP(128.244.47.29:3102) Sock(21) Msg(STP)
 Created(05/08/2000,18:43:06) 1stMsg(,) LastMsg(,)
 Apid: 4 Frnt: 3 Srce: ALL Ssys: Vchn: ALL Drty: NO Begn: YES
Number of status port hits:(319771)
Output IP Group status
   IP Group (MOC): (0) connections of (20) maximum
      IP Group (MOC) IP Address (oliver.jhuapl.edu)
      IP Group (MOC) IP Address (haney.jhuapl.edu)
      IP Group (MOC) IP Address (lisa.jhuapl.edu)
```

```
      IP Group (MOC) IP Address (arnold.jhuapl.edu)
      IP Group (MOC) IP Address (ralph.jhuapl.edu)
      IP Group (MOC) IP Address (alf.jhuapl.edu)
      IP Group (MOC) IP Address (sam.jhuapl.edu)
      IP Group (MOC) IP Address (kimball.jhuapl.edu)
      IP Group (MOC) IP Address (kate.jhuapl.edu)
      IP Group (MOC) IP Address (newt.jhuapl.edu)
   IP Group (MDC): (0) connections of (10) maximum
      IP Group (MDC) IP Address (tmdc-ts2.jhuapl.edu)
      IP Group (MDC) IP Address (tmdc-dev2.jhuapl.edu)
      IP Group (MDC) IP Address (127.0.0.1)
   IP Group (MISC): (0) connections of (20) maximum
END_STATUS
```

## F.2. Spooler Status Reports

```
Spooler IS STARTED
Input Listener, Port(3200) Sock(5)
Console Listener, Port(3210) Sock(6)
Status Listener, Port(3211) Sock(7)
Past/present inputs, Total:(1)
Current inputs, Total:(1)
Input, IP(128.244.149.37:3200) Sock(8)
 Created(05/03/2000,14:22:21) 1stMsg(05/03/2000,14:22:52) LastMsg(05/09/2000,16:12:49)
Number of status port hits:(1)
Current spool file: (/d4016/tmdc/spool/TS05092000-1548.arv)
Number of records in current spool file: (29184)
Total processed spool files: (149)
Spool file close delta time: (3600)seconds
END_STATUS
```

## F.3. Archive Server Status

```
BEGIN_STATUS
ArchiveServer IS STARTED
Input Listener, Port(3300) Sock(5)
Output Listener, Port(3302) Sock(6)
Console Listener, Port(3310) Sock(7)
Status Listener, Port(3311) Sock(8)
Past/present inputs:(151)
Current inputs:(3)
Input, IP(128.244.149.37:3300) Sock(14)
 Created(05/09/2000,08:46:57) 1stMsg(05/09/2000,08:47:07) LastMsg(05/09/2000,08:48:08)
Input, IP(128.244.149.37:3300) Sock(9)
 Created(05/09/2000,08:25:10) 1stMsg(05/09/2000,08:25:20) LastMsg(05/09/2000,08:46:58)
Input, IP(128.244.149.37:3300) Sock(22)
 Created(05/09/2000,15:48:06) 1stMsg(05/09/2000,15:48:16) LastMsg(05/09/2000,16:13:43)
Past/present outputs:(5511)
Current outputs:(0)
Number of status port hits:(52219)
Output IP Group status
   IP Group (MOC): (0) connections of (20) maximum
      IP Group (MOC) IP Address (oliver.jhuapl.edu)
      IP Group (MOC) IP Address (haney.jhuapl.edu)
      IP Group (MOC) IP Address (lisa.jhuapl.edu)
      IP Group (MOC) IP Address (arnold.jhuapl.edu)
      IP Group (MOC) IP Address (ralph.jhuapl.edu)
      IP Group (MOC) IP Address (alf.jhuapl.edu)
      IP Group (MOC) IP Address (sam.jhuapl.edu)
      IP Group (MOC) IP Address (kimball.jhuapl.edu)
      IP Group (MOC) IP Address (kate.jhuapl.edu)
      IP Group (MOC) IP Address (newt.jhuapl.edu)
   IP Group (MDC): (0) connections of (5) maximum
      IP Group (MDC) IP Address (tmdc-ts2.jhuapl.edu)
      IP Group (MDC) IP Address (tmdc-dev2.jhuapl.edu)
      IP Group (MDC) IP Address (127.0.0.1)
   IP Group (MISC): (0) connections of (20) maximum
END_STATUS
```

## Appendix G. Router Internals

The following is a narrative of how the router works.
The router is basically a TCP/IP socket server. It begins by creating a number of objects whose task it is to listen on certain ports. A select statement is used to wake up the router when a client requests services. The following listeners are created:

1. InData – this is the normal input for STF data
2. Leot – this is the input port for LEOT data
3. Output – this is the output for real-time clients
4. Console – this is the port for control input and output
5. Status – this is the port for getting a status out of the router applicaiton

## Appendix H. FTP Monitoring

The FTP monitoring task replaces the normal execute of ftpd with a program that executes ftpd and then a script which starts the ingest application. In implementation it is a little more complicated then that.

Inetd is a normal UNIX daemon which listens on a multiple number of ports and when a client attempts to connect to a port it spawns an application to handle the request. The configuration of inetd is controlled by a inetd.conf file. Under the Solaris the inetd.conf file is in /etc. The directive for ftp normally looks like this:

```
ftp     stream  tcp     nowait  root    /usr/sbin/in.ftpd
```

The syntax for this line is:

```
service-name  endpoint-type  protocol  wait-status  uid  server-
program server-arguments
```

Everything after the service name is treated as arguments to the service. On all SRS administrated programs we use an alternate FTP daemon from the University of Washingon call wuftp and we use a TCP wrapper program for the services which allows us to log and to restrict port access. Thus, our normal inet.conf looks like this:

```
ftp     stream  tcp     nowait  root    /usr/etc/tcpd   /usr/wu-
ftpd-2.6.0/sbin/in.ftpd -l -a
```

Here the server-program is /usr/etc/tcpd and the arguments are passed to it. In this case it is just the real FTP daemon. I hope you are still following this, because it gets worse. A program was written (ftp_wrapper) following the pattern of tcpd which first executes its arguments and then executes a script. Thus a TIMED MDC system that has the spool files on it will have an inetd.conf line that looks like this:

```
ftp     stream  tcp     nowait  root    /usr/etc/tcpd
/usr/etc/ftp_wrapper /usr/wu-ftpd-2.5.0/etc/ftpd -l –a
```

After executing the FTP daemon, the ftp_wrapper program executes a Bourne Shell script called /usr/etc/ftp_watcher (also written for this application). The version of the script as of the writing of this guide is reproduced below:

```
#!/bin/sh
/tmdc/bin/ftp_monitor >> `date -u +/tmdc/logs/%Y%m%d.log` 2>&1
```

This starts the ftp_monitor script, redirecting its standard output and standard error to a log file in the /tmdc/logs directory named with the current date and time.

In the nominal case (it depends on the host running the script), this script executes another script, “/tmdc/bin/tlm_monitor”. The ftp_monitor script is reproduced below:

```
#!/bin/ksh

######################################
# $Id$ #
# $Log$ #
# $Name$ #
######################################

hostname="`/usr/bin/hostname`"

if [[ $hostname = 'clanton'  || \
      $hostname = 'tmdc-ts2' || \
      $hostname = 'dix' || \
      $hostname = 'tmdc-ts4' ]]
then
    /tmdc/bin/tlm_monitor &
fi


if [ $hostname = 'eaton' ]
then
    if [ "`/software/bin/whoami`" = 'root' ]
    then
        su timedops -c "/tmdc/bin/prod_monitor.p" &
    else
        /tmdc/bin/prod_monitor.p &
    fi
fi
```

The tlm_monitor PERL script uses a simple file lock as a mechanism to prevent multiple copies of itself being instantiated as a result of successively FTP'd files.

The tlm_monitor script globs files whose names match the pattern of the Front End-produced files (*.vc6, *.vc7, *.cmp). It then loops over that file list, first checking to see if the ftp process is complete. It determines this by running the Unix fstat function on the file and comparing the last modification date/time to the current time. If the times differ by more than 5 seconds, it concludes the file transfer is complete.

If the file is complete, the script moves the file from the incoming directory to the ingest directory and initiates an ingest process for it. It terminates when it has initiated processing for all files or 3 hours has elapsed since it started, whichever comes first. If one or more files were processed, it then executes the "WaitThenRunMOCArchMap.pl" script.

The WaitThenRunMOCArchMap script waits for the ingest files to be moved out of the ingest area (this would be done by the ingest processes once the ingest was complete). If there are still files present after 10 minutes, this script exits in error (note that this means the system must be able to ingest some number of files within the 10 minute period). If all the files are processed and moved within the alotted time, this script invokes another script (MOCArchMap.pl) to create a MOC archive map file. Following file creation, the scrip subsequently FTPs the file to the MOC "drop_out_report" directory (using the script FTP2MOC.pl). For example, see oliver:/d3/home/epoch/timed/out/<stream name>/drop_out_report/*.map. The files are named by YYYY + DOY + HHMMSS.map.

MOCArchMap.pl in turn executes the /tmdc/bin/ArchiveMap_sparc_solaris program, feeding it a number of directives ultimately intended for the Archive Server to specify that it wants data for the previous 36 hours for this spacecraft ID, all APIDs, ordered by spacecraft time, and in STP format.

## Default Port Numbers

## Specific TIMED Configurations

### Mini-MOC

During Mini-MOC testing there was a router in the MOC which first received the data, serviced the clients within the MOC and passed on data to a router running in the MDC. The MDC systems were physically collocated with the MOC but are separately managed. The router in the MDC sends data to the spooler and the archive server picks up the data from there.

### JHU/APL I&T

### GSFC I&T

### VAFB Launch

### JHU/APL Operations