Automated Vehicle Identification

Model Deployment Initiative

System Design Document

Version 1.0

January 23, 1998

SwRI Project No. 10-8684

P.O. No. 7-70030

Req. No. 60115-7-70030

Prepared For:

Texas Department of Transportation

TransGuide

3500 NW Loop 410

San Antonio, Texas 78229

Prepared by:

Southwest Research Institute

P.O. Drawer 28510

San Antonio, Texas 78228

Approval Page

_____	_____
MDI AVI Project Manager	Date
_____	_____
SwRI MDI Project Manager	Date
_____	_____
Software Engineering Director	Date

Acronyms and Abbreviations

AC	Alternating Current
ADP	Automatic Data Processing
AT&T	American Telephone and Telegraph
ATIS	Advanced Traveler Information System
ATMS	Advanced Traffic Management System
AWG	American Wire Gauge
bps	Bits Per Second
C	Celsius
CD	Compact Disc
COTS	Commercial Off-The-Shelf
DCIC	Data Collection and Monitoring Component
DFD	Data Flow Diagram
DPFC	Data Processing and Filtering Component
DS	Data Server
FWHA	Federal Highway Administration
GB	Gigabyte
GUI	Graphical User Interface
Hz	Hertz
ICD	Interface Control Document
IRD	Interface Requirements Document
ITS	Intelligent Transportation Systems
IVN	In-Vehicle Navigation
K	1024
MB	Megabyte
MDI	Model Deployment Initiative
MHz	Megahertz
PCC	Process Coordination Component
RAM	Random Access Memory
RF	Radio Frequency
RFO	Request For Offer
ROM	Read Only Memory
SCSI	Small Computer System Interface
SGUIC	Status/Configuration GUI Component
SwRI	Southwest Research Institute
TBD	To Be Determined
TTI	Texas Transportation Institute
TxDOT	Texas Department of Transportation
V	Volts
VAC	Volts AC

Table of Contents

List of Tables

List of Figures

1. SCOPE

This System Design Document (SDD) is developed to provide the requirements and system design for the Automated Vehicle Identification (AVI) Project of the Model Deployment Initiative (MDI). This SDD is based on the *Automated Vehicle Identification Model Deployment Initiative Preliminary Design Document Version 1.0*.

1.1 Identification

The AVI System consists of the AVI Data Processing System, the AVI Reader Field Site Systems, and the AVI Tags. The AVI Reader Field Site Systems and the AVI Tags are being supplied by Amtech Systems Corporation and are documented elsewhere. This document focuses on the AVI Data Processing System, Version 1.0.

1.2 System Overview

The AVI System is one of several systems being developed for the MDI program. The system will be an important source of real-time traffic data for other MDI systems as well as the TransGuide Advanced Traffic Management System (ATMS) and the TransGuide Advanced Traveler Information System (ATIS).

ATMS systems rely on real-time traffic data to detect incidents, monitor traffic flow, and inform the public. Data is traditionally collected by placing sensors in the road to measure the point speed of vehicles as they travel past the sensor. This point speed data is very useful in determining traffic conditions and detecting incidents.

A more comprehensive measurement of traffic conditions is the travel time between two instrumented points. The travel time data considers not only the speed at the endpoints of a link, but also the conditions between the endpoints. The AVI System will provide travel time data which can be used as a source of real-time traffic data in the TransGuide ATMS system, the TransGuide ATIS system, and other MDI systems such as the In-Vehicle Navigation (IVN) system and Traveler Information Kiosk system.

The AVI System involves the deployment of thousands of vehicle tags, referred to as AVI Tags, the installation of multiple AVI Reader Field Site Systems, and the development of a computer system, the AVI Data Processing System, to collect and process data to calculate travel times. An overview of the system concept is presented in Figure 1.

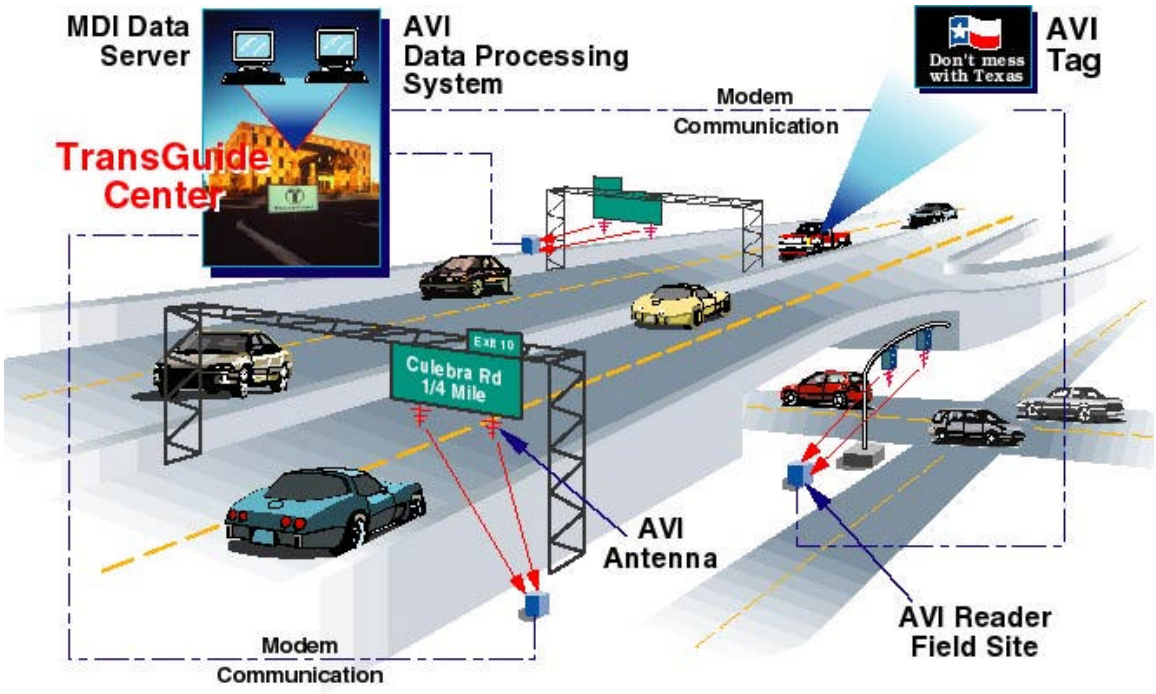


Figure 1. Automated Vehicle Identification System Concept Diagram

The figure shows several vehicles that have been equipped with AVI Tags. As the vehicles pass by the AVI Reader Field Sites, the AVI Antennas recognize the tags and report the tag reads to the AVI Data Processing System, located at the TransGuide Operations Center. The AVI Data Processing System processes this data and reports travel times, travel speeds, and equipment status to the MDI Data Server.

The AVI System is based on Electronic Toll Collection (ETC) technology that is used to automate toll collection at toll facilities. These systems have been adapted in other cities to provide travel time data. The MDI AVI System differs from existing ETC systems in that it does not rely on an existing ETC infrastructure (i.e., existing readers and tags), and it instruments arterial roadways as well as traditional freeways. Also, the MDI In-Vehicle Navigation System and Traveler Information Kiosk System will use the information obtained from the MDI AVI System in unique ways.

Section 1 of this SDD provides introductory information describing the purpose, operational concept, and goals and objectives of the system. Section 2 defines the external interfaces of the system. Section 3 lists the system requirements. Section 4 contains the system design. Finally, Section 5 contains the requirements traceability matrix that will be used throughout the project.

1.3 Purpose of the System

The purpose of the AVI Data Processing System is to collect tag read data from AVI Reader Field Sites, computes vehicle travel times and speeds along instrumented sections of roadway, and make this data available to other systems within the TransGuide and MDI environment.

1.4 Operational Concept

When vehicles equipped with AVI Tags pass by AVI Reader Field Sites, the readers contact the AVI Data Processing System to report the tag read information. The AVI Data Processing System receives the tag and determines if any other readers in the system have read the tag. If the AVI Data Processing System determines that another reader has read the tag, it calculates the time that it took the tag to travel between the two readers. From this value, it is able to derive the travel speed of the vehicle.

The roadways that are instrumented by the AVI System are divided into links. The AVI Data Processing System correlates tag matches into average travel times and travel speeds and assigns these values to the appropriate links. This data is reported to the MDI Data Server System so that it may be used by other systems.

The AVI Data Processing System maintains the status of each of the AVI Reader Field Sites as well as its own status and reports this data to the MDI Data Server and to the TransGuide operations staff.

1.5 Goals and Objectives

The AVI System has the following goals. The system should:

- provide real-time traffic condition information,
- provide data that can be used by traffic management personnel to manage traffic,
- provide data that can be used by the traveling public,
- be flexible to allow additional sensors in the future,
- provide easy system diagnosis and configuration, and
- process data in a timely manner to make the data available in a real-time fashion.

1.6 Referenced Documents

Campbell, Joe. *C Programmer's Guide to Serial Communications, 2nd Edition*. Sams Publishing, 1994.

Department of Planning, City of San Antonio Metropolitan Planning Organization. *Travel Rate Study Technical Memorandum*. 1995.

Mitretek Systems. *Review of ITS Benefits: Emerging Successes*. Report FHWA-JPO-97-0001, FHWA, U.S. Department of Transportation.

Lomax, T., S. Turner, and G. Shunk. *Quantifying Congestion: User's Guide*. Draft NCHRP, TRB, NRC, 1996.

Ogle, Jennifer, and Joseph Baumgartner. *Considerations in the Development and Implementation of a Travel Speed Database Integrating ATMS, AVI, GPS, and Theoretical Data*. Paper No. 981104, Proceedings of the 77th Annual Transportation Research Board, January, 1998.

Southwest Research Institute. *Automated Vehicle Identification Model Deployment Initiative Preliminary Design Document, Version 1.0*. February, 1997.

Southwest Research Institute. *Data Server Model Deployment Initiative System Design Document, Version 1.0*. January, 1998.

Southwest Research Institute. *In Vehicle Navigation Model Deployment Initiative System Design Document*. January, 1998.

Southwest Research Institute. *Proposal for the Model Deployment Initiative System Integration*. SwRI Proposal No. 10-20342, November, 1996.

Texas Department of Transportation. *Request for Offer (RFO) for the Model Deployment Initiative System Integration, 60115-7-70030*. Specification No. TxDOT 795-SAT-01, October, 1996.

2. EXTERNAL INTERFACES

The AVI Data Processing System has three external interfaces: the MDI Data Server, The AVI Reader Field Sites, and the AVI Operator. Figure 2 shows the interfaces between the AVI Data Processing System and the external systems. Each of these is discussed in more detail in the following sections.

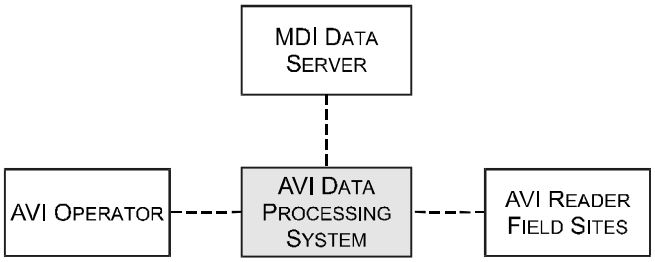


Figure 2. AVI Data Processing System External Interfaces

2.1 Model Deployment Initiative Data Server

The MDI Data Server is a data repository of real-time travel data. The AVI Data Processing System acts as a data generator to the Data Server, providing travel time, travel speed, and equipment status data to the Data Server. The Data Server distributes the AVI data to data consumer clients that use the data in a variety of ways.

2.2 Automated Vehicle Identification Operator

The AVI Data Processing System interfaces with the TransGuide operations staff. The operations staff oversee the operation of the AVI System by monitoring the travel time and travel speed data, monitoring the status of the equipment in the system, and monitoring the status of the software components of the system.

2.3 Automated Vehicle Identification Reader Field Site

The AVI Data Processing System interfaces with the AVI Reader Field Sites by collecting data from the readers. The readers detect AVI Tags and report the tag identifiers to the AVI Data Processing System. The AVI Data Processing System sends commands to the AVI Reader Field Sites.

3. SYSTEM REQUIREMENTS

The following sections contain the system requirements for the AVI Data Processing System. The requirements are organized by level and category. The levels that are defined in this document are *general*, *system* and *subsystem*. General requirements are non-technical requirements that apply to the program in general. System requirements apply to the system level of the AVI Data Processing System and subsystem requirements apply to the subsystem design elements.

The categories of requirements that are defined are general, interface, functional, performance, physical, and miscellaneous. If there are no requirements of a particular category at a particular level, there is no reference to that category at that level. Each of these categories are described below.

- General requirements describe a requirement that applies to the overall program (e.g., documentation).
- Interface requirements describe the interface between the system and external systems (e.g., the user interface).
- Functional requirements describe the operations that the system must perform (e.g., initialize, acquire data).
- Performance requirements describe performance constraints of the system (e.g., CPU utilization, minimum throughput).
- Physical characteristic requirements describe physical constraints of the system (e.g., maximum size, and minimum weight).
- Miscellaneous requirements describe other constraints of the system (e.g., maintainability).

In addition to the categories described above, there are three types of requirements presented in these sections: MDI RFO requirements, SwRI MDI proposal requirements, and derived requirements. Where a conflict exists, the SwRI MDI Proposal requirements supersede the MDI RFO requirements. In these cases, only the SwRI MDI Proposal requirements are documented. Derived requirements are generated by analysis of the existing requirements.

Several notations are used in the following tables. The requirement number is a three-part number that is used to uniquely identify each requirement. The number consists of the following fields:

<System Mnemonic>-<Requirement Category Mnemonic>-<Requirement Number>

System Mnemonic

The system mnemonic uniquely identifies the AVI Data Processing System to distinguish its requirements from the requirements of the other MDI systems. The system mnemonic for the AVI Data Processing System is *AVI*.

Requirement Category Mnemonic

A mnemonic has been created for each of the requirement categories. They are *GN* - general, *IF* - interface, *FN* - functional, PF - performance, *PY* - physical, and *MS* - miscellaneous.

Requirement Number

The requirements are numbered sequentially within a given category. The requirements at the system level each have a single requirement number. As requirements are derived at the subsystem level, additional numbers are added to show the relationship between requirements. For example, requirement *AVI-IF-1* at the system level may have two children and the subsystem level, *AVI-IF-1.1* and *AVI-IF-1.2*. With this numbering scheme it is easy to determine a requirement's parent and the level of the requirement.

Each of these requirements are further documented in Section 5 in the traceability matrix. For each requirement, the matrix contains traceability information to show the relationship between the requirement and other requirements, design elements, and the Acceptance Test Plan (ATP).

3.1 General Requirements

This section contains the general requirements for the system. These requirements are non-technical and involve such things as deliverable documentation. The general requirements for the AVI Data Processing System are listed in Table 1.

Table 1. General Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-GN-1	The program shall have its developmental progress documented every four weeks.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-2	The program shall have a final report prepared at the end of the project.	Required by SwRI proposal.
AVI-GN-3	The program shall have its status reported in interim reports, as necessary.	Required by SwRI proposal.
AVI-GN-4	The program shall have a Software Development Plan.	Required by SwRI proposal.
AVI-GN-5	The program shall have a Software Requirements Specification.	Required by SwRI proposal.
AVI-GN-6	The program shall have a Software Design Document.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-7	The program shall have a Software Acceptance Test Plan.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-8	The program shall have a Software User Manual.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-9	The program shall have a Software Version Description Document.	Required by SwRI proposal.
AVI-GN-10	The program shall have training provided on AVI Reader Field Sites.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-11	The program shall have training provided on the AVI Data Processing System.	Required by TxDOT RFO and SwRI proposal.
AVI-GN-12	The program shall have a Tag specification developed.	Required by TxDOT RFO and SwRI proposal.

3.2 System Level Requirements

The following sections contain the system level requirements for the MDI AVI Data Processing System.

3.2.1 Automated Vehicle Identification System Interface Requirements

The interface requirements for the AVI Data Processing System are listed in Table 2.

Table 2. Automated Vehicle Identification System Interface Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-IF-1	The system shall interface to the MDI Data Server.	The MDI Data Server will be the repository for travel data generated by the AVI Data Processing System.
AVI-IF-2	The system shall interface to the Reader Field Site.	The Reader Field Site will be the source of AVI Tag read data for the AVI Data Processing System.
AVI-IF-3	The system shall interface with the user.	The user will configure the system, monitor its status, and monitor both AVI Tag read data and travel speed and time data.

3.2.2 Automated Vehicle Identification System Functional Requirements

The functional requirements for the AVI Data Processing System are listed in Table 3.

Table 3. Automated Vehicle Identification System Functional Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-FN-1	The AVI Data Processing System shall gather AVI Tag read data from Reader Field Sites.	The AVI Tag read data must be acquired to calculate travel speed and time.
AVI-FN-2	The AVI Data Processing System shall process AVI Tag read data.	The AVI Data Processing System will use (process) the AVI Tag read data to generate the travel speed and time.
AVI-FN-3	The AVI Data Processing System shall process status data.	The AVI Data Processing System will gather process data internally to control itself as well as reporting its status to the user.
AVI-FN-4	The AVI Data Processing System shall allow for configuration of system components.	The AVI Data Processing System will be able to be easily reconfigurable. This includes remote reconfiguration of field sites.
AVI-FN-5	The AVI Data Processing System shall handle errors.	The AVI Data Processing System will be able to detect, report, and recover (in some cases) from error conditions.

3.2.3 Automated Vehicle Identification System Physical Characteristic Requirements

The physical characteristic requirements for the AVI Data Processing System are listed in Table 4.

Table 4. Automated Vehicle Identification System Physical Characteristic Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-PY-1	The AVI Data Processing System shall have a dedicated master computer.	A dedicated computer is required to assure adequate computer system response.
AVI-PY-2	The AVI Data Processing System shall have a modem pool.	A modem pool is required to simulate leased line performance.

3.2.4 Automated Vehicle Identification System Miscellaneous Requirements

The miscellaneous requirements for the AVI Data Processing System are listed in Table 5.

Table 5. Automated Vehicle Identification System Miscellaneous Requirements

REQUIREMENT	REQUIREMENT	RATIONAL

NUMBER		
AVI-MS-1	The AVI Data Processing System shall have software designed using either structured or object-oriented methodologies.	A good design approach is desirable.
AVI-MS-2	The AVI Data Processing System shall have software written in C or C++.	This is necessary to be compatible with the target platform and existing TxDOT computer tools.
AVI-MS-3	The AVI Data Processing System shall have software tested at the unit and system integration levels, as appropriate.	These levels of testing are necessary to enhance confidence in the software produced.

3.3 Subsystem Level Requirements

The subsystem level requirements for the AVI Data Processing System are described in the following sections. The AVI subsystems include the AVI Master Computer System, the AVI Modem Server System, and the AVI Data Processing Software.

3.3.1 AVI Master Computer System Requirements

The following sections contain the requirements for the AVI Master Computer System.

3.3.1.1 AVI Master Computer System Interface Requirements

The interface requirements for the AVI Master Computer System are listed in Table 6.

Table 6. AVI Master Computer System Interface Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-IF-1.1	The AVI Master Computer System shall interface with the MDI Data Server.	The MDI Data Server will be the repository for travel data generated by the AVI Data Processing System.

3.3.1.2 AVI Master Computer System Physical Characteristic Requirements

The physical characteristic requirements for the AVI Master Computer System are listed in Table 7.

Table 7. AVI Master Computer System Physical Characteristic Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-PY-1.1	The AVI MC shall be a Sun Microsystems Ultra SPARCStation or better	A computer of this capability can meet the expected processing requirements of the AVI MC.
AVI-PY-1.2	The AVI MC shall have, at a minimum, the following items: • 167mhz SPARC CPU • 4.2 GB Hard Disk • 128 MB RAM • Floppy Disk Drive • Sun CD-ROM drive • Turbo GX+ Graphics card	Required by TxDOT RFO and SwRI proposal.

	• 20" Sun color monitor • 2 Ethernet cards • 2 SCSI channels • 2 RS-232 ports • Keyboard • 2-Button mouse • 64 serial ports (SCSI attached) • 1 modem/phone line	

3.3.2 AVI Modem Server System Requirements

The requirements for the AVI Modem Server System are listed in the following sections.

3.3.2.1 AVI Modem Server System Interface Requirements

The interface requirements for the AVI Modem Server System are listed in Table 8.

Table 8. AVI Modem Server System Interface Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-IF-2.1	The AVI Modem Server System shall interface with the AVI Reader Field Sites.	The AVI Reader Field Site will be the source of AVI Tag read data for the AVI Data Processing System.

3.3.3 AVI Data Processing Software Requirements

The requirements for the AVI Data Processing Software are listed in the following sections.

3.3.3.1 AVI Data Processing Software Interface Requirements

The interface requirements for the AVI Data Processing Software are listed in Table 9.

Table 9. AVI Data Processing Software Interface Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-IF-1.2	The AVI Data Processing Software shall interface with the MDI Data Server.	The MDI Data Server will be the repository for travel data generated by the AVI Data Processing Software.
AVI-IF-2.2	The AVI Data Processing Software shall interface with the AVI Reader Field Sites.	The Reader Field Sites will be the source of AVI Tag read data for the AVI Data Processing Software.
AVI-IF-3.1	The AVI Data Processing Software shall interface with the user.	The user will configure the system, monitor its status, and monitor both AVI Tag read data and travel speed and time data.

3.3.3.2 AVI Data Processing Software Functional Requirements

The functional requirements for the AVI Data Processing Software are listed in Table 10.

Table 10. AVI Data Processing Software Functional Requirements

REQUIREMENT NUMBER	REQUIREMENT	RATIONAL
AVI-FN-1.1	The AVI Data Processing Software shall gather AVI Tag read data from the AVI Reader Field Site.	The AVI Tag read data must be acquired to calculate travel speed and time.
AVI-FN-2.1	The AVI Data Processing Software shall store the AVI Tag read data collected from the Reader Field Sites .	The AVI Tag read data must be archived.
AVI-FN-2.2	The AVI Data Processing Software shall provide travel time and speed data to the MDI Data Server.	The MDI Data Server will be the repository for AVI Tag read data.
AVI-FN-2.3	The AVI Data Processing Software shall process tag read data gathered from the Reader Field Sites.	The AVI Data Processing Software will use (process) AVI Tag read data to generate the travel speed and time.
AVI-FN-3.1	The AVI Data Processing Software shall determine AVI Reader Field Site status.	The AVI Data Processing Software will gather external status data to control itself and to report to the user.
AVI-FN-3.2	The AVI Data Processing Software shall monitor its own status.	The AVI Data Processing Software will gather internal status data to control itself and to report to the user.
AVI-FN-4.1	The AVI Data Processing Software shall accept configuration data.	The AVI Data Processing Software will get new configuration information from the user.
AVI-FN-4.2	The AVI Data Processing Software shall store configuration data.	The AVI Data Processing Software will maintain its configuration information.
AVI-FN-4.3	The AVI Data Processing Software shall perform configuration operations.	The AVI Data Processing Software will be able to apply its configuration information.
AVI-FN-5.1	The AVI Data Processing Software shall detect communications errors between itself and the AVI Reader Field Site.	The AVI Data Processing Software will be able to detect some communications errors.
AVI-FN-5.2	The AVI Data Processing Software shall log communications errors between itself and the AVI Reader Field Site.	The AVI Data Processing Software will record detectable communications errors.
AVI-FN-5.3	The AVI Data Processing Software shall report communications errors between itself and the AVI Reader Field Site.	The AVI Data Processing Software will report detectable communications errors.

4. SYSTEM DESIGN

The system design is presented in two forms: system architecture and software design. The system architecture defines the framework of the major hardware and software components of the AVI System and the methods used to communicate between the components. The software design focuses on the software components of the AVI System using data flow analysis and structure charts. More attention is paid to the software components of the AVI System because the software components have been custom-developed. Less attention is paid to the hardware components, which are off-the-shelf systems.

Because Amtech Systems Corporation has supplied the AVI Reader Field Sites and AVI Tags under a separate contract, this document discusses these systems only briefly. The primary focus of this document is the AVI Data Processing System.

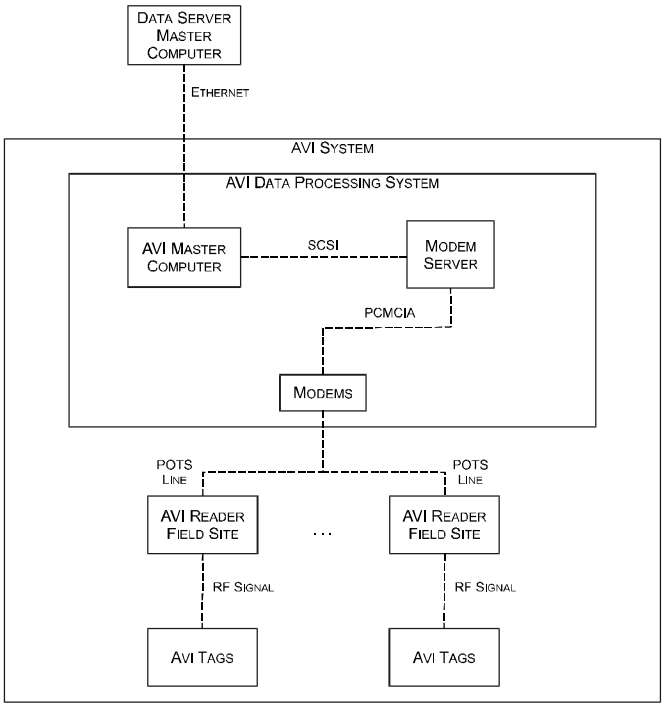


Figure 3. Automated Vehicle Identification System Block Diagram

4.1 System Architecture

Two views of the AVI System architecture are presented: the hardware architecture and the software architecture. Both views utilize block diagrams to illustrate the components of the architecture and the methods of communication between the components.

4.1.1 Hardware Architecture

The hardware architecture of the AVI System is presented in Figure 3. This figure shows the three main components of the AVI System: the AVI Data Processing System, the AVI Reader Field Site Systems, and the AVI Tags. The AVI Reader Field Sites and AVI Tags are being supplied by Amtech Systems Corporation and are discussed only briefly in this document.

The AVI Data Processing System is connected to the Data Server Master Computer via Ethernet. This Ethernet connection is the TransGuide Ethernet network. Though not shown, the TransGuide Ethernet network consists of a collection of cables, routers, and bridges that serve many computer systems, including the AVI Master Computer and the Data Server Master Computer.

The AVI Data Processing System is connected to the AVI Reader Field Sites via plain old telephone system (POTS) lines. There are two POTS lines installed for each AVI Reader Field Site – one for the AVI Master Computer and one for the AVI Reader Field Site. The AVI Reader Field Sites and the AVI Data Processing System are both designed to maintain a constant connection, thus ensuring the most timely data collection possible. It is the responsibility of the AVI Reader Field Sites to initiate and maintain communication with the AVI Data Processing System.

The AVI Reader Field Sites communicate with the AVI Tags via a RF signal that is emitted by the AVI Reader Field Site. This signal excites the transponder of a passing AVI Tag, causing the tag to emit a RF signal. The AVI Reader Field Site reads the signal emitted by the AVI Tag, which contains the unique AVI Tag identifier.

The three major components of the AVI System are discussed in more detail in the following sections.

4.1.1.1 AVI Data Processing System

Figure 3 shows the three major components of the AVI Data Processing System, which consists of the AVI Master Computer, the AVI Modem Server, and several PCMCIA modems. Drawing 8684-5003, which is included in the *In Vehicle Navigation System Design Document*, contains detailed drawings of the physical characteristics of these systems and wiring diagrams illustrating the physical installation of the system.

The AVI Master Computer communicates with the AVI Modem Server via a SCSI interface. A special vendor-supplied software driver executes on the AVI Master Computer to allow the master computer to access the AVI Modem Pool serial ports through the SCSI interface. This vendor-supplied software driver creates one logical serial port for each of the available modem slots in the AVI Modem Server.

The AVI Modem Server contains slots for PCMCIA modems. These modems contain connection points where POTS telephone lines can be attached. The number of PCMCIA modems installed in the AVI Modem Server is equal to the number of telephone lines installed for communicating with the AVI Reader Field Sites.

The AVI System telephone lines are installed in a *hunt* configuration. Each of the AVI Reader Field Sites uses the same base telephone number to access the AVI Data Processing System. The telephone service provider assigns incoming calls to the next available line in the system. Therefore, the POTS line to which the call is assigned determines which modem and serial port will service an incoming call.

4.1.1.2 AVI Reader Field Site System

The AVI Reader Field Site Systems are being supplied by Amtech Systems Corporation and are discussed only briefly in this document. The reader field sites consist of a number of components including an automatic data processor (ADP), reader modules, RF modules, antennas, a cabinet, and a modem. A block diagram of the AVI Reader Field Site System is shown in Figure 4 and the components of the system are described below.

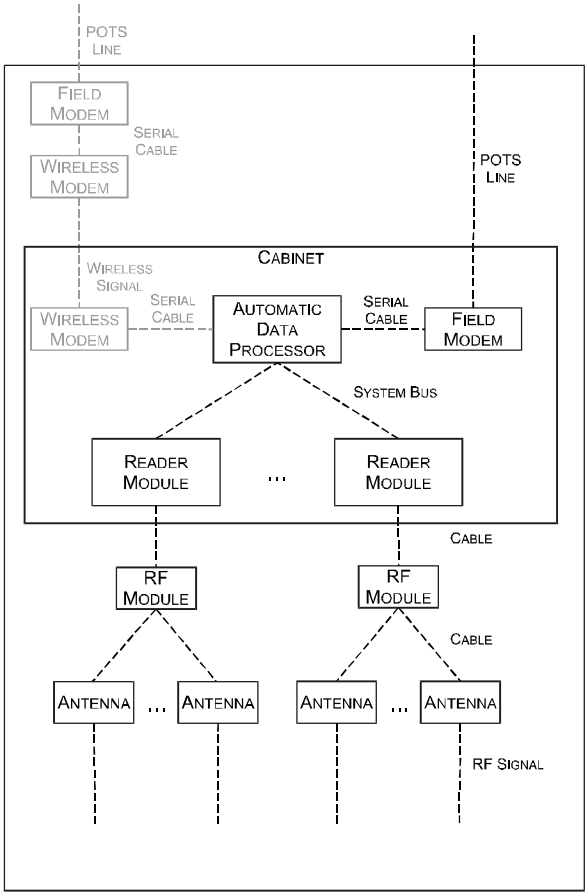


Figure 4. AVI Reader Field Site System Block Diagram

The ADP is the central processing unit of the AVI Reader Field Site System. It initializes the system, controls the reader modules, and communicates via the modem with the AVI Data Processing System. The ADP can contain up to four Reader Modules.

The Reader Modules are connected via specialized cable to the RF Modules and can accommodate up to two RF Modules. The Reader Module controls the RF Module and receives the data that is read by the module in response to the RF Module signal.

The field modem is connected to a POTS line and is controlled by the ADP unit, which initiates communication with the AVI Data Processing System. The ADP is programmed to dial the AVI Data Processing System base telephone number to initiate communication. The modem communication can be configured in one of two ways. The default configuration, shown in the figure in black, is to connect the ADP directly to the field modem. The second configuration, shown in the figure in gray, is to connect the ADP to a wireless modem, which communicates with another wireless modem to pass the information along to the field modem. This configuration is used in locations where telephone service cannot be supplied directly to the ADP cabinet.

4.1.1.3 AVI Tags

The AVI Tags are being supplied by Amtech Systems Corporation and are discussed only briefly in this document. The AVI Tags are passive read-only tags that are used in Electronic Toll Collection (ETC) systems. These tags contain a transponder, which emits a RF signal containing a tag identifier whenever the Reader Module energizes the transponder. This tag identifier uniquely identifies the tag.

4.1.2 Software Architecture

The software architecture is presented in Figure 5 and shows the major software components of the AVI System and the method of communication between them. This diagram corresponds very closely to the hardware architecture diagram presented in Section 4.1.1. The software architecture shows two main components: the AVI Data Processing Software and the AVI Reader Field Site Software. Each of the AVI Data Processing System components shown in the diagram represents a UNIX process.

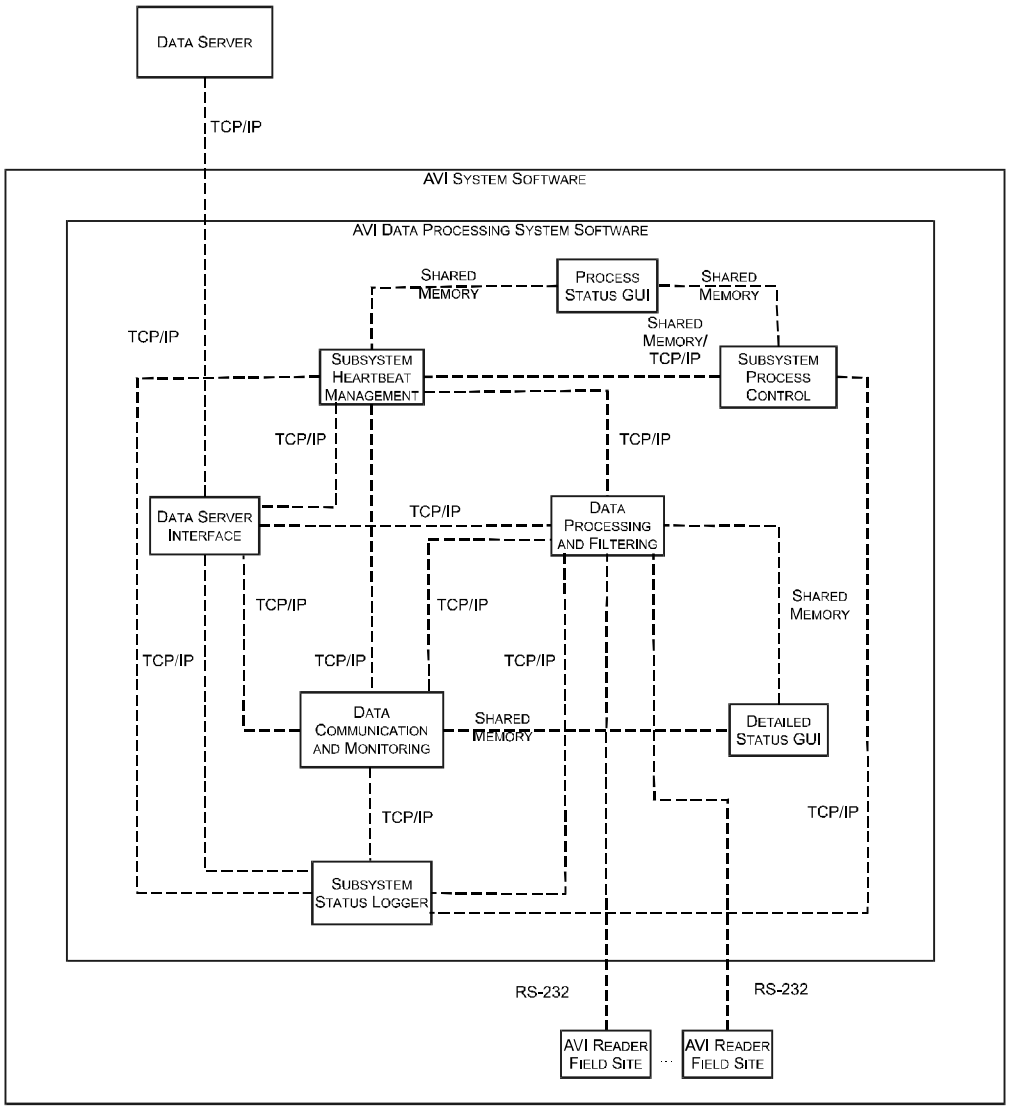


Figure 5. Automated Vehicle Identification Software Block Diagram

The AVI Data Processing Software communicates with the Data Server at the network layer using the TCP/IP protocol. This protocol provides a standardized mechanism for two software applications running on two different host computers to communicate with each other. At the application layer, the two applications communicate using a proprietary protocol defined by the Data Server itself. This protocol is documented in the *Model Deployment Initiative Data Server System Design*.

The AVI Data Processing Software communicates with the AVI Reader Field Site Software at the network layer using RS-232. This protocol is a standard protocol for serial devices. At the application layer, the applications communicate using a proprietary protocol defined by Amtech Systems Corporation. This protocol defines the data messages and the sequencing of the messages between the reader and the host software.

Within the AVI Data Processing Software there are several components. These components communicate using one of two methods: TCP/IP or shared memory. The decision to use one

method over the other is based primarily on ease of implementation and the potential for software components to be distributed to other host systems in the future.

The Subsystem Master Process communicates with the Process Status GUI and the Subsystem Heartbeat Process using shared memory. The shared memory is created by the Subsystem Master Process and accessed by the Process Status GUI and the Subsystem Heartbeat Process.

The Data Collection and Monitoring process communicates with the Data Processing and Filtering process and the Data Server Interface process at the network layer using TCP/IP. In both cases, the Data Collection and Monitoring process acts as the client. The application layer protocol between each of the processes is a proprietary protocol that is shared by these processes. The Data Collection and Monitoring process also communicates with the Detailed Status GUI via shared memory. The shared memory is created by the Data Collection and Monitoring process and accessed by the Detailed Status GUI.

The Data Processing and Filtering process communicates with the Detailed Status GUI via shared memory. The Data Processing and Filtering process creates the shared memory, which is accessed by the Detailed Status GUI.

Each of the AVI Data Processing Software processes communicates with the Subsystem Heartbeat Process and the Status Logger process at the network layer using TCP/IP. The application layer protocol for communicating with each of these processes is defined by the process itself. These protocols are defined in the documentation for the Subsystem Heartbeat Process and the Status Logger Process.

4.2 System Level Design

The system level design focuses on the AVI Data Processing Software. Three views of the software design are presented: the data model, data flow model, and structure charts. The data model discusses some of the more important data structures that are used by the AVI Data Processing System. The data flow model describes the data that flows through the system and the logical processes that transform the data as it flows from input to output. The structure charts provide a physical road map of the software source code, describing the functions that compose the software.

4.2.1 Data Model

The AVI Data Processing Software uses several data tables to perform its functionality. These include the site table, the link table, and the site-link cross-reference table. This section describes the methodology used to create these tables and provides a definition of the format of each table.

4.2.1.1 Site Table

The site table contains a record for each site in the AVI System. In order to collect the data to populate this table, the site locations were determined and a site naming convention was established. Detailed information about this process is presented in the following sections and in the paper *Considerations in the Development and Implementation of a Travel Speed Database Integrating ATMS, AVI, GPS, and Theoretical Data* by Jennifer Ogle and Joseph Baumgartner.

4.2.1.1.1 AVI Site Selection

Sites were selected by consulting several information sources including: *Quantifying Congestion User's Guide* produced by the Texas Transportation Institute, *Travel Rate Study Technical Memorandum* produced by the San Antonio Metropolitan Planning Organization, and *Review of ITS Benefits: Emerging Successes* produced by Mitretek Systems.

The Mitretek report indicated that the majority of AVI System benefits could be obtained by locating sites in the most congested locations in an urban area. Specifically, this report concluded that reporting travel times for the most congested links provides 90% of the system benefit at a lower cost. The *Travel Rate Study Technical Report* was used as an initial guideline for the determination of San Antonio's most congested roadways.

After earmarking the sites mentioned in the *Travel Rate Study Technical Memorandum*, traffic volumes from the TxDOT 1995 District Highway Traffic Map of the San Antonio area were correlated with the proposed AVI site locations. These traffic volumes were representative of the annual average daily traffic (AADT) volume for a 24-hour period. In order to determine the peak hour traffic volume, peak hour percentages of AADT were taken from the TxDOT 1995 Permanent Automatic Traffic Recorder Stations Data and applied to the daily volumes. The determination of relative congestion was based on the ratio of peak hour traffic volume of the roadway versus capacity of the roadway. Capacity was estimated for each AVI site using information on the total number of lanes in the roadway, and the occurrence of traffic signals, if any. As the peak hour volume approaches hourly capacity, the volume to capacity ratio (V/C) approaches one, signifying increasing congestion. The resultant V/C ratios represent a macroscopic view showing relative estimates of congestion in the San Antonio area.

Although V/C ratios were the primary factor in placing the AVI sites, other factors were considered. To maximize the ability of the TransGuide system to collect data, the AVI sites were not placed in areas that were served by the TransGuide ATMS. Also, consideration was given to place sites in all parts of the city. Site cost, which was influenced by the availability of existing infrastructure (e.g., mechanical structure, telephone service, and electrical service), was also considered.

Lastly, the *Quantifying Congestion User's Guide* was used to determine the appropriate linear spacing of the sites. Table 11 shows the spacing guidelines there were applied based on road classification (i.e., expressway/arterial) and volume (i.e., high/low access).

Table 11. AVI Site Spacing Guidelines

	EXPRESSWAYS	ARTERIALS
HIGH ACCESS	1 - 3 miles	1 - 2 miles
LOW ACCESS	3 - 5 miles	2 - 3 miles

4.2.1.1.2 Site Naming Convention

Each AVI Reader Field Site has two identifiers. The Amtech identifier is a unique identifier assigned by Amtech Systems Corporation at the time of installation. This identifier is stored electronically in the reader and is used to uniquely identify the reader when it connects to the AVI Data Processing System.

The second identifier is the TransGuide AVI Site Identifier which is a unique identifier that names the site and allows the TransGuide operations staff to determine the location of a site based on its name. The TransGuide identifier is created using a standard methodology based on the location and type of reader installation.

The format of the TransGuide AVI Site Identifier closely resembles the format of the TransGuide Link Identifiers. It is a 19-character string broken into fields with the format RDAAAAA-BBBBB-CCCCC, which is described in Table 12.

Table 12. TransGuide AVI Site Identifier Fields

FIELD	NAME	WIDTH	VALUES	DESCRIPTION
R	Identifier Type	1	R	Specifies the type of identifier so that it is clear that this identifier is for an AVI reader site.
D	Direction	1	B, N, S, E, W	Specifies the direction that the reader serves. "B" is used for bi-directional.
AAAAA	Road Name	5	Alphanumeric	Identifies the name of the road that the reader serves.
-	Separator	1	-	Field separator.
BBBBB	Cross Street	5	Alphanumeric	Identifies the name of the closest major cross street.
-	Separator	1	-	Field separator.
CCCCC	Support Structure	5	OSB__ WIRE_ SPOLE BRIDG	Indicates the type of structure on which the reader antennas are mounted. Can be an overhead sign bridge (OSB), span wire, service pole, or traffic bridge.

4.2.1.1.3 Site Table Definition

The site table contains a record for each AVI Reader Field Site. Each record consists of the TransGuide AVI Site Identifier and the associated Amtech reader identifier.

4.2.1.2 Link Table

The link table contains a record for each link in the AVI System. In order to collect the data to populate this table, the roadways were divided into a network of links and a naming convention for the links was established. Detailed information about this process is presented in the following sections and in the paper *Considerations in the Development and Implementation of a Travel Speed Database Integrating ATMS, AVI, GPS, and Theoretical Data* by Jennifer Ogle and Joseph Baumgartner.

4.2.1.2.1 AVI Link Selection

The most logical way to define the links is to use the AVI Reader Field Sites as endpoints for the links. In the simplest case, a single link is sufficient to define the roadway between two reader sites. This case occurs when a reader has a single adjacent reader to which a vehicle can travel. This case is illustrated in Figure 6.

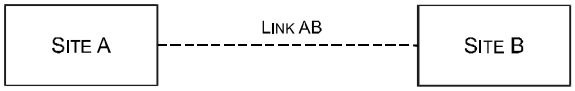


Figure 6. Simple AVI Link Definition

A more complicated case arises when there are multiple readers to which a vehicle can travel after leaving a reader, and the paths between the two readers share a common path for a portion of the distance. Figure 7 illustrates this case. A vehicle leaving *Site A* can travel to either *Site B* or *Site C*. Regardless to which reader the vehicle travels, there is a section of roadway that is common to both paths. To resolve this situation, a *virtual node* is created where the two paths diverge and three links are created.

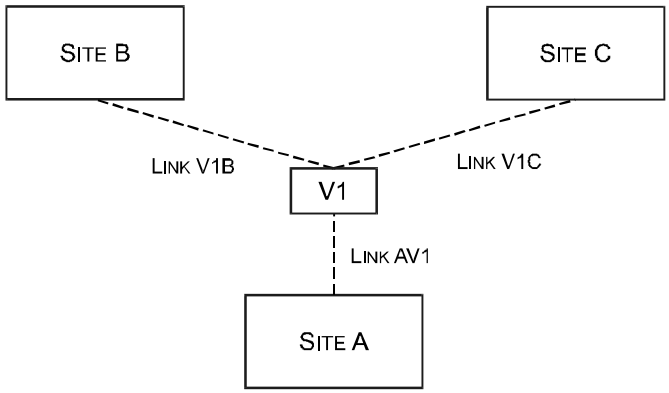


Figure 7. Complex AVI Link Definition

4.2.1.2.2 Link Naming Convention

The TransGuide AVI Link Identifier is a unique identifier that names a link and allows the TransGuide operations staff to determine the location of a link based on its name. The identifier is created using a standard methodology based on the location of the link.

The format of the TransGuide AVI Link is a 19-character string broken into fields with the format IDAAAAA-BBBBB-CCCCC, which is described in Table 13.

Table 13. AVI Link Identifier Fields

FIELD	NAME	WIDTH	VALUES	DESCRIPTION
I	Identifier Type	1	I	Specifies the type of identifier so that it is clear that this identifier is for an AVI link.
D	Direction	1	N, S, E, W	Specifies the direction of travel along the link.
AAAAA	Road Name	5	Alphanumeric	Identifies the name of the road that the link follows.
-	Separator	1	-	Field separator.
BBBBB	North or East Cross Street	5	Alphanumeric	Identifies the northern or eastern (depending on the Direction) cross street.
-	Separator	1	-	Field separator.
CCCCC	South or West Cross Street	5	Alphanumeric	Indicates the southern or western (depending on the Direction) cross street.

4.2.1.2.3 Link Table Definition

The link table contains an entry for each TransGuide AVI Link Identifier. No other data is contained in the table.

4.2.1.3 Site-Link Table

In order to calculate travel times from tag reads, the AVI Data Processing software must know the relationship between AVI Reader Field Sites and TransGuide AVI Link Identifiers. The software must also know the length of each of the links so that the travel speed can be computed. The site-link table provides this data.

The AVI Reader Field Sites are organized as a set of *source-destination* pairs. One or more link identifiers connect each source-destination pair. The site-link table contains one record for each source-destination pair in the AVI System.

A record in the site-link table has the following fields:

<source site id> <destination site id><threshold> < <link id> <length> <nominal value> >[*]

The asterisk indicates that these fields are repeated as necessary. The fields of the site-link record are described in Table 14.

Table 14. Site-Link Record Definition

FIELD	VALUES	DESCRIPTION
source site id	A valid TransGuide AVI Site Identifier	The source site in a source-destination pair.
destination site id	A valid TransGuide AVI Site Identifier	The destination site in a source-destination pair.
threshold	(0.0 - 1.0)	A value indicating the threshold that should be used for filtering.
link id	A valid TransGuide AVI Link Identifier	A link identifier that is part of the path connecting the source site with the destination site.
length	Numeric	The length of the link.
nominal value	Numeric	The nominal speed of the link.

4.2.2 Data Flow Design

The data flow design is presented in a series of data flow diagrams. These diagrams use several symbols to represent data flows, data stores, and logical processes that manipulate the data. A legend is presented in Figure 8 and a description of each of the symbols is presented in Table 15.

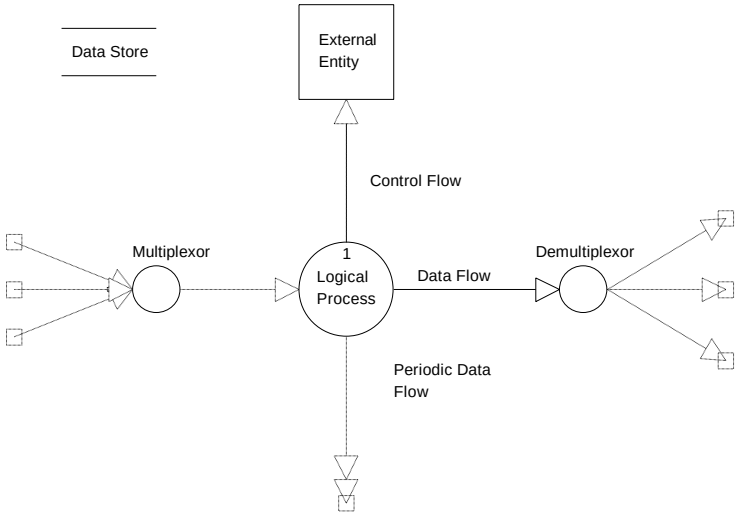


Figure 8. Data Flow Diagram Legend

Table 15. Data Flow Diagram Legend

SYMBOL	DESCRIPTION
Control Flow	A flow that passes control data from one process to another.
Data Flow	Data element that is passed from one process to another. The flows can represent single data items or collections of data.
Data Store	Storage mechanism for data (e.g., disk file, shared memory segment).
Demultiplexor	Symbol used to split an aggregate data flow into its individual data flows.
External Entity	An external process that provides or receives data from the system.
Logical Process	An entity that transforms data. In most cases this corresponds to a UNIX process, but it can also represent the logical transformation of data within a UNIX process.
Multiplexor	Symbol used to combine several data flows into an aggregate data flow.
Periodic Data Flow	A data flow that occurs on a periodic basis (e.g., once a minute).

The data flow design begins at the highest level with the context diagram and is decomposed hierarchically to reveal increasing levels of detail. The design is presented in depth-first order, meaning that a process will be expanded to its lowest level before the next process is discussed.

The context diagram for the AVI Data Processing Software is presented in Figure 9. The figure shows the AVI Data Processing Software process in the center, surrounded by each of the external entities that the system interfaces with. The Subsystem Process Control, Subsystem Heartbeat Management, Subsystem Status Logger, and Process Status GUI are shown as external entities because they are documented elsewhere. In reality, these processes are an integral part of the AVI Data Processing Software.

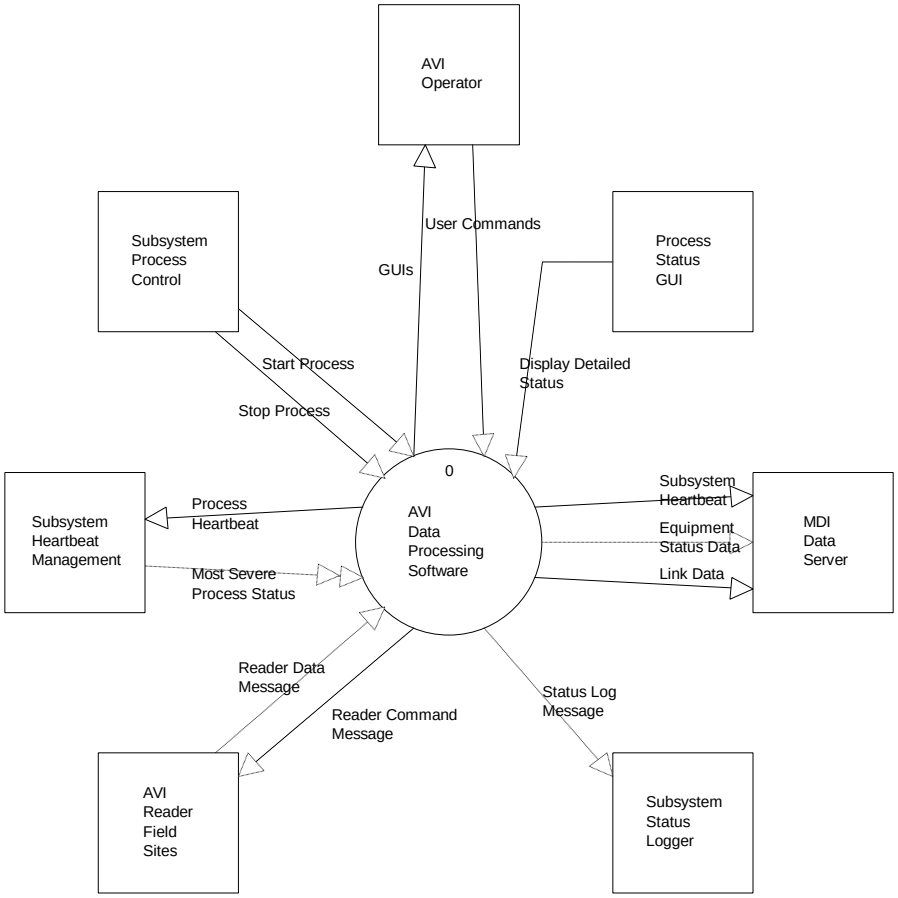


Figure 9. AVI Data Processing Software Context Diagram

The external entities and data flows of the AVI Data Processing Software context diagram are described in Table 16 and Table 17.

Table 16. AVI Data Processing Software Context Diagram External Entities

ITEM	DESCRIPTION
AVI Operator	The AVI Operator represents the operations and system administration personnel assigned to the AVI System. These are end-users of the AVI Data Processing Software and will interact with the system via graphical user interfaces associated with the detailed status GUI.
AVI Reader Field Sites	The AVI Reader Field Sites are the AVI readers which are located in the field. The readers represent several pieces of equipment including a modem, a data processing computer, an RF module, and one or more antennas.

ITEM	DESCRIPTION
MDI Data Server	The MDI Data Server is the central repository of information generated and maintained by the MDI subsystems. The AVI Data Processing Software sends link data and equipment status data to the Data Server. The Data Server also receives the subsystem-level heartbeat which includes the overall status of the AVI Data Processing Software.
Process Status GUI	The Process Status GUI is the graphical user interface providing the visual description of each of the processes within the subsystem. The user has the ability to stop and start processes as configured by the status GUI. The user can also invoke the detailed status GUI of the subsystem from the Process Status GUI. The detailed status GUI can provide information about field equipment associated with the subsystem or other information of importance.
Subsystem Heartbeat Management	Subsystem Heartbeat Management receives all the process-level heartbeat messages and maintains the current status information for the subsystem. The most severe process-level status is sent periodically to the Data Server through the subsystem's Data Server Interface.
Subsystem Process Control	Subsystem Process Control is responsible for starting and automatically restarting the processes associated with the AVI Data Processing Software.
Subsystem Status Logger	Subsystem Status Logger is the process responsible for logging status information to a log file. A log file for each day of the week is maintained. These log files are kept only for the current week.

Table 17. AVI Data Processing Software Context Diagram Data Flows

ITEM	DESCRIPTION
Equipment Status Data	Equipment Status Data are the status of the AVI Reader Field Sites.
GUIs	GUIs are graphical user interfaces. These interfaces are used to communicate information from the subsystem to the user and to allow the user to control certain aspects of the execution of the subsystem.
Link Data	Link Data are data about the AVI Links. The data includes the travel time, speed, and status of each link.
Most Severe Process Status	Most Severe Process Status is the value of the process status being managed by the Subsystem Heartbeat Management that represents the worst status of all the processes. For example if all processes indicated an ok status except one process indicated a warning status then the Most Severe Process Status would be warning.
Process Heartbeat	Process Heartbeat is the heartbeat pulse sent from each process within the subsystem. The Process Heartbeat contains the status information for the process along with the process identifier.

ITEM	DESCRIPTION
Reader Data Message	Reader Data Messages are data sent from the reader. These messages follow the proprietary protocol and format of the AVI Reader and include tag read messages as well as acknowledgement messages.
Status Log Message	Status Log Message contains information to be logged to the subsystem log file. Typical Status Log Messages include error messages such as memory allocation errors or data being logged from field equipment associated with the subsystem.
Subsystem Heartbeat	Subsystem Heartbeat is the heartbeat message containing the overall status of the AVI Data Processing subsystem. This message is generated by the Subsystem Heartbeat Management process and is passed on to the Data Server by the subsystem's Data Server Interface process.

4.2.2.1 AVI Data Processing Software

The AVI Data Processing Software data flow diagram is presented in Figure 10. This figure shows the major processes of the AVI Data Processing Software. The input and output data flows that are shown on the context diagram have been carried down to this level. The processes are described in Table 18 and the data flows are described in Table 19.

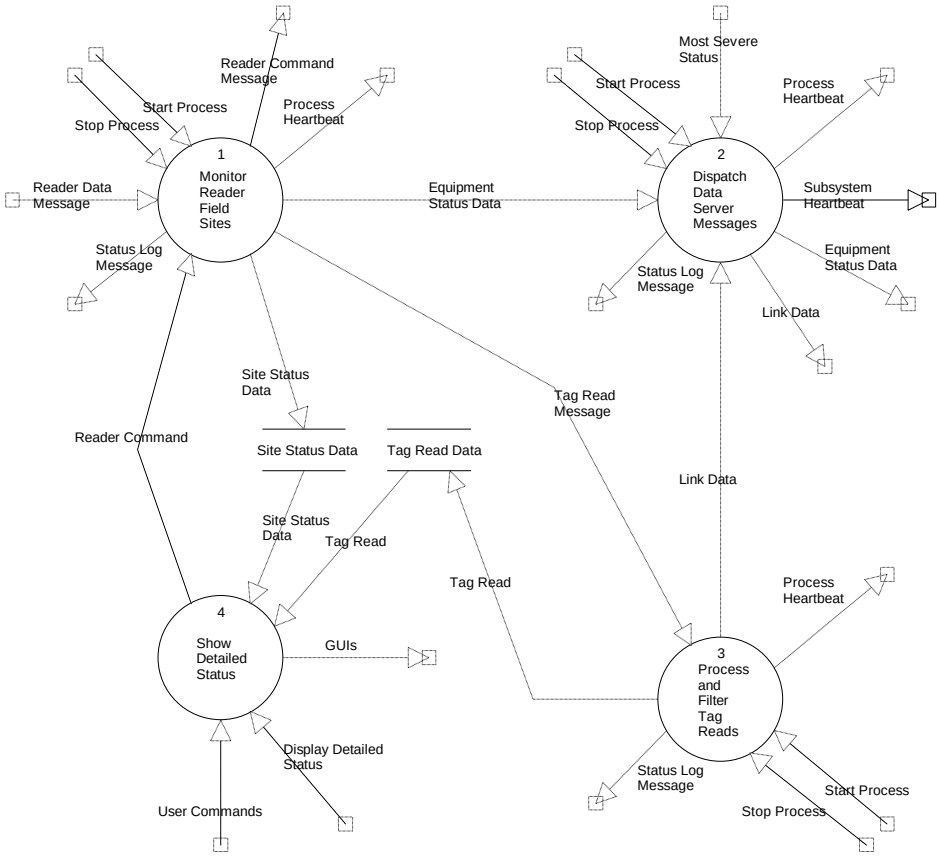


Figure 10. AVI Data Processing Software Data Flow Diagram

Table 18. AVI Data Processing Software Data Processes

ITEM	DESCRIPTION
Dispatch Data Server Messages	Dispatch Data Server Messages receives messages to be sent to the Data Server and sends these messages on to the Data Server. This process represents the subsystem's single interface point to the Data Server. This process periodically sends a heartbeat message indicating the status of the process.
Monitor Reader Field Sites	The Monitor Reader Field Sites process communicates with the AVI Readers. It collects data from the readers and forwards it to the appropriate processes and data stores. It also sends command messages to the AVI Readers.
Process and Filter Tag Reads	The Process and Filter Tag Reads process is responsible for collecting the raw tag read data, locating tag matches between source and destination AVI Reader Field Sites, and updating the AVI Link Data. The process also updates a shared repository of tag read data that is used to display tag reads to the user. A heartbeat is sent periodically indicating the status of the process.

ITEM	DESCRIPTION
Show Detailed Status	Show Detailed Status is the graphical user interface providing the TransGuide personnel with the ability to view the current status and data for the AVI Reader Field Sites and to send commands to the readers.

Table 19. AVI Data Processing Software Data Flows

ITEM	DESCRIPTION
Equipment Status Data	Equipment Status Data are the status of the AVI Reader Field Sites.
GUIs	GUIs are graphical user interfaces. These interfaces are used to communicate information from the subsystem to the user and to allow the user to control certain aspects of the execution of the subsystem.
Link Data	Link Data are data about the AVI Links. The data includes the travel time, speed, and status of each link.
Most Severe Status	Most Severe Status is the value of the process status being managed by the Subsystem Heartbeat Management that represents the worst status of all the processes. For example if all processes indicated an ok status except one process indicated a warning status then the Most Severe Status would be warning.
Process Heartbeat	Process Heartbeat is the heartbeat pulse sent from each process within the subsystem. The Process Heartbeat contains the status information for the process along with the process identifier.
Reader Data Message	Reader Data Messages are data sent from the reader. These messages follow the proprietary protocol and format of the AVI Reader and include tag read messages as well as acknowledgement messages.
Site Status Data	Data indicating the current status of reader field sites.
Status Log Message	Status Log Message contains information to be logged to the subsystem log file. Typical Status Log Messages include error messages such as memory allocation errors or data being logged from field equipment associated with the subsystem.
Subsystem Heartbeat	Subsystem Heartbeat is the heartbeat message containing the overall status of the AVI Data Processing subsystem. This message is generated by the Subsystem Heartbeat Management process and is passed on to the Data Server by the subsystem's Data Server Interface process.
Tag Read	Tag Reads are the internal representation of the tag read data sent by the AVI Reader. The data contains the time the tag was read and the tag identifier.
Tag Read Message	Tag Read Messages are tag reads that are sent by the AVI Reader. The messages follow the proprietary format and protocol of the AVI Reader and contain the time the tag was read, the AVI Reader Field Site number, and the tag identifier.

4.2.2.2 Monitor Reader Field Sites

The *Monitor Reader Field Sites* process communicates and controls the AVI Reader Field Sites. Figure 11 shows the data flow diagram for this process.

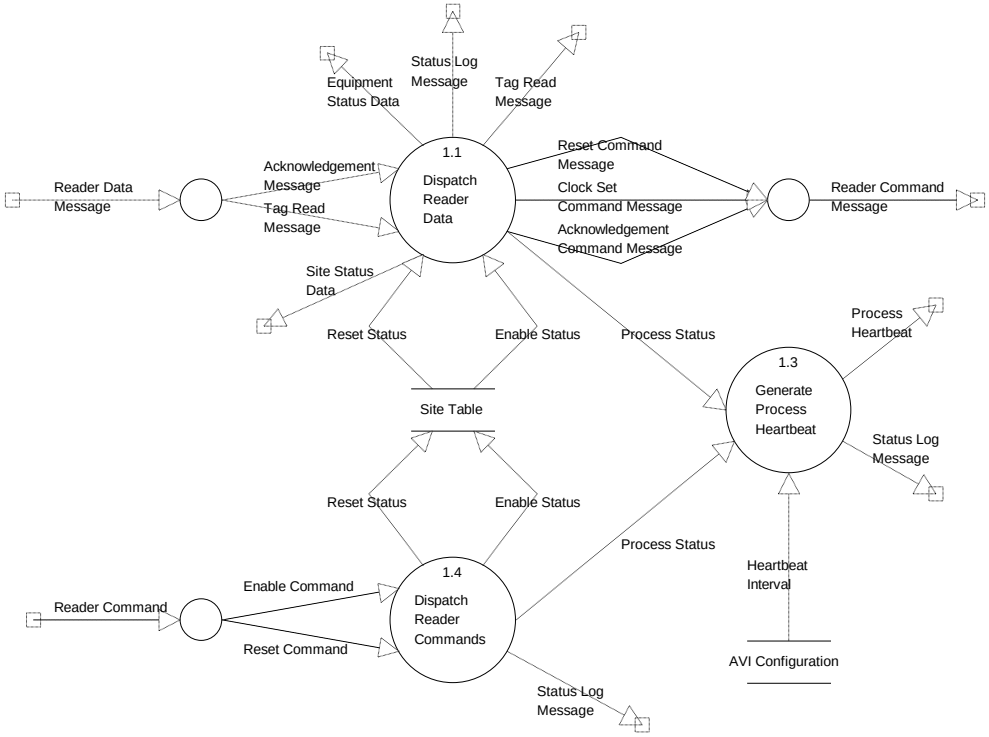


Figure 11. Monitor Reader Field Sites Data Flow Diagram

The processes shown in the data flow diagram are logical processes and are described in more detail in Table 20. The data flows are described in Table 21.

Table 20. Monitor Reader Field Sites Data Processes

ITEM	DESCRIPTION
Dispatch Reader Commands	Dispatch Reader Commands receives commands that are to be sent to the readers and updates the site table to indicate that a command should be issued to a reader.
Dispatch Reader Data	The Dispatch Reader Data process communicates with the readers. It receives messages from the readers and sends commands to the readers. The process implements the protocol defined by the Amtech reader.
Generate Process Heartbeat	Generate Process Heartbeat periodically sends the Process Heartbeat to the Subsystem Heartbeat Management process. The Heartbeat Interval determines how frequently the heartbeat is sent.

Table 21. Monitor Reader Field Sites Data Flows

ITEM	DESCRIPTION
Acknowledgement Message	Acknowledgement Messages are messages sent by the AVI Reader indicating a successful (ACK) or unsuccessful (NAK) data transmission. These message follow the proprietary protocol and format of the AVI Reader.
Enable Status	The Enable Status indicates the current disposition of an AVI Reader Field Sites enable/disable status. This status indicates whether or not data from a reader should be ignored.
Equipment Status Data	Equipment Status Data are the status of the AVI Reader Field Sites.
Heartbeat Interval	The frequency with which process heartbeats should be sent.
Process Heartbeat	Process Heartbeat is the heartbeat pulse sent from each process within the subsystem. The Process Heartbeat contains the status information for the process along with the process identifier.
Process Status	Process Status contains the current value associated with the execution status of the process. This status can indicate an OK condition, a warning condition, or an error condition.
Reader Data Message	Reader Data Messages are data sent from the reader. These messages follow the proprietary protocol and format of the AVI Reader and include tag read messages as well as acknowledgement messages.
Reset Status	The Reset Status indicates the current disposition of an AVI Reader Field Sites' reset status. This status indicates whether or not a reset command should be sent to the reader at the next opportunity.
Site Status Data	Data indicating the current status of reader field sites.
Status Log Message	Status Log Message contains information to be logged to the subsystem log file. Typical Status Log Messages include error messages such as memory allocation errors or data being logged from field equipment associated with the subsystem.
Tag Read Message	Tag Read Messages are tag reads that are sent by the AVI Reader. The messages follow the proprietary format and protocol of the AVI Reader and contain the time the tag was read, the AVI Reader Field Site number, and the tag identifier.

4.2.2.2.1 Dispatch Reader Data

The *Dispatch Reader Data* process communicates with the AVI Reader Field Sites. Several types of messages are received from the readers including tag reads and acknowledgment messages. Although there are other messages that can be sent by the reader, they are not used by the AVI Data Processing Software and are ignored. Tag reads contain data indicating a tag identifier, time, and location of a tag read. This data is forwarded to other processes. Acknowledgment messages are sent by the readers to indicate a successful transaction has occurred.

Several types of command messages are sent to the reader by the *Dispatch Reader Data* process. These include reset commands, clock set commands, and acknowledgment commands. Clock set

commands are sent periodically to update the system clock of the reader so that it remains synchronized with the rest of the system. Acknowledgment commands are sent as part of the protocol with the Amtech reader to indicate that a transaction was successfully completed. Reset commands are sent to cause the reader to perform an internal reset. The reset command is generated whenever the *Dispatch Reader Commands* process indicates in the *Site Table* that a reader should be reset.

The *Dispatch Reader Data* process also has the ability to enable or disable a reader. This function is initiated when the *Dispatch Reader Commands* process indicates in the *Site Table* that a reader should be enabled or disabled. Disabling a reader is an internal operation performed by the *Dispatch Reader Data* process that has no effect on the reader itself. It simply means that the data received by the reader will not be processed.

The *Dispatch Reader Data* process implements the protocol defined by the Amtech reader. This protocol is illustrated in the following figures. Figure 12 shows the protocol between the reader and the *Dispatch Reader Data* process when the reader sends a *Tag Read Message*.

Figure 13 shows the protocol for the *Dispatch Reader Data* process sending a message to set the clock of a reader. The protocol for the reset command is identical.

The Amtech reader uses the acknowledgment (ACK) and negative acknowledgment (NAK) to indicate the status of a transaction. If a transaction fails, the reader will attempt to execute the transaction again. Figure 14 shows the protocol for this scenario.

Finally, the Amtech reader has an upper limit on the number of retransmissions that it will attempt. After three failed attempts to send a message, the Amtech reader will stop attempting to send the message. This protocol is shown in Figure 15.

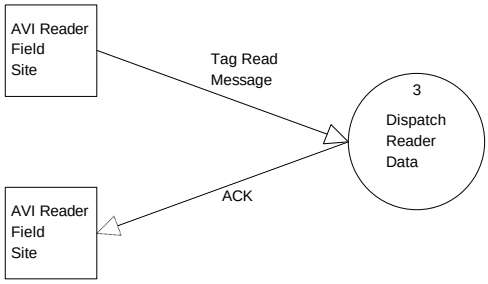


Figure 12. Tag Read Message Protocol

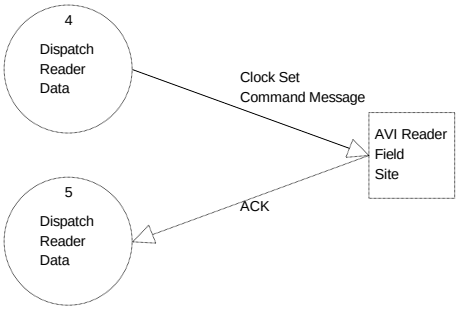


Figure 13. Clock Set Command Message Protocol

4.2.2.2.2 Dispatch Reader Commands

The *Dispatch Reader Commands* process is a logical process. It receives commands that are intended to alter the operation of the *Monitor Reader Field Sites* process. These commands include commands to reset a reader and commands to enable or disable a reader.

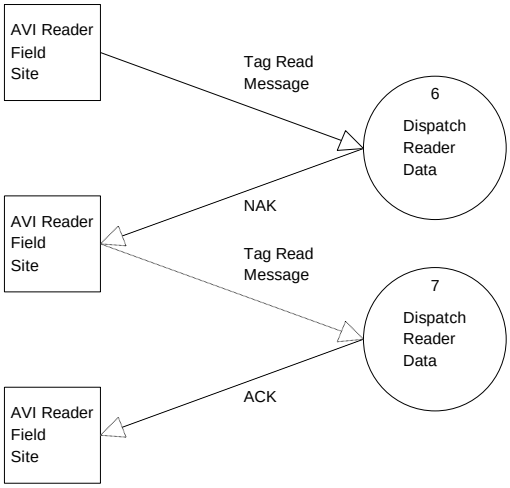


Figure 14. Retry Message Protocol

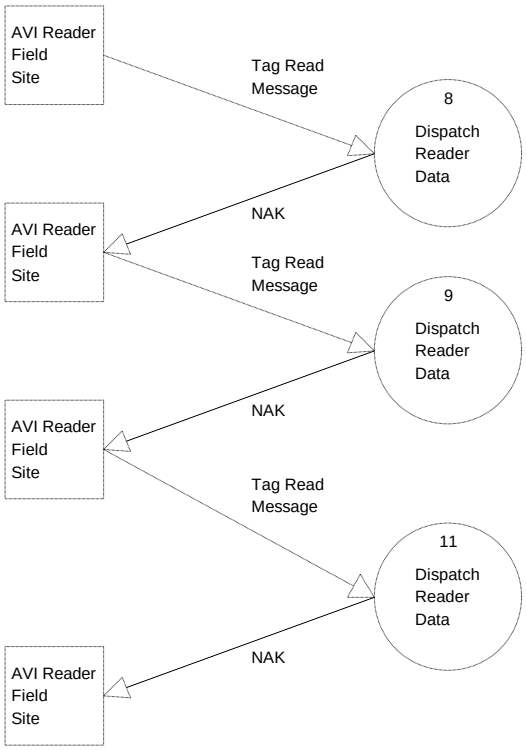


Figure 15. Failed Message Protocol

The process updates the *Site Table* based on the commands that it receives. The *Dispatch Reader Data* process accesses the S*ite Table* to determine when a command should be sent to a reader.

4.2.2.3 Process and Filter Tag Reads

The *Process and Filter Tag Reads* process transforms tag reads into link data. This is accomplished by searching the list of current tag reads to find a matching tag at an adjacent reader site. The process updates the link data by keeping a rolling average of travel time and travel speed for each link. The process also archives various types of data and generates a heartbeat and log messages. Figure 16 shows the data flow diagram for the *Process and Filter Tag Reads* process.

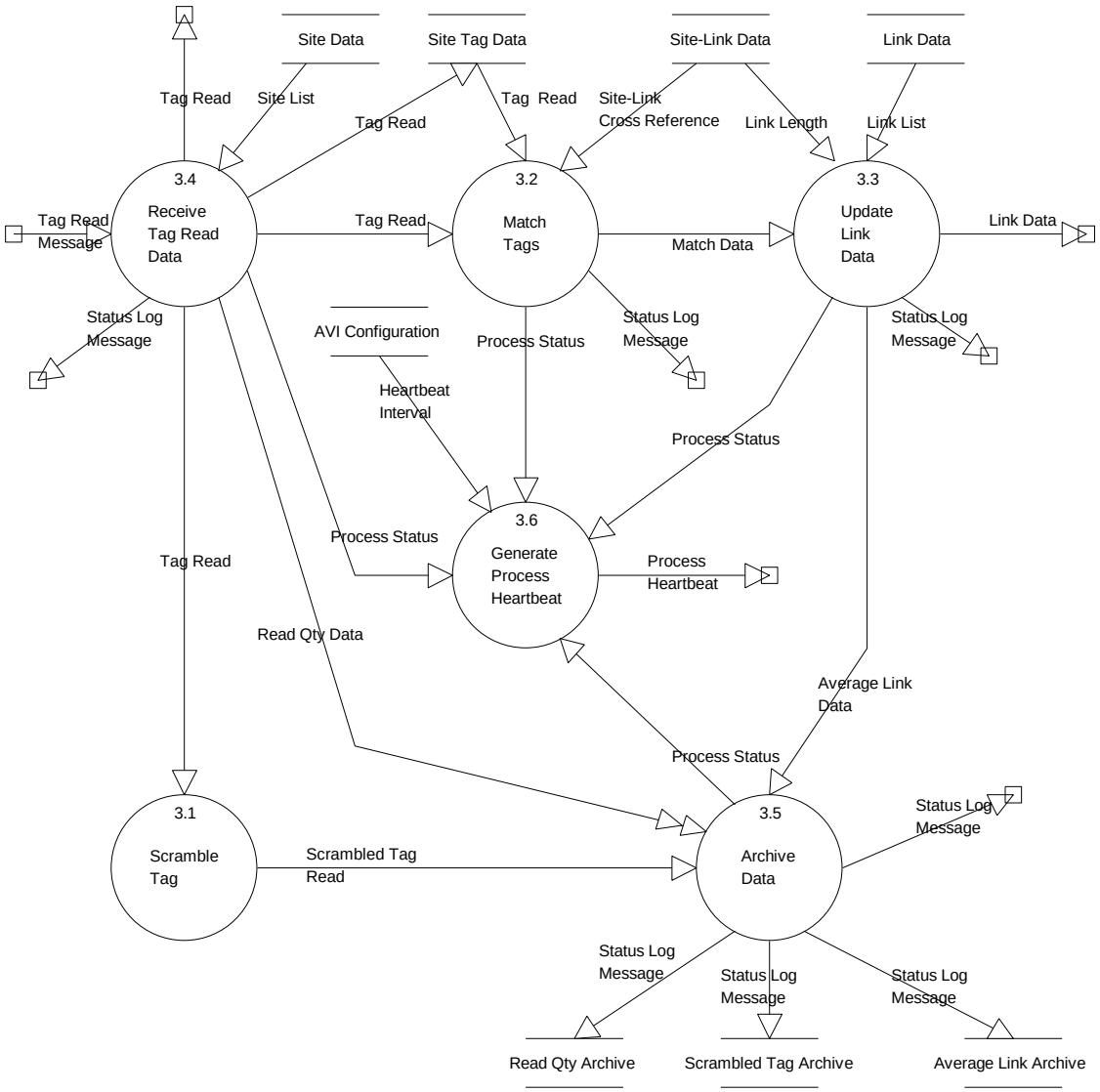


Figure 16. Process and Filter Tag Reads Data Flow Diagram

The logical processes and data flows of the *Process and Filter Tag Reads* data flow diagram are described in Table 22 and Table 23.

Table 22. Process and Filter Tag Reads Data Processes

ITEM	DESCRIPTION
Archive Data	The Archive Data process receives archive data and maintains archive files. A new archive file is created each day. The archive file for a given day is overwritten each week.
Generate Process Heartbeat	Generate Process Heartbeat periodically sends the Process Heartbeat to the Subsystem Heartbeat Management process. The Heartbeat Interval determines how frequently the heartbeat is sent.

ITEM	DESCRIPTION
Match Tags	Locates tag read matches between adjacent reader field sites. The process maintains a history of tag reads for each reader. When a new tag arrives, the process searches adjacent readers looking for a tag match. If it finds one, it records the match so that the travel time can be calculated.
Receive Tag Read Data	Receives tag read data packets from the Data Communication and Monitoring process and keeps track of the number of reads that have been received from each reader field site. This data is periodically logged to a data file.
Scramble Tag	Scramble Tag is responsible for encrypting tag identifiers to ensure the privacy of the users of the AVI System.
Update Link Data	

Table 23. Process and Filter Tag Reads Data Flows

ITEM	DESCRIPTION
Average Link Data	Average Link Data contains the average travel time and travel speed for each link for a specified period of time. The period of time for which the average is calculated is a configuration item.
Heartbeat Interval	The frequency with which process heartbeats should be sent.
Link Data	Link Data are data about the AVI Links. The data includes the travel time, speed, and status of each link.
Link Length	The Link Length is the length of a link in feet.
Link List	
Match Data	Match Data is data related to a tag match. The data contains the two timestamps representing the time that the tag passed the source and destination readers respectively as well as the link(s) that are defined between the two readers.
Process Heartbeat	Process Heartbeat is the heartbeat pulse sent from each process within the subsystem. The Process Heartbeat contains the status information for the process along with the process identifier.
Process Status	Process Status contains the current value associated with the execution status of the process. This status can indicate an OK condition, a warning condition, or an error condition.
Read Qty Data	A count of the number of reads over the previous time period for each reader.
Scrambled Tag Read	The Scrambled Tag Read is a tag read which has had its tag identifier encrypted.
Site List	A list of the sites that are defined in the AVI System.
Site-Link Cross Reference	Site-Link Cross Reference is an association between a source site identifier and a destination site identifier.
Status Log Message	Status Log Message contains information to be logged to the subsystem log file. Typical Status Log Messages include error messages such as memory allocation errors or data being logged from field equipment associated with the subsystem.

ITEM	DESCRIPTION
Tag Read	Tag Reads are the internal representation of the tag read data sent by the AVI Reader. The data contains the time the tag was read and the tag identifier.
Tag Read Message	Tag Read Messages are tag reads that are sent by the AVI Reader. The messages follow the proprietary format and protocol of the AVI Reader and contain the time the tag was read, the AVI Reader Field Site number, and the tag identifier.

4.2.2.3.1 Match Tags

The *Match Tags* logical process accepts tag reads and produces match data. The tag read contains a TransGuide AVI Site Identifier, which the process assumes is the destination site of a potential source-destination pair. The process uses the site-link table to locate potential source sites for the destination. If potential source sites exist, the process determines if one or more of the potential source sites have received the tag identifier. If the tag identifier is found, a match is created.

4.2.2.3.2 Update Link Data

The *Update Link Data* process is a logical process that represents the transformation of match data into link data. The link data represents a *rolling average* of tag matches over a pre-defined period of time. This section describes the algorithm that is used to calculate the rolling average travel time and gives some examples of the calculation.

4.2.2.3.2.1 Rolling Average Algorithm

Let l be a link between two reader field sites with attributes l_{th}, the link threshold for link l, l_{tt}, the current rolling average travel time for link l, and l_s, the current rolling average speed for link l. Let t_c be the current time and t_w be the duration of the rolling average window in seconds.

Let Ml be the set of all tag matches, m, received for link l. Let each match m have the following attributes: m_t, the time that match m was received, m_{tt}, the travel time measured by match m, and m_s, the speed measured by match m.

Let the current set of valid tag matches, Ml', be defined by the following:

$$Ml' = \{m \mid t_c - t_w < m_t < t_c \text{ and } (l_{tt} - l_{tt} \times l_{th}) < m_{tt} < (l_{tt} + l_{tt} \times l_{th})\}$$

The rolling average travel time for link *l*, l_{tt} ¢and the rolling average speed, l_s ¢can be calculated by the following equations:

$$l_{tt'} = \frac{\sum_{m=1}^{|Ml'|} m_{tt}}{|Ml'|}$$

$$l_{s'} = \frac{\sum_{m=1}^{|Ml'|} m_s}{|Ml'|}$$

4.2.2.3.2.2 Rolling Average Examples

A *tag match* is produced whenever a tag is received by an adjacent pair of reader sites that are defined as a source-destination pair in the configuration file. Tag matches are stored in a table along with the time that they were received and the travel time and speed measured for the tag as it passed between the two readers. Table 24 shows an example of a table for link identifier IN0035I-RANDO-WALZE, which is one (1) mile in length.

Table 24. Tag Match Table for link IN0035I-RANDO-WALZE

MATCH NUMBER	TAG IDENTIFIER	TIME	TRAVEL TIME (SEC)	SPEED (MPH)
1	189	10:05:00	65	55
2	94	10:05:02	54	53
3	256	10:05:10	66	40
4	343	10:05:11	64	59
5	984	10:05:19	62	68
6	598	10:05:25	56	67
7	501	10:05:28	60	62
8	402	10:05:35	69	67
9	639	10:05:39	59	55
10	54	10:05:42	68	56
11	798	10:05:44	90	58
12	603	10:05:52	61	64
13	402	10:06:03	53	60
14	609	10:06:09	54	52
15	799	10:06:20	58	61

Periodically, the average travel time and speed for each link are calculated using a rolling-average algorithm. Two parameters are used to control the function of this algorithm: the *rolling-average window* and the *link threshold*.

The rolling-average window is used to determine the period of time that should be considered when calculating the rolling-average. For example, if the rolling-average window is 60 seconds, only the tag matches received in the past 60 seconds will be used to calculate the rolling average. The

average that is produced from this calculation would be referred to as a *60-second rolling-average*.

The second parameter, the link threshold, is used to filter tag matches. For example, if the link threshold is 20%, any match which is 20% greater or less than the previous rolling-average travel time will not be included in the calculation.

The rolling-average is calculated by taking the sum of all tag matches which were recorded within the past rolling-average window and which have travel time values that fall within the link threshold. This sum is then divided by the number of matches that were included in the sum.

Example 1

Assume that the current time is 10:05:20, the tag match table for link IN0035I-RANDO-WALZE is as presented in Table 24, and the previous calculation for the rolling-average travel time was 59 seconds. Using a rolling-average window of 20 seconds and a threshold of 20%, all matches received since 10:05:00 with a travel time between 47.2 seconds and 70.8 seconds will be considered. This produces the following equation:

$$Ml = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$
$$Ml' = \{1, 2, 3, 4, 5\}$$
$$|Ml'| = 5$$
$$l_{tt} = \frac{\sum_{m=1}^{5} m_{tt}}{5} = 62.2$$

Equation 1. Calculation of Rolling-Average Travel Time for Example 1

The link rolling-average speed, l_s, can be calculated in a similar fashion by replacing the match travel time, m_{tt}, with the match speed, m_s. This would produce a rolling-average speed of 58.2 mph.

Example 2

Assume that the current time is 10:06:10, the tag match table for link IN0035I-RANDO-WALZE is as presented in Table 24, and the previous calculation for the rolling-average travel time was 62 seconds. Using a rolling-average window of 30 seconds and a threshold of 20%, all matches received since 10:05:40 with a travel time between 49.6 seconds and 74.4 seconds will be considered. This produces the following calculation:

$$Ml = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$
$$Ml' = \{10, 12, 13, 14\}$$
$$|Ml'| = 4$$
$$l_{tt} = \frac{\sum_{m=1}^{4} m_{tt}}{4} = 59.0$$

Equation 2. Calculation of Rolling-Average Travel Time for Example 2

Note that in Equation 2, tag match number 11 is not included in the current match set, $Ml \notin$ because its travel time value, 90, falls outside the valid range of travel times. The link rolling-average speed, l_s, can be calculated in a similar fashion by replacing the match travel time, m_{tt}, with the match speed, m_s. This would produce a rolling-average speed of 61.75 MPH.

4.2.2.3.3 Scramble Tag

The *Scramble Tag* process is a logical process that transforms tag read data into scrambled, or encrypted, tag read data. The tag data is encrypted to ensure the privacy of the users of the AVI System.

The tag identifier is scrambled using the UNIX *crypt* function. This function uses a one-way encryption algorithm to uniquely encrypt a text string. The function accepts the string that is to be encrypted and a *salt* value, which is used to alter the encryption. The encryption algorithm is deterministic in that it will always produce the same encrypted string for a given input string and salt value.

To reduce the possibility that the encrypted tag identifiers could be decrypted, two steps are taken. First, the salt value is changed on a daily basis. As a result, the same tag identifier will have a different encrypted value from one day to the next. Second, the salt value is assigned based on the current value of the system clock. Although the salt value is set shortly after midnight each morning, the system clock is measured in total seconds since January 1, 1970. As a result, the salt value is rarely the same from day to day and the actual value that was used cannot be determined.

4.2.2.3.4 Archive Data

The *Archive Data* process is responsible for archiving various types of data. Each archive is stored separately and the archives are organized by day. The archives are valid for one week, at which time they are overwritten to make room for the new archive. Table 24 describes each of the archives.

Table 25. AVI Data Processing Software Archives

ARCHIVE NAME	DESCRIPTION
Average Link	For each link in the AVI System, the average travel time and travel speed are periodically archived.
Number of Reads	For each AVI Reader Field Site in the AVI System, the total number of tag reads received from the site is periodically archived.
Tag	For each AVI Reader Field Site in the AVI System, the total number of tag reads received from the site is periodically archived.

4.2.2.4 Dispatch Data Server Messages

The *Dispatch Data Server Messages* process is the interface between the AVI Data Processing Software and the MDI Data Server. This process handles requests to write link data, heartbeat, and equipment status messages to the Data Server.

4.2.2.5 Show Detailed Status

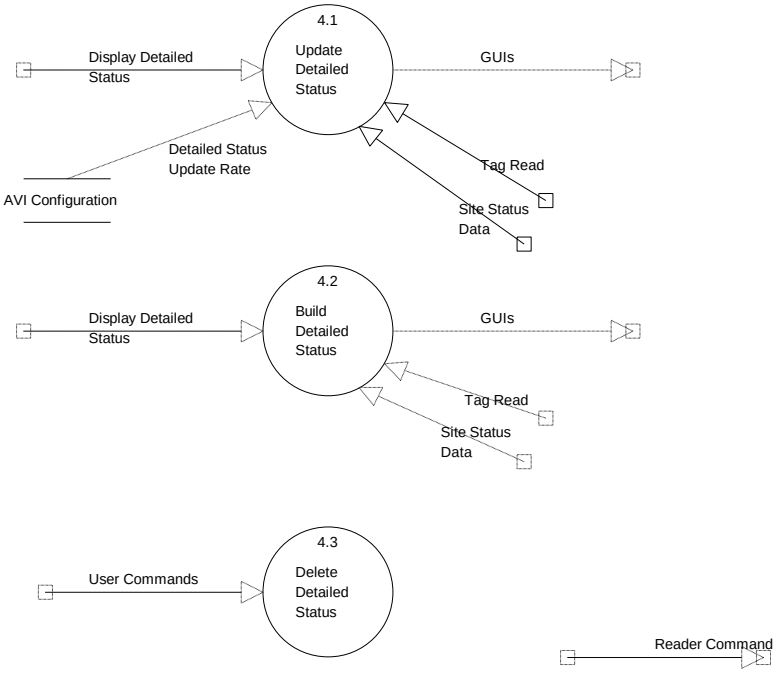


Figure 17. Show Detailed Status Data Flow Diagram

4.2.3 Structure Chart Design

The structure chart design is presented in a series of structure chart diagrams. These diagrams use several symbols to represent functions and library functions. A legend of symbols is presented in Figure 18.

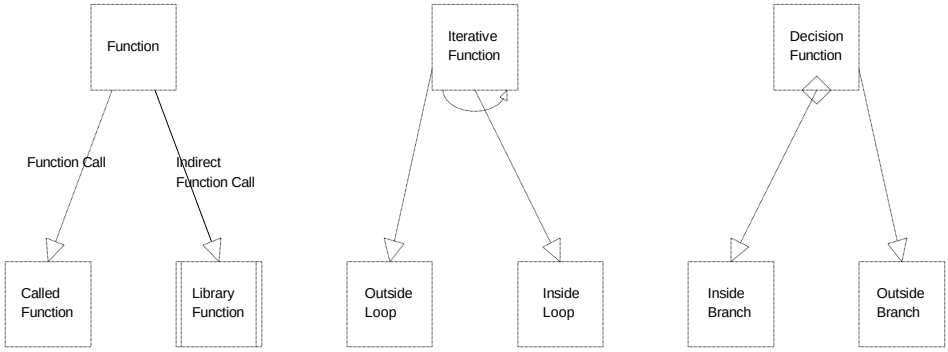


Figure 18. Structure Chart Legend

The structure charts define the functions that are called by each function. The functions that are called are listed once per chart, even if the function is called more than once. The symbols presented in the legend are described in Table 26.

Table 26. Structure Chart Legend

SYMBOL	DESCRIPTION
Decision Function	A function that contains a significant decision or branch. The functions that are called inside the branch are attached to the bottom of the function. The functions that are called outside the branch are attached to the side of the function.
Function	A source code function. Each function will be shown only once per chart, even though it may be called more than once.
Function Call	Indicates that one function calls another.
Indirect Function Call	A call that is not made directly but occurs indirectly. For example, a function that is called as a result of a signal.
Iterative Function	A function that contains an iterative loop. The functions that are called inside the loop are attached to the bottom of the function. The functions that are called outside the loop are attached to the side of the function.
Library Function	A function that is part of an external library of source code.

The AVI Data Processing Software consists of the following processes:

- AVI Master - starts and monitors the processes of the AVI Data Processing System,
- AVI Status GUI - provides an interface for the user to monitor the status of the AVI Data Processing System and to manually control the execution of the system,
- AVI Detailed Status GUI - provides an interface for the user to monitor the status of the AVI field equipment,
- AVI Heartbeat - collects and monitors heartbeats from the processes within the AVI Data Processing System,
- AVI Status Logger - provides a mechanism to collect and log status information from the AVI Data Processing System processes,
- AVI Data Server Interface - provides an interface between the AVI Data Processing System and the MDI DataServer,
- AVI Data Collection and Monitoring - communicates with the AVI readers, and
- AVI Data Processing and Filtering - processing tag reads and determines the current conditions of the AVI links.

Like other parts of the MDI system, the Master, Heartbeat, Status GUI, and Status Logger processes are instances of common programs. The Data Server Interface (DSIF), Data Collection and Monitoring (DCM), Data Processing and Filtering (DPF), and Detailed Status GUI programs are unique to AVI. The remaining programs are MDI common programs that are customized for each subsystem through the use of configuration files. They are described in other MDI documents. The programs that are unique to AVI are described in detail in the following sections. In addition, several libraries that are unique to the AVI system are described.

Each function that makes up the AVI Data Processing Software is described. Three components are included: textual description, structure chart, and a table of called functions (including macros). AVI project, MDI project, and system calls are included in the charts; however, some functions are considered so basic that they add little value to the charts. These functions are the *printf* family, *mem*xxx family, *sizeof*, and *exit*, and they are not included in the charts, tables, or descriptions.

4.2.3.1 Global Data Structures

There are several data structures that are maintained globally so that they may be accessed by more than one of the AVI Data Processing System processes. These are discussed in more detail in the following sections.

4.2.3.1.1 Site Status Table

The site status table is an array of data structures located in shared memory. There is one data structure for each site in the AVI System, which contains current information about the status of each site. This table and the data that is included in each structure is depicted graphically in Figure 19.

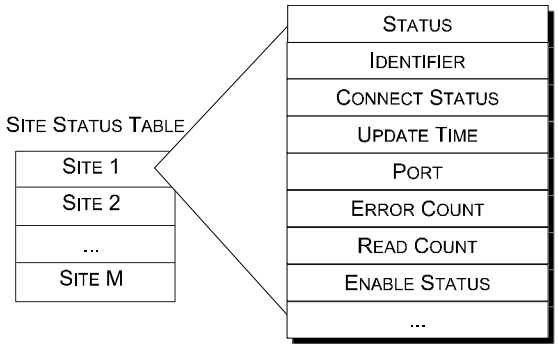


Figure 19. Site Status Table Data Structure

4.2.3.1.2 Site Tag Data Table

The site tag data table is located in shared memory and contains the most recent tag reads for each of the sites. The table and its fields are depicted graphically in Figure 20. The site tag data table is a linear table containing one block of data for each site in the AVI System. Each block contains a fixed number of slots for tag data. The number of slots available is configurable at runtime. Each entry in the table is a data structure containing a tag identifier and the time that the tag was read.

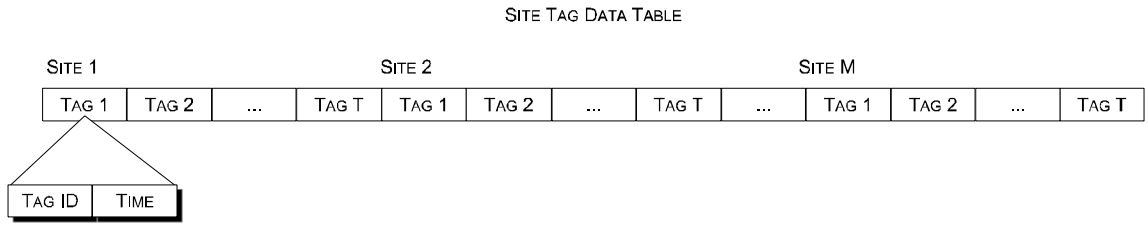


Figure 20. Site Tag Data Table Data Structure

4.2.3.2 Data Collection and Monitoring (DCM)

The Data Collection and Monitoring (DCM) process controls the modem data pool to accept data from the reader field sites. The data is sent to the DPF process. DCM handles such details as modem initialization, connection, and error handling. It also maintains the status of the field units and sends them time updates in order to keep the entire system synchronized. The following sections describe the data structures that are specific to the DCM process and the functions that

make up the DCM process, which are listed in depth-first order beginning with the main entry point.

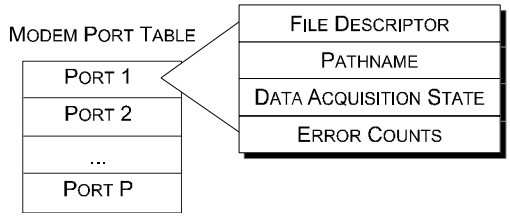


Figure 21. Modem Port Table Data Structure

4.2.3.2.1 Data Structures

This section describes the primary data structure used by DCM, the modem port table. This table is depicted graphically in Figure 21.

There is one entry in the table for each modem port in the system. Each entry contains several fields including the file descriptor of the port, the pathname, the current state of the port, and a count of errors associated with the port.

4.2.3.2.2 main

The main routine is responsible for setting up the clean up routines, configuring the appropriate signals to catch and ignore, etc. Virtually all the actual initialization operations occur in subroutines that are detailed later in the document. The structure chart for the main routine is shown in Figure 22. Descriptions of the routines called by the main routine of DCM are provided in Table 27.

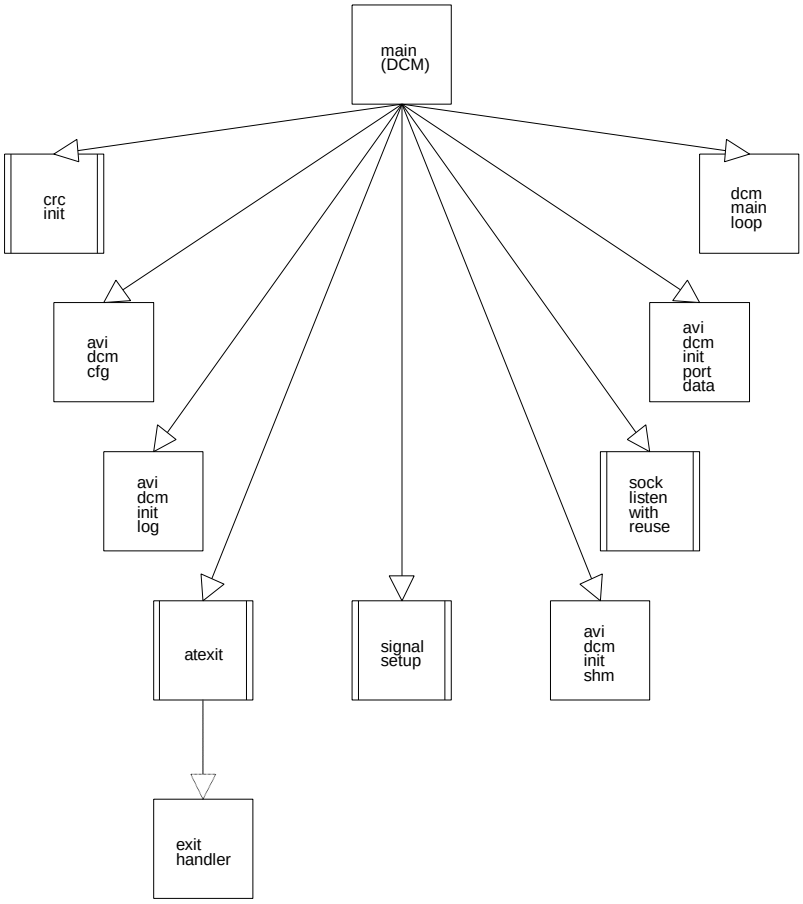


Figure 22. AVI DSIF *main* structure chart

Table 27. Routines called by AVI DCM *main*

ITEM	DESCRIPTION
atexit	C Library Function used to register routines to be called on normal termination of a program.
avi dcm cfg	Load MDI and AVI configuration data and AVI specific data.
avi dcm init log	Initialize status logging for DCM.
avi dcm init port data	Allocate and initialize the DCM physical modem port data.
avi dcm init shm	Attach/create shared memory used by DCM.
crc init	This function builds tables for the CCITT and CRC-16 CRCs and their reverses.
dcm main loop	This function polls each enabled port in turn, processing received data. It also performs other periodic update functions including heartbeat, shared memory update and configuration change processing.
exit handler	Perform exit operations.
main (DCM)	This is the main entry point for the Data Collection and Monitoring process.

ITEM	DESCRIPTION
signal setup	This function sets up the signal handler for all signals that are not currently handled within the calling process.
sock listen with reuse	MDI Common Socket routine used to set up a socket to listen for connections and to make the socket address reusable.

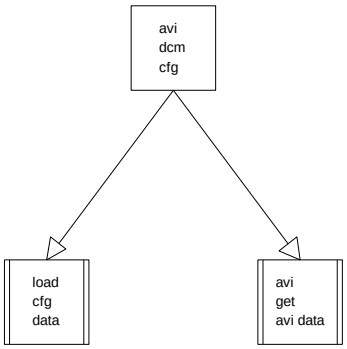


Figure 23. *avi dcm cfg* structure chart

4.2.3.2.3 avi dcm cfg

avi dcm cfg loads the AVI configuration data and AVI specific data files. It is passed the MDI and AVI configuration pathnames that are passed to the AVI configuration utility function *load cfg data*. The AVI site data and site/link cross-reference data files are loaded using the *avi get avi data* utility funciton. The structure chart for *avi dcm cfg* is shown in Figure 23. Descriptions of the routines called by *avi dcm cfg* are provided in Table 28.

Table 28. Routines called by *avi dcm cfg*

ITEM	DESCRIPTION
avi dcm cfg	Load MDI and AVI configuration data and AVI specific data.
avi get avi data	This function loads the AVI link and site data from files into dynamically allocated data structures.
load cfg data	This function loads the MDI and AVI configuration data. Data is obtained from configuration files and system function calls.

4.2.3.2.4 avi dcm init log

This routine determines from the configuration data on which host the AVI status logger is running. Once the host name is determined, the routine connects to the AVI status logger process. The structure chart for *avi dcm init log* is shown in Figure 24. Descriptions of the routines called by *avi dcm init log* are provided in Table 29.

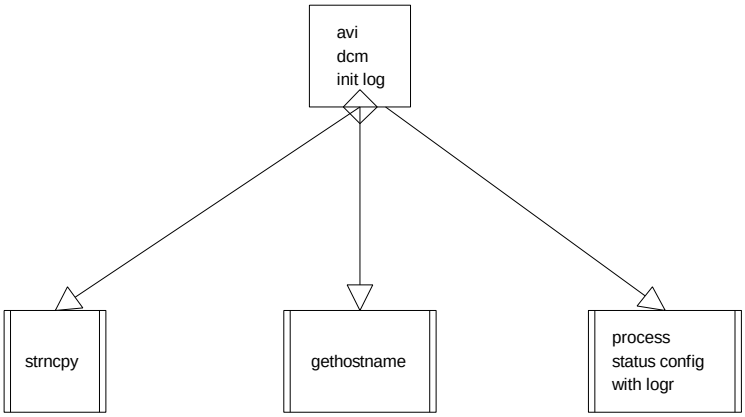


Figure 24. *avi dcm init log* structure chart

Table 29. Routines called by *avi dcm init log*

ITEM	DESCRIPTION
avi dcm init log	Initialize status logging for DCM.
gethostname	C library function to get the hostname on which the calling process is running.
process status config with logr	This routine is sets up the connection to the status logger used by the calling program.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.

4.2.3.2.5 exit handler

This routine performs exit functions for DCM. It is registered with the system using the system *atexit* function and is called automatically when a program exit condition occurs. The function logs a message to the status log indicating that DCM exited. The *exit handler* routine calls only *process status message* routine, so no structure chart is provided. Descriptions of the routines called by *exit handler* are provided in Table 30.

Table 30. Routines called by *exit handler*

ITEM	DESCRIPTION
exit handler	Perform exit operations.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

4.2.3.2.6 avi dcm init shm

DCM maintains the status data in shared memory for DPF and the Detailed Status GUI. DPF uses the information to mark the link status in the link speed and time messages sent to Data Server. The Detailed Status GUI uses the information to update the user interface. This function first creates or attaches to the shared memory depending on whether or not it already exists. Next, it allocates a copy of the status data that is global to DCM. The DCM copy will be periodically updated and written to the shared memory. The structure chart for *avi dcm init shm* is shown in Figure 25. Descriptions of the routines called by *avi dcm init shm* are provided in Table 31.

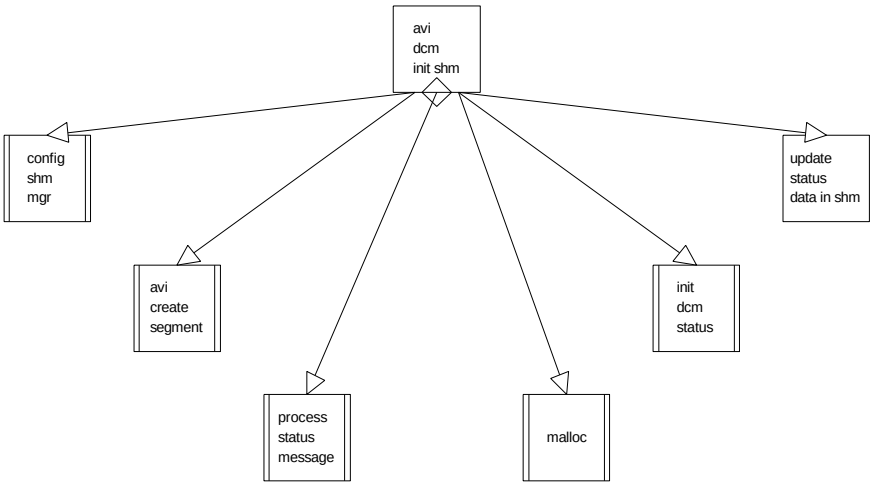


Figure 25. *avi dcm init shm* structure chart

Table 31. Routines called by *avi dcm init shm*

ITEM	DESCRIPTION
avi create segment	This function connects to or creates a shared memory segment with error checking.
avi dcm init shm	Attach/create shared memory used by DCM.
config shm mgr	MDI Shared Memory Manager routine used to initialize and configure the shared memory manager library routines for the calling program.
init dcm status	Initialize the passed DCM status structure.
malloc	C library function to allocate memory on the heap.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
update status data in shm	Update global site status information in shared memory.

4.2.3.2.7 update status data in shm

This function writes the site status data maintained by DCM in a global array to the status data shared memory using the *write segment* utility. The structure chart for *update status data in shm* is shown in Figure 26. Descriptions of the routines called by *update status data in shm* are provided in Table 32.

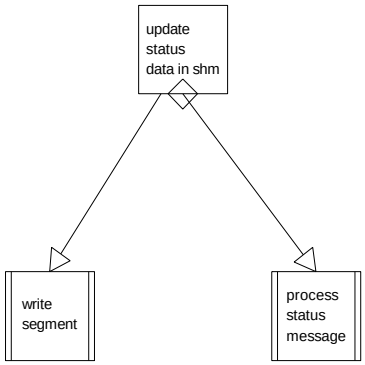


Figure 26. *update status data in shm* structure chart

Table 32. Routines called by *update status data in shm*

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
update status data in shm	Update global site status information in shared memory.
write segment	This function writes the passed data into the identified shared memory segment.

4.2.3.2.8 avi dcm init port data

This routine allocates a global data structure for the port data then loads it from file. It begins by building the pathname and reading the number of entries in the file in order to size the data structure. Next, it allocates it. Then it calls *dcm read modem data* to read the data from the file into the data structure. The structure chart for *avi dcm init port data* is shown in Figure 27. Descriptions of the routines called by *avi dcm init port data* are provided in Table 33.

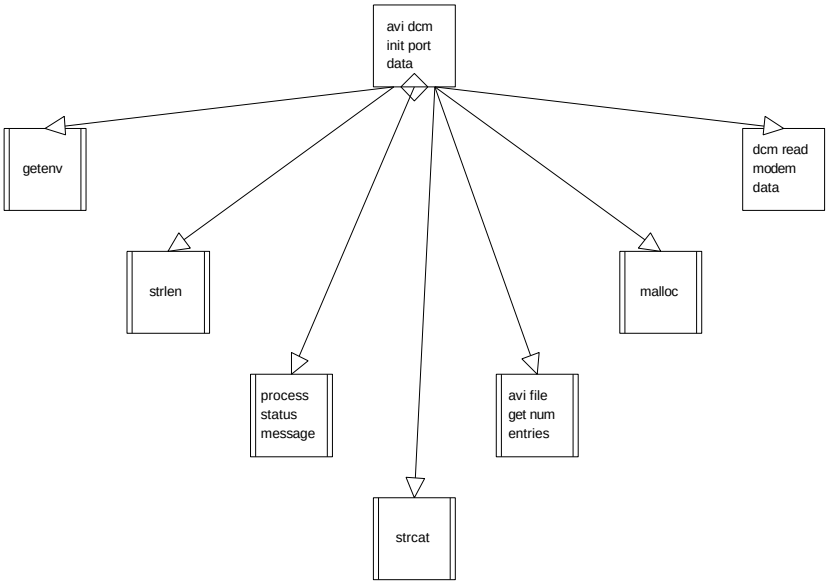


Figure 27. *avi dcm init port data* structure chart

Table 33. Routines called by *avi dcm init port data*

ITEM	DESCRIPTION
avi dcm init port data	Allocate and initialize the DCM physical modem port data.
avi file get num entries	Get the number of entries in a configuration file from the first record of the file.
dcm read modem data	Read the modem data file and initialize the affected global modem port data control structure fields appropriately.
getenv	C library function which returns a pointer to the value of the passed environment variable.
malloc	C library function to allocate memory on the heap.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
strcat	C library function which concatenates a copy of the second argument to the first.
strlen	C library function to return the length of the passed string.

4.2.3.2.9 dcm read modem data

This routine reads the modem data file to get the configuration information for the modems. The file is opened and the first record is skipped. The first record contains the count of remaining records in the file and was previously used to size the array of structures that was passed to the this routine to be initialized. Each line of the data file is read until the number of records equivalent to the number of ports is reached. Each line contains an enable field ("enable" or "disable") and the port pathname (for example, "/dev/sts/ttyD40"). Each entry in the data structure is initialized by

calling *init modem data* passing the enable flag and pathname for the port. Errors are detected and reported and cause file processing to stop. The structure chart for *dcm read modem data* is shown in Figure 28. Descriptions of the routines called by *dcm read modem data* are provided in Table 34.

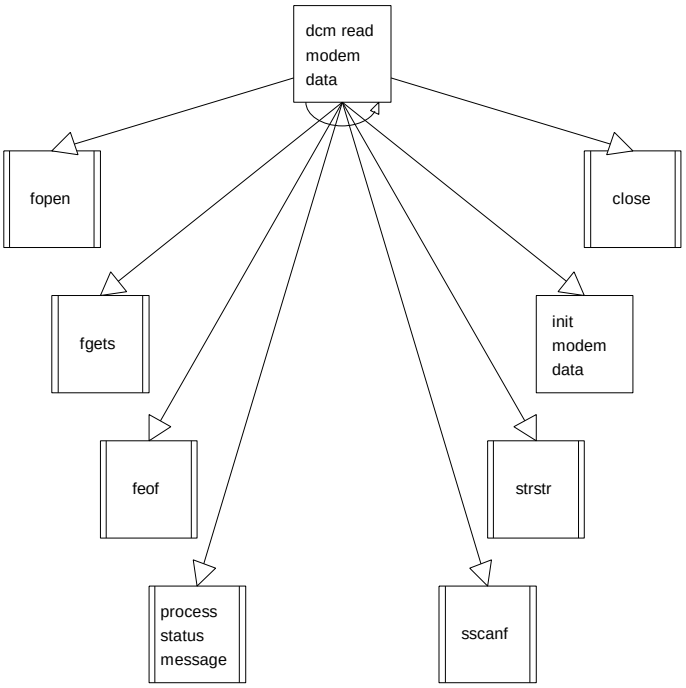


Figure 28. *dcm read modem data* structure chart

Table 34. Routines called by *dcm read modem data*

ITEM	DESCRIPTION
close	C library function to close a file or device.
dcm read modem data	Read the modem data file and initialize the affected global modem port data control structure fields appropriately.
feof	C library function to check for end of file on a stream.
fgets	C library function to read a string from an input stream.
fopen	C library function to open a stream file.
init modem data	Initialize the passed modem data structure.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
sscanf	C library function to scan a string into the referenced variables according to the specified format string.
strstr	Locate the first occurrence of the second string in the first string.

4.2.3.2.10 init modem data

This function initializes a port data structure. The fields of the structure are initialized with the option of preserving the port pathname, file descriptor, and enabled and opened flags. *init modem data* calls only *strcpy* so no structure chart is provided. Descriptions of the routines called by *init modem data* are provided in Table 35.

Table 35. Routines called by *init modem data*

ITEM	DESCRIPTION
init modem data	Initialize the passed modem data structure.
strcpy	C Library Function used to copy a source string to a destination string.

4.2.3.2.11 dcm main loop

This function performs the main operational processing for DCM. Once started, it operates continuously until terminated by external means. The routine begins by doing some initialization of the socket descriptor sets used in later *select* calls. Next, it enters a while loop when only exits when the program is terminated. The loop consists of two parts: an inner loop that processes data for each modem port and a section of code that performs periodic updates (for example, send heartbeat).

The inner loop processes each physical port in turn. First, the port enable is checked. If disabled, no further processing occurs. Otherwise, if the port is in the connected state, then data received on the port is processed. If the port is not connected, then modem initialization and reconnection are performed. Each time through the loop, the configuration change socket is checked for input from the user via the AVI GUI. The user may enable or disable a site or reset a site. These actions may occur out-of-sequence when *handle config changes* is called.

Each time through the outer loop the DCM copy of the site status data is updated, then the shared memory status is updated from DCM's internal status. If the time-out has occurred to send the next heartbeat, then the summary status sent with the heartbeat is updated, the heartbeat sent, and the heartbeat timer reinitialized.

The structure chart for *dcm main loop* is shown in Figure 29. Descriptions of the routines called by *dcm main loop* are provided in Table 36.

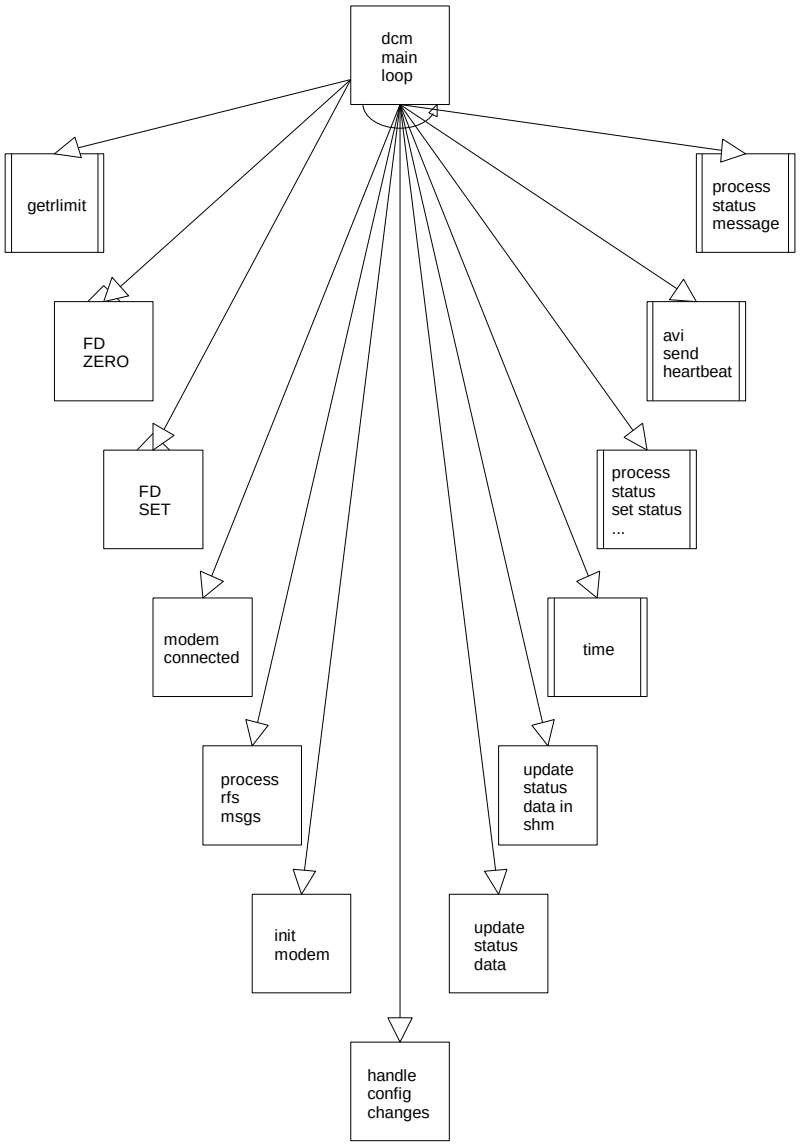


Figure 29. *dcm main loop* structure chart

Table 36. Routines called by *dcm main loop*

ITEM	DESCRIPTION
avi send heartbeat	This function sends the heartbeat to the AVI heartbeat process with automatic reconnection and error checking.
dcm main loop	This function polls each enabled port in turn, processing received data. It also performs other periodic update functions including heartbeat, shared memory update and configuration change processing.
FD SET	C library macro to set a file descriptor in a file descriptor set.

ITEM	DESCRIPTION
FD ZERO	C library macro to zero a file selector set used with select().
getrlimit	C library function to get the specified system limit from the system.
handle config changes	Handle configuration changes which arrive on the configuration change socket.
init modem	Manage the state of a modem from initialization through a connection.
modem connected	Check the serial port status lines to see if the modem on the passed file descriptor is still connected.
process rfs msgs	This function performs data collection and monitoring for one modem port.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
time	C library function to get the current system time in Unix format.
update status data	Update global site status information.
update status data in shm	Update global site status information in shared memory.

4.2.3.2.12 modem connected

This function uses an *ioctl* call to check the status (connected or not connected) of the passed modem port. *modem connected* calls only *ioctl*, so no structure chart is provided. Descriptions of the routines called by *modem connected* are provided in Table 37.

Table 37. Routines called by *modem connected*

ITEM	DESCRIPTION
ioctl	C library function to perform I/O control operations on devices and streams.
modem connected	Check the serial port status lines to see if the modem on the passed file descriptor is still connected.

4.2.3.2.13 process rfs messages

This routine processes input from the reader field sites for one port. Actions are not performed on the port if it is undefined (the file descriptor is invalid). First, the routine checks to see if it is time to update the clock for the RFS. Second, data is read from the port. A loop is entered where the received data is processed. Once all data in the input stream has been processed, the loop exits. Third, the port is reset if the reset command has been received from the AVI GUI. Fourth, the port status is changed "warning" if the inactivity time-out has expired.

For each pass of the loop, the received data is passed to *confirm msg*, which examines the data to determine if a message has been received. Its processing is significantly controlled by the status of the message being accumulated that is returned by *confirm msg*. The message status may be one of the following:

- incomplete - the input data does not make up a message
- incomplete with error - the input data has an error but does not form a complete message
- complete with error - the input data is a complete message, but has an error
- complete - a complete message without any errors was received from the RFS

A status of incomplete indicates that the message, which may be received through multiple calls to *process rfs messages*, is incomplete or is not present in the data at all. Incomplete with error indicates that the start of message code was received and processed, but the CRC for the header failed. Complete with error indicates that the message was received but had an error in the CRC for the body of the message. Finally, complete indicates that a complete message was received without errors.

A status value of *incomplete* results in no action in *process rfs messages*. A status value of *incomplete with error* causes the error status for the site to be updated, if the site index is valid. A status value of *complete with error* causes a negative acknowledge (NAK) to be sent to the RFS. A status value of *complete* causes the message to be processed. If this is the first valid message from the RFS since the last reconnection, then the site status data is updated. *eval msg* is called to process the message further. Finally, regardless of status returned by *confirm msg*, the loop is repeated until all input data is consumed by *confirm msg*. This means that multiple messages may be processed while within this loop.

The structure chart for *process rfs messages* is shown in Figure 30. Descriptions of the routines called by *process rfs messages* are provided in Table 38.

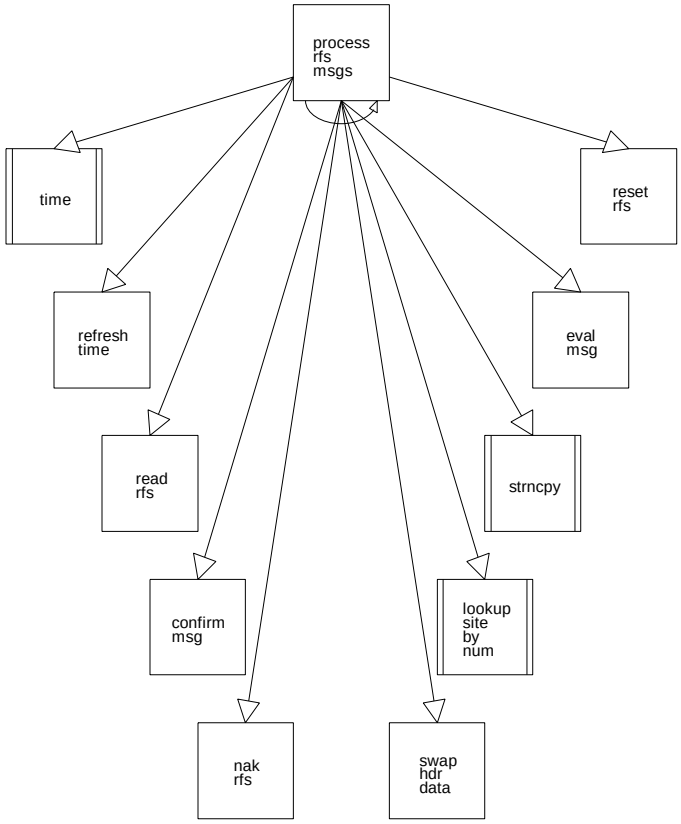


Figure 30. *process rfs msgs* structure chart

Table 38. Routines called by *process rfs msgs*

ITEM	DESCRIPTION
confirm msg	Confirm that the input data stream contains a valid message. This routine may have to be called multiple times to assemble an entire message.
eval msg	Determine what type of message was received (e.g., tag data) and perform the appropriate processing.
lookup site by num	Lookup the site index for a site using its hardware "source number".
nak rfs	Send a NAK to the RFS.
process rfs msgs	This function performs data collection and monitoring for one modem port.
read rfs	Read data from an RFS.
refresh time	Refresh the data/time for an RFS if the time-out has expired.
reset rfs	Reset an RFS if the unit is non-responsive or the user has requested a reset.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.
swap hdr data	Swap the integer fields within the header: source address, destination address and data byte count.
time	C library function to get the current system time in Unix format.

4.2.3.2.14 refresh time

This function gets the current time from the host computer, assembles a time update message, and sends it to the RFS. The structure chart for *refresh time* is shown in Figure 31. Descriptions of the routines called by *refresh time* are provided in Table 39.

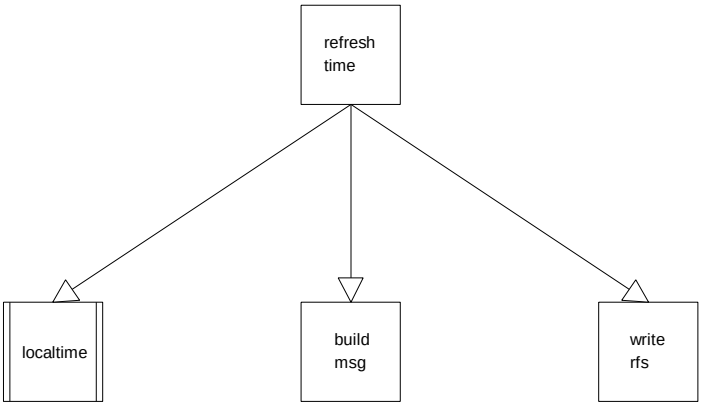


Figure 31. *refresh time* structure chart

Table 39. Routines called by *refresh time*

ITEM	DESCRIPTION
build msg	Construct an RFS message from its constituent parts.
localtime	C library function to convert a standard Unix time value into the Unix broken down structure form. This function accounts for local time variations such as daylight savings time.
refresh time	Refresh the data/time for an RFS if the time-out has expired.
write rfs	Send data to an RFS.

4.2.3.2.15 build msg

This function takes the passed information (message code, sequence number, destination address, and message body) and assembles a message in the correct format for the RFS. Operations include assigning data to the correct fields within the message, swapping integer data bytes, calculating the CRCs for the header and body of the message, and performing zero byte insertion on the message. The structure chart for *build msg* is shown in Figure 32. Descriptions of the routines called by *build msg* are provided in Table 40.

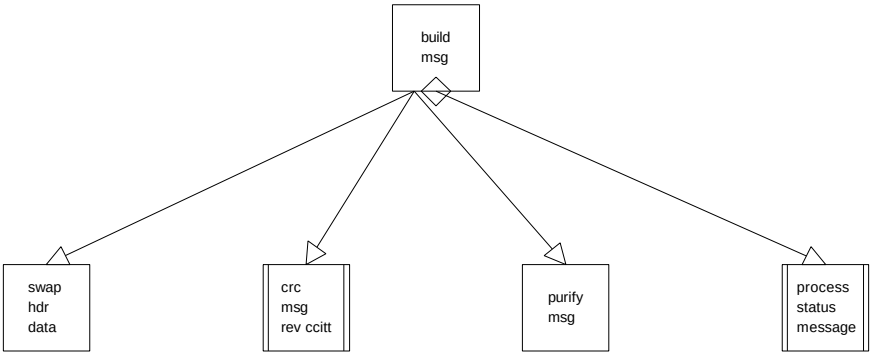


Figure 32. *build msg* structure chart

Table 40. Routines called by *build msg*

ITEM	DESCRIPTION
build msg	Construct an RFS message from its constituent parts.
crc msg rev ccitt	This function calculates the reverse CCITT CRC on the passed message.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
purify msg	Convert SOM (0xF1) and XLAT (0xF2) characters in the passed message into <XLAT (0xF2) XLAT (0xF2)> and <XLAT (0xF2) XLAT_B (0xF3)>, respectively.
swap hdr data	Swap the integer fields within the header: source address, destination address and data byte count.

4.2.3.2.16 swap hdr data

This function swaps the integer fields (source address, destination address, and byte count) in the header. This is necessary due to byte order differences between the host computer and the RFS. *swap hdr data* calls only the *SWAP* macro, so no structure chart is provided. Descriptions of the routines called by *swap hdr data* are provided in Table 41.

Table 41. Routines called by *swap hdr data*

ITEM	DESCRIPTION
SWAP	Macro to swap the bytes of an 16 bit integer in place.
swap hdr data	Swap the integer fields within the header: source address, destination address and data byte count.

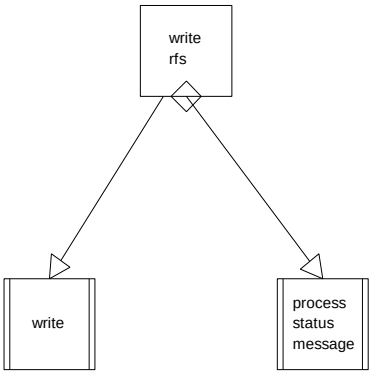
4.2.3.2.17 purify msg

This function performs a process generically called zero byte insertion. The purpose of zero byte insertion is to exclude special characters from the body of a message. In this case there are two

special characters: the start of message (SOM [$F1_{16}$]) and the translate (XLAT [$F2_{16}$]) character. Each SOM except for the one that starts the message is translated to "$F2_{16}$ $F2_{16}$". Each XLAT except the ones inserted to replace an SOM is translated to "$F2_{16}$ $F3_{16}$". These translations can cause the message to grow beyond the maximum size of the work buffer. This condition is checked, and an error is returned if the buffer size is inadequate. *purify msg* calls only *process status message*, so no structure chart is provided. Descriptions of the routines called by *purify msg* are provided in Table 42.

Table 42. Routines called by *purify msg*

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
purify msg	Convert SOM (0xF1) and XLAT (0xF2) characters in the passed message into <XLAT (0xF2) XLAT (0xF2)> and <XLAT (0xF2) XLAT_B (0xF3)>, respectively.



4.2.3.2.18 write rfs

This function writes a message to the RFS. If an error occurs, a message is sent to the log process. The structure chart for *write rfs* is shown in Figure 33. Descriptions of the routines called by *write rfs* are provided in

Table 43.

Figure 33. *write rfs* structure chart

Table 43. Routines called by *write rfs*

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
write	C library function to write data to a file or device.
write rfs	Send data to an RFS.

4.2.3.2.19 read rfs

This function reads data from the RFS. It reads as much data as is available, up to the destination buffer size. The data read does not have to be a complete message. If an error occurs, a message is sent to the log process. The structure chart for *read rfs* is shown in Figure 34. Descriptions of the routines called by *read rfs* are provided in

Table 44.

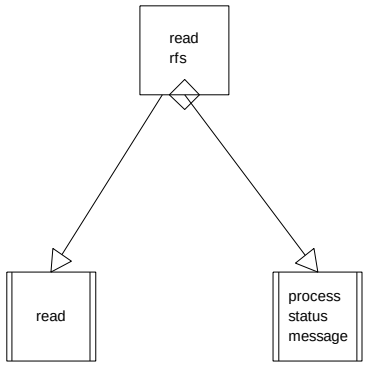


Figure 34. *read rfs* structure chart

Table 44. Routines called by *read rfs*

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
read	C library function to read data from a file or device.
read rfs	Read data from an RFS.

4.2.3.2.20 confirm msg

This function acquires a message from an RFS. It is designed so that it can be passed partial information and process as much data as is available. Each time it is called, the current state is passed in, including newly received data. The message status (*incomplete*, *incomplete with error*, *complete with error*, or *complete*) is returned by this function. These status values are described in detail in the section for the function *process rfs message*. The message acquisition occurs through the use of a state machine that is shown in **Error! Reference source not found.**. The state machine has the following states:

- Wait for SOM - Wait for the Start of Message (SOM [$F1_{16}$]) character
- Body of Header - Process characters which make up the body of the header
- XLAT in Header - Process an XLAT ($F2_{16}$) character in the header
- Body of Message - Process characters which make up the body of the message
- XLAT in Body - Process an XLAT character in the body of the message

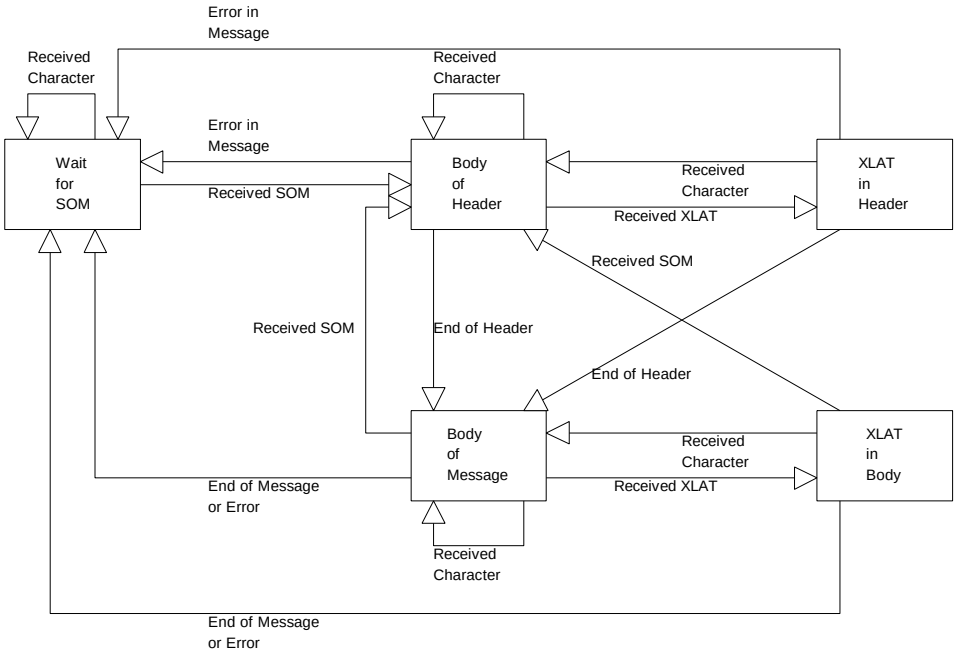


Figure 35. RFS Message Acquisition State Machine

The Wait for SOM state is the initial state and is used to synchronize message acquisition with received data. Every message starts with an SOM character. The program stays in this state until the an SOM is seen in the input data stream. The Body of Header state is used to acquire the characters that make up the header of the message. The header size is fixed and the program transitions to the Body of Message state when all of the header characters are received. If an XLAT character is received in the header, Body of Header changes to XLAT in Header, which processes the character after the XLAT. XLAT in Header evaluates the next character received and transitions back to Body of Header, unless the end of the header has been reached, in which case it transitions to Body of Message. Body of Message and XLAT in Body work similar to Body of Header and XLAT in Header except that the message size is not fixed and comes from the header. Also, when the message is complete, the status is marked as *complete* and the state is changed to Wait for SOM. If an error occurs, then the next state will always be Wait for SOM.

confirm msg implements the states by calling the *confirm body* (for Body of Header and Body of Message) and *confirm xlat* (for XLAT in Header and XLAT in Message) functions that perform the actions required by the passed state information. The structure chart for *confirm msg* is shown in Figure 36. Descriptions of the routines called by *confirm msg* are provided in Table 45.

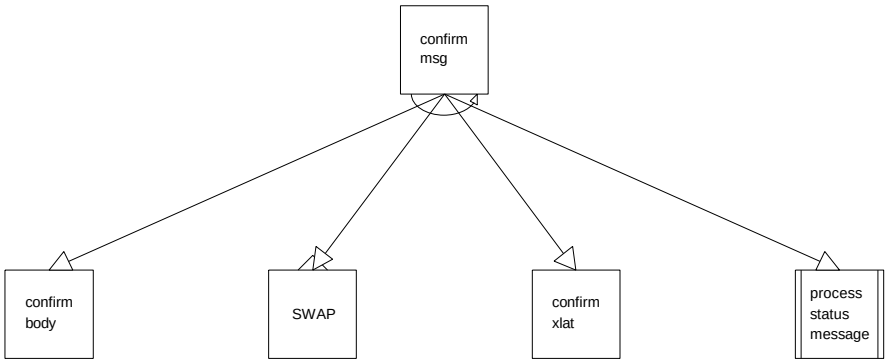


Figure 36. *confirm msg* structure chart

Table 45. Routines called by *confirm msg*

ITEM	DESCRIPTION
confirm body	Process the receipt of characters in the header or message body.
confirm msg	Confirm that the input data stream contains a valid message. This routine may have to be called multiple times to assemble an entire message.
confirm xlat	Process the receipt of characters in the header or message body following the receipt of an XLAT.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
SWAP	Macro to swap the bytes of an 16 bit integer in place.

4.2.3.2.21 confirm body

This function evaluates the current input character to determine the next state in the message acquisition process. If the received character is an SOM ($F1_{16}$), the new state is set unconditionally to Body of Header with a data index of one. If the received character is an XLAT ($F2_{16}$), then the state is changed to the passed XLAT state and no character is stored. If any other character is received, then it is stored. *eom check* is called to determine if the header/message is complete. *confirm body* calls only *eom check*, so no structure chart is provided. Descriptions of the routines called by *confirm body* are provided in Table 46.

Table 46. Routines called by *confirm body*

ITEM	DESCRIPTION
confirm body	Process the receipt of characters in the header or message body.
eom check	Check to see if the end of a message has been reached.

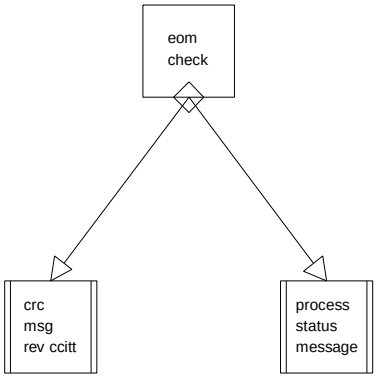


Figure 37. *eom check* structure chart

4.2.3.2.22 eom check

This function checks to see if the end of the header or message has been reached. It checks the received character count against the header/message size and if it has been reached, then the CRC for the header/message is calculated. The final value should be zero; if it is, and the header is complete, then the state is set to Body of Message; otherwise, the message is complete and the return status is set accordingly. If the CRC did not agree, then the return status is set to *incomplete with error* or *complete with error* depending on whether the header or message is being processed. The structure chart for *eom check* is shown in Figure 37. Descriptions of the routines called by *eom check* are provided in Table 47.

Table 47. Routines called by *eom check*

ITEM	DESCRIPTION
crc msg rev ccitt	This function calculates the reverse CCITT CRC on the passed message.
eom check	Check to see if the end of a message has been reached.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

4.2.3.2.23 confirm xlat

This function evaluates the current input character, which was "escaped" by the preceding XLAT ($F2_{16}$). If a SOM ($F1_{16}$) is received, the next state is set to Body of Header, and the data index is set to one. If an XLAT is received, then an SOM is inserted into the data. If an XLAT_B ($F3_{16}$) is received, then an XLAT is inserted into the data. If any other character is received it is stored, although this is probably an error. If it is an error, the error should be detected by the CRC. *eom check* is called to determine if the header/message is complete. The structure chart for *confirm xlat* is shown in Figure 38. Descriptions of the routines called by *confirm xlat* are provided in Table 48.

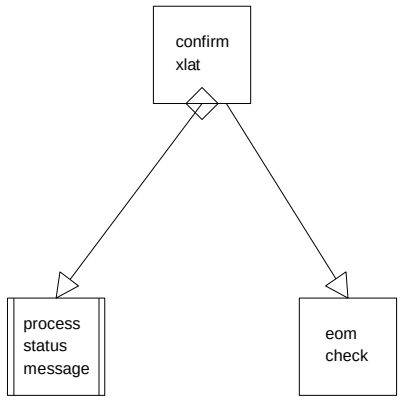


Figure 38. *confirm xlat* structure chart

Table 48. Routines called by *confirm xlat*

ITEM	DESCRIPTION
confirm xlat	Process the receipt of characters in the header or message body following the receipt of an XLAT.
eom check	Check to see if the end of a message has been reached.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

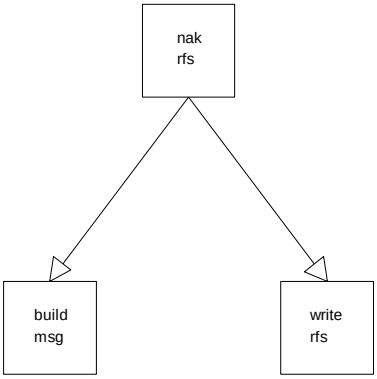


Figure 39. *nak rfs* structure chart

4.2.3.2.24 nak rfs

This function assembles a Negative Acknowledge (NAK) message and sends it to the RFS. The structure chart for *nak rfs* is shown in Figure 39. Descriptions of the routines called by *nak rfs* are provided in

Table 49.

Table 49. Routines called by *nak rfs*

ITEM	DESCRIPTION
build msg	Construct an RFS message from its constituent parts.
nak rfs	Send a NAK to the RFS.
write rfs	Send data to an RFS.

4.2.3.2.25 eval msg

This function takes a complete valid message and determines its disposition. If the message is neither an acknowledge (ACK) nor a negative acknowledge (NAK), then an ACK is built and sent to the RFS. The most common message is tag read data. When this message is received, it is written to DPF over the tag data socket. The message time and count are updated. Event messages are also received frequently. No processing is associated with them and they are ignored. If the received message is an ACK, then it should be for the most recent reset or time update command. The ACK flag associated with the appropriate message is updated, or the received

ACK is ignored if it does not match a transmitted command. A NAK may also be received. If the NAK is for either a reset or time update command, then the appropriate command is resent (up to three times). Otherwise, the function updates the error counts and ignores the NAK. Finally, if any other message type is received, it is ignored and a warning message is logged, since no other message types are expected. The structure chart for *eval msg* is shown in Figure 40. Descriptions of the routines called by *eval msg* are provided in Table 50.

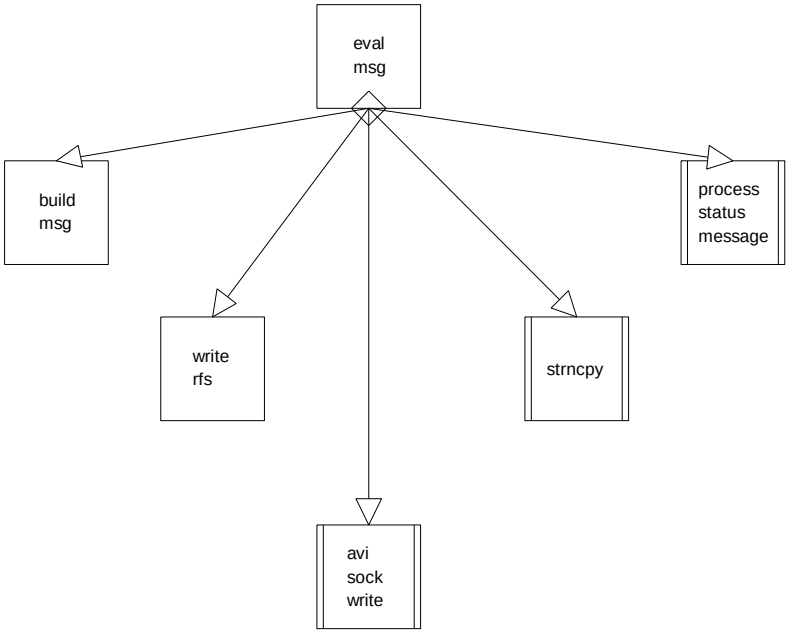


Figure 40. *eval msg* structure chart

Table 50. Routines called by *eval msg*

ITEM	DESCRIPTION
avi sock write	Write to a socket with retry.
build msg	Construct an RFS message from its constituent parts.
eval msg	Determine what type of message was received (e.g., tag data) and perform the appropriate processing.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.
write rfs	Send data to an RFS.

4.2.3.2.26 reset rfs

If the reset rfs flag is set, then this function assembles a reset rfs message, and sends it to the RFS. The structure chart for *reset rfs* is shown in Figure 41. Descriptions of the routines called by *reset rfs* are provided in

Table 51.

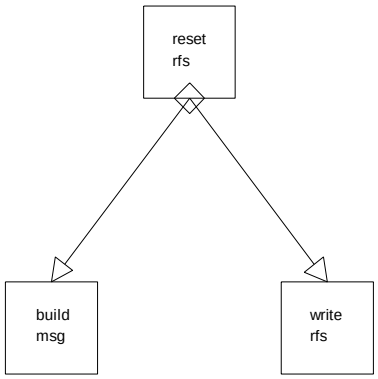


Figure 41. *reset rfs* structure chart

Table 51. Routines called by *reset rfs*

ITEM	DESCRIPTION
build msg	Construct an RFS message from its constituent parts.
reset rfs	Reset an RFS if the unit is non-responsive or the user has requested a reset.
write rfs	Send data to an RFS.

4.2.3.2.27 init modem

This function implements a state machine that drives modem operations. Each time the function is called the modem initialization can proceed to the next step. A master time-out mechanism is implemented outside of the state machine and is checked prior to evaluating the current state. It can be disabled. The state machine is shown in Figure 42 and has the following states:

- Wait - Wait the minimum period for a command to be executed by the modem
- Initial - Starting state for modem initialization
- Hangup - Send the modem the hangup command
- Reset - Send the modem the reset command
- Init - Send the modem the initialization command
- Answer - Waif for the modem to responded to the initialization command
- Answer2 - Wait for a modem connection to occur
- Connected - The modem is connected

The *wait* state is used to delay a minimum amount of time for a modem command to be executed and a response sent to DCM. The *initial* state closes and reopens the serial port and sends the modem attention command, and sets the next state to *hangup*. The *hangup* state flushes the input port, writes the hangup command to the modem, and sets the next state to *reset*. The *reset* state

reads the modem's response to the hangup command. If the response indicates success, then it sends the reset command and sets the next state to *init*. The *init* state reads the modem's response to the reset command. If the response indicates success, then it sends the modem initialization string and sets the next state to *answer*. The *answer* state reads the modem's response to the initialization command. If the response indicates success, then it sets the next state to *answer2*. The *answer2* state waits for the modem to connect. When the modem connects, it sets the next state to *connected*. The *connected* state monitors the modem connection. If the connection is dropped, it sets the next state to *initial*. All states except *answer* and *answer2* used *wait* to delay before checking for a modem response. All states except *connected* have a maximum time-out, which is checked at the beginning of the function, and will cause the process to begin again. The *connected* state starts over when it detects that the line has been dropped.

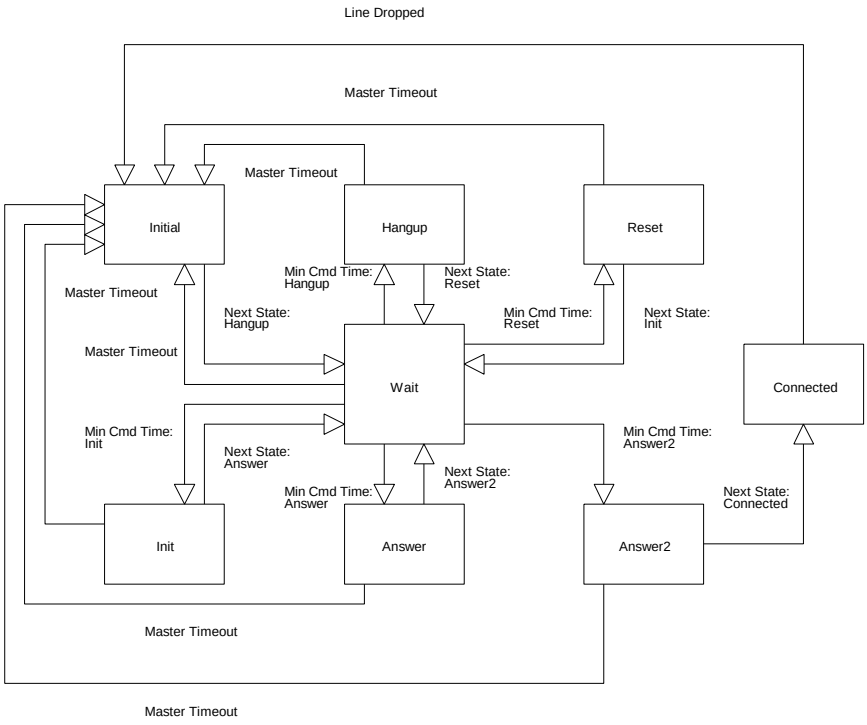


Figure 42. Modem initialization state machine

The structure chart for *init modem* is shown in Figure 43. Descriptions of the routines called by *init modem* are provided in Table 52.

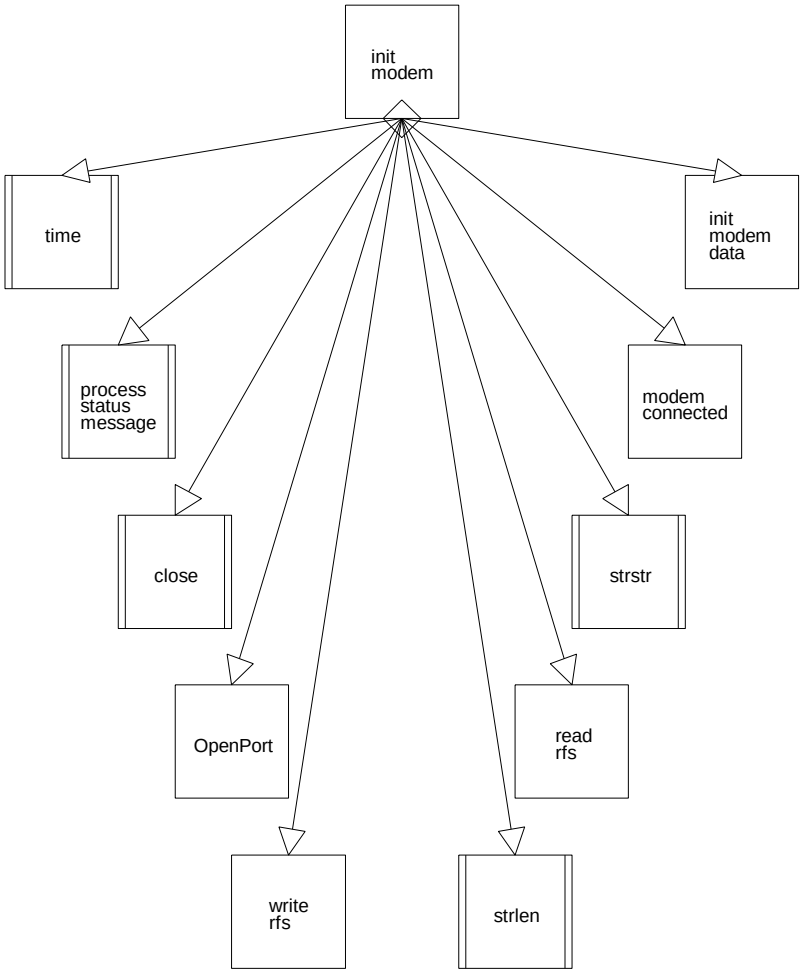


Figure 43. *init modem* structure chart

Table 52. Routines called by *init modem*

ITEM	DESCRIPTION
close	C library function to close a file or device.
init modem	Manage the state of a modem from initialization through a connection.
init modem data	Initialize the passed modem data structure.
modem connected	Check the serial port status lines to see if the modem on the passed file descriptor is still connected.
OpenPort	This function opens and configures a serial port.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
read rfs	Read data from an RFS.
strlen	C library function to return the length of the passed string.
strstr	Locate the first occurrence of the second string in the first string.

ITEM	DESCRIPTION
time	C library function to get the current system time in Unix format.
write rfs	Send data to an RFS.

4.2.3.2.28 OpenPort

This function opens and configures a serial port. The port is opened and the BAUD rate is passed to CofigurePort, which will configure the port attributes. The structure chart for *OpenPort* is shown in Figure 44. Descriptions of the routines called by *OpenPort* are provided in Table 53.

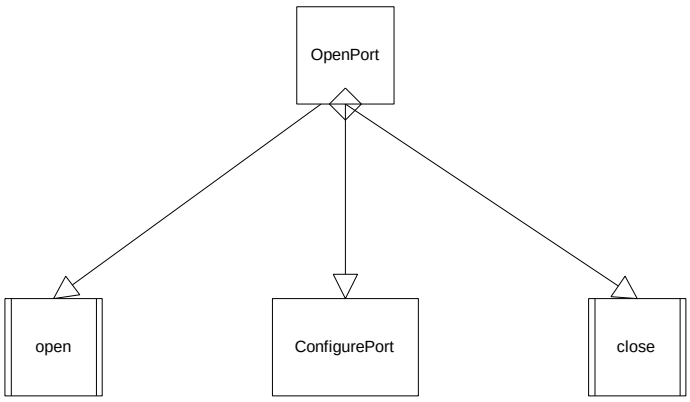


Figure 44. *OpenPort* structure chart

Table 53. Routines called by *OpenPort*

ITEM	DESCRIPTION
close	C library function to close a file or device.
ConfigurePort	Configure a port for AVI use including BAUD rate and other settings.
open	C library function to open a file or device.
OpenPort	This function opens and configures a serial port.

4.2.3.2.29 ConfigurePort

This function gets the current settings for the port attributes, determines the code for the requested BAUD rate, and sets the revised settings required for the AVI ports. The port configuration parameters are unchanged except that the following are set explicitly:

- Ignore break condition and parity errors
- Do not post-process output
- Set bits per character to 8 and stop bits to 2
- Enable inbound and outbound flow control
- Disable terminal functions
- Set read time-out to 1 second

The structure chart for *ConfigurePort* is shown in Figure 45. Descriptions of the routines called by *ConfigurePort* are provided in Table 54.

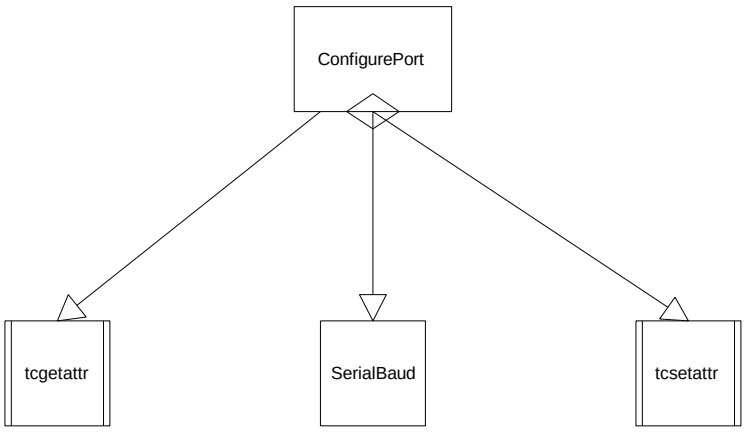


Figure 45. *ConfigurePort* structure chart

Table 54. Routines called by *ConfigurePort*

ITEM	DESCRIPTION
ConfigurePort	Configure a port for AVI use including BAUD rate and other settings.
SerialBaud	This function returns the BAUD code used by tcgetattr and tcsetattr when passed an integer representing a BAUD rate (e.g., 9600).
tcgetattr	C library function to get the attributes associated with asynchronous communications ports (serial ports).
tcsetattr	C library function to set the attributes associated with asynchronous communications ports (serial ports).

4.2.3.2.30 SerialBaud

This function converts a numeric BAUD rate (for example, 9600) into its appropriate code for the *tcsetattr* function. Supported BAUD rates include: 1200, 2400, 4800, 9600 and 19200. Neither a structure chart nor a description of routines called table is provided as *SerialBaud* calls no subroutines.

4.2.3.2.31 handle config changes

This function handles configuration changes that are received on the configuration changes socket. If data is received and the site index is in the valid range, then the request is processed. The request may be to reset a site, enable a site, or disable a site. To reset a site, the current port number on which the site is connected is checked. If it is valid, then the flag to send a reset to the site is set and *process rfs msgs* is called to perform out-of-turn processing on the required port. For an enable or disable the appropriate value is set in the dcm_status structure. Invalid sites or codes are ignored. Completion status is written to the requester and the socket is disconnected if an error occurred. The structure chart for *handle config changes* is shown in Figure 46. Descriptions of the routines called by *handle config changes* are provided in Table 55.

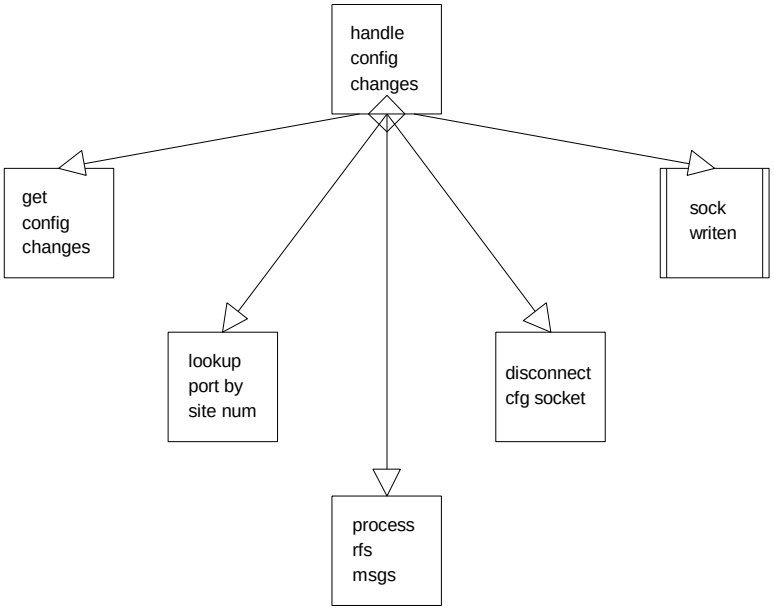


Figure 46. *handle config changes* structure chart

Table 55. Routines called by *handle config changes*

ITEM	DESCRIPTION
disconnect cfg socket	Disconnect from a socket.
get config changes	Look for data arriving on the configuration changes socket. Read the data when it is present.
handle config changes	Handle configuration changes which arrive on the configuration change socket.
lookup port by site num	Lookup the current port number for a site using its site number.
process rfs msgs	This function performs data collection and monitoring for one modem port.
sock writen	MDI Socket routine used to write a specified number of bytes to a specified socket.

4.2.3.2.32 get config changes

This function sets up and performs a *select* on the configuration change socket. If the select indicates a request, then it is processed. The structure chart for *get config changes* is shown in Figure 47. Descriptions of the routines called by *get config changes* are provided in Table 56.

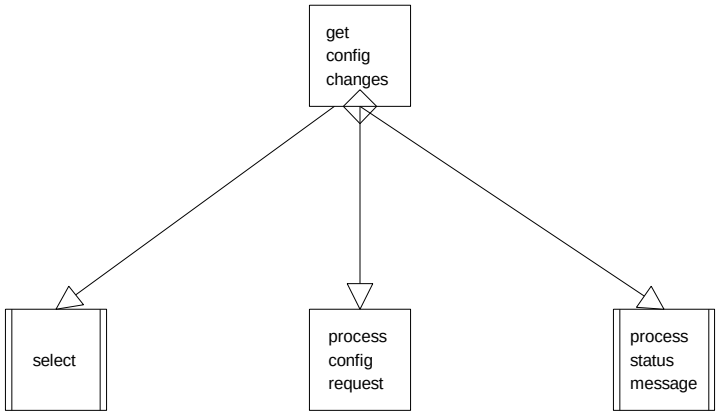


Figure 47. get config changes structure chart

Table 56. Routines called by *get config changes*

ITEM	DESCRIPTION
get config changes	Look for data arriving on the configuration changes socket. Read the data when it is present.
process config request	Read the configuration change from the socket with error checking.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
select	C Library Function used to multiplex synchronous I/O. The list of file descriptors for reading, writing, and receiving exceptions are examined and any file descriptors that are ready for reading, writing, or have an exceptional condition pending are identified.

4.2.3.2.33 process config request

This function processes the requests received on the configuration change socket. Requests may be of two types: a connection request or a change request. If the request is on the connect socket, then unless there are errors, the connection is accepted and the new socket is set to be non-blocking. Errors may cause the socket to be disconnected. If the request is not on the listen socket, then read the configuration change data. If an error occurs then the socket may be disconnected. Otherwise, return the configuration change data to the caller. The structure chart for *process config request* is shown in Figure 48. Descriptions of the routines called by *process config request* are provided in Table 57.

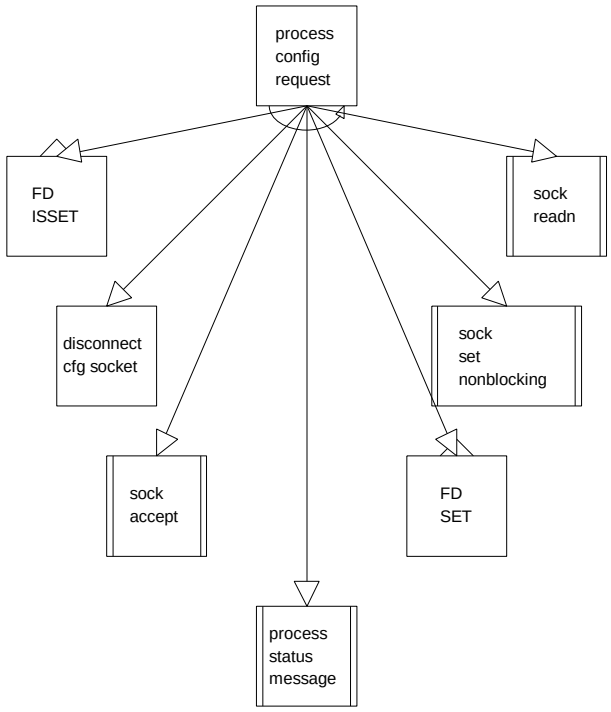


Figure 48. *process config request* structure chart

Table 57. Routines called by *process config request*

ITEM	DESCRIPTION
disconnect cfg socket	Disconnect from a socket.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
FD SET	C library macro to set a file descriptor in a file descriptor set.
process config request	Read the configuration change from the socket with error checking.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
sock accept	MDI Socket routine that accepts connections on the specified listen socket.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.
sock set nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

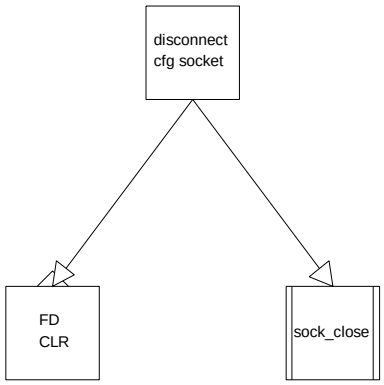


Figure 49. *disconnect cfg socket* structure chart

4.2.3.2.34 disconnect cfg socket

This function clears the a socket descriptor from a descriptor set and closes the socket. The structure chart for *disconnect cfg socket* is shown in Figure 49. Descriptions of the routines called by *disconnect cfg socket* are provided in

Table 58.

Table 58. Routines called by *disconnect cfg socket*

ITEM	DESCRIPTION
disconnect cfg socket	Disconnect from a socket.
FD CLR	C library macro which clears a given file descriptor in a file descriptor set.
sock close	MDI Socket routine used to close the specified socket connection.

4.2.3.2.35 lookup port by site num

This function uses the passed site number and looks for a corresponding port on which data is arriving for that port. The indexed port number is returned or -1 if no port was found. Neither a structure chart nor a description of routines called table is provided as *lookup port by site num* calls no subroutines.

4.2.3.2.36 update status data

This function updates the DCM global status information. It first marks all sites as not connected. Next, it updates the connected status for each port with a defined site. If all sites are okay, then overall status is refreshed to be okay. The status is set no okay elsewhere. Finally, the status is cleared for all sites that are not connected. *update status data* calls only *process status set status type value*, so no structure chart is provided. Descriptions of the routines called by *update status data* are provided in Table 59.

Table 59. Routines called by *update status data*

ITEM	DESCRIPTION
process status set status ...	This function is used to set the value associated with the specified process status status type.
update status data	Update global site status information.

4.2.3.3 Data Processing and Filtering (DPF)

The Data Processing and Filtering (DPF) process receives tag reads from DCM, archives them to disk, and stores them in a local hash table. Periodically, it updates the speed, time and status of the AVI links for the Data Server. It also archives read quantities and speed and time averages. Tag data is purged when it has aged and is no longer useful. The following sections describe the major DPF data structures and the functions that are called by DPF, which are listed in depth-first order beginning with the main entry point.

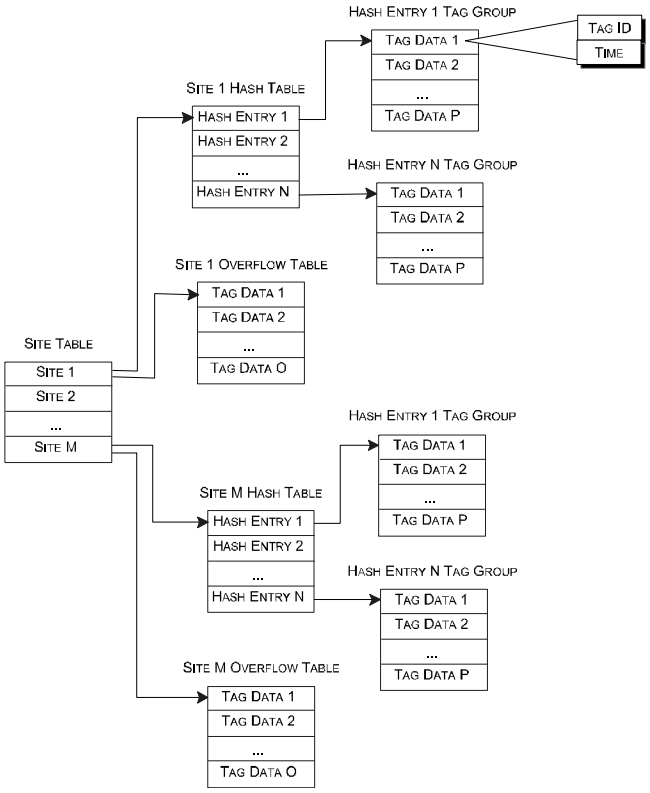


Figure 50. Tag Data Hash Table Data Structure

4.2.3.3.1 Data Structures

This section describes the major data structures used by DPF. The major data structures include the tag data hash table, the site link table, and the link data table.

4.2.3.3.1.1 Tag Data Hash Table

A diagram of the tag data hash table is presented in Figure 50. This hash table is used to store the tag read data in an organized fashion so that it can be efficiently located and maintained.

The tag data is organized by site and there is one entry for each site in the system. Each site entry includes a pointer to a hash table and to an overflow table, for a total of one hash table and one overflow table per site.

The number of entries in each hash table is configurable at runtime. Each entry in the hash table has a pointer to a tag data group, which is a table of tag data. Each tag data structure contains the unscrambled tag identifier and the time at which the tag was read. The number of entries in the tag group is configurable at runtime.

The overflow table is used to store tag reads when the tag hash table is full. The size of the overflow table is configurable at runtime.

Tag data can be located for a particular site by computing an index into the has table for that site. This computation is based on the tag identifier and is designed to evenly distribute the tag

identifiers across each of the entries in the hash table. Once the correct hash entry is located, the corresponding tag group is sequentially searched. If the tag is not located, the overflow table is then searched.

4.2.3.3.1.2 Site Link Table

The site link table is shown graphically in Figure 51. This table is an $M \times M$ matrix where M is the number of sites in the system. The matrix serves as a cross reference between source-destination pairs. If two sites form a source-destination pair in the site network, the corresponding entry in the site link table will indicate the relationship. For example, if site 1 is a source site to site 2, then entry [1,2] in the matrix will contain an entry containing data related to source-destination pair [1,2].

Each entry in the matrix includes generic data about the site-destination pair such as the distance, nominal speed, and current average speed of the links between the sites. Each entry also contains pointers to linked lists that contain the tag matches and the links that connect the two sites.

The linked list of links contains a list of all of the TransGuide AVI Link Identifiers that connect the two sites. The list of matches contains each of the matches that are currently active for the given source-destination pair.

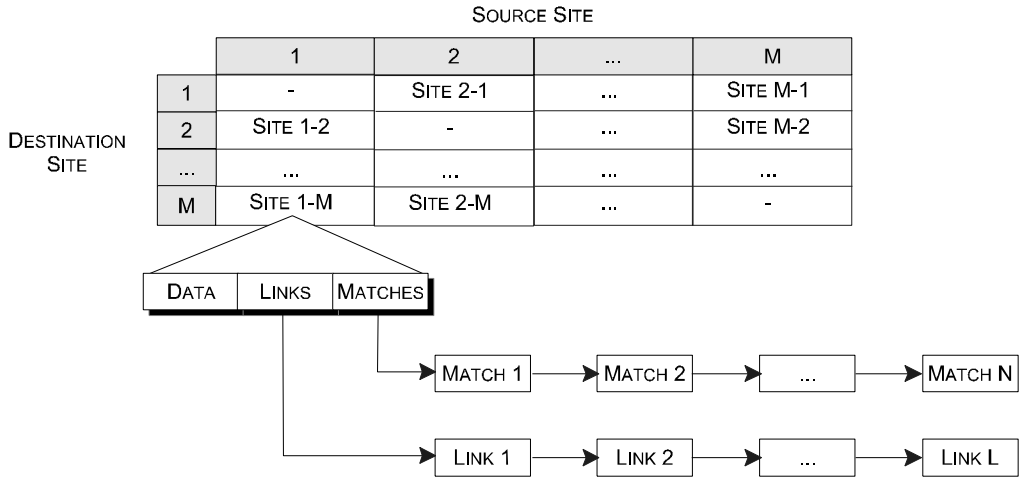


Figure 51. Site Link Table Data Structure

4.2.3.3.1.3 Link Data Table

A link data table is also maintained by the DPF process and is depicted in. This table contains one entry for each link in the AVI System. The data structures stored in this table contain the current average travel time and average speed associated with each link.

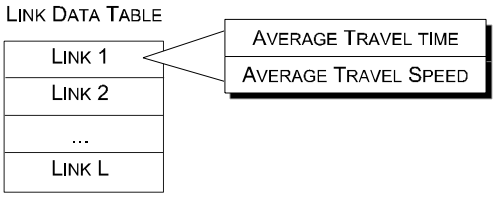


Figure 52. Link Data Table Data Structure

4.2.3.3.2 main

The main routine is responsible for setting up the clean up routines, configuring the appropriate signals to catch and ignore, etc. Virtually all the actual initialization operations occur in subroutines that are detailed later in the document. The structure chart for the main routine is shown in Figure 53. Descriptions of the routines called by the main routine of DPF are provided in Table 60.

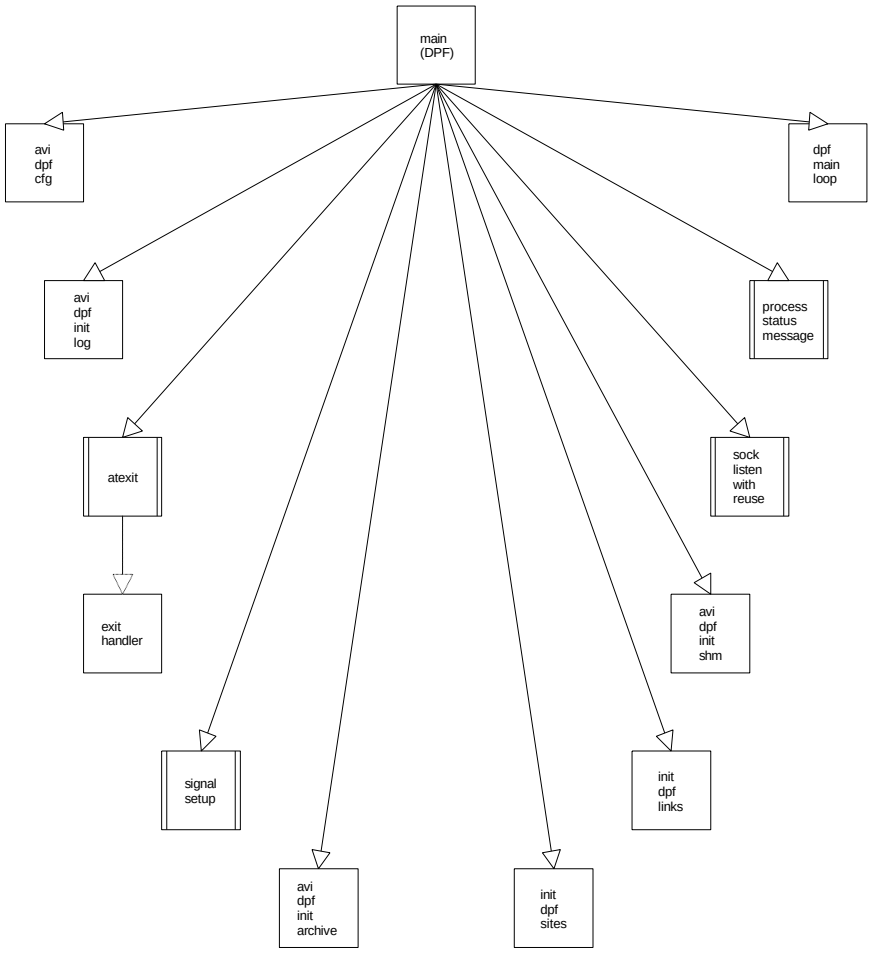


Figure 53. AVI DPF *main* structure chart

Table 60. Routines called by DPF *main*

ITEM	DESCRIPTION
atexit	C Library Function used to register routines to be called on normal termination of a program.
avi dpf cfg	Load configuration data and AVI specific data (from data files) and allocate speed and time data arrays based on the number of links.
avi dpf init archive	Open DPF archive files.
avi dpf init log	Initialize status logging for DPF.

ITEM	DESCRIPTION
avi dpf init shm	Attach/create shared memory used by DPF.
dpf main loop	This is the main loop for DPF. It alternates between waiting for new tag data to arrive from DCM and performing the periodic tasks: sending heartbeats, calculating new averages and sending them to Data Server, and archiving long term averages to file.
exit handler	Perform exit operations.
init dpf links	Initialize static data structures for DPF link information.
init dpf sites	Initialize static data structures for DPF site information. Allocate hashing table for tag data. Initialize encryption for tag scrambling.
main (DPF)	This is the main entry point for the Data Processing and Filtering process.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
signal setup	This function sets up the signal handler for all signals that are not currently handled within the calling process.
sock listen with reuse	MDI Common Socket routine used to set up a socket to listen for connections and to make the socket address reusable.

4.2.3.3.3 avi dpf cfg

avi dpf cfg loads the AVI configuration data and AVI specific data files. It is passed the MDI and AVI configuration pathnames which are passed to the AVI configuration utility function *load cfg data*. The AVI site data and site/link cross-reference data files are loaded using the *avi get avi data* AVI file utility. This routine also allocates global time and speed data structures for the AVI links. The structure chart for *avi dpf cfg* is shown in Figure 54. Descriptions of the routines called by *avi dcm cfg* are provided in Table 61.

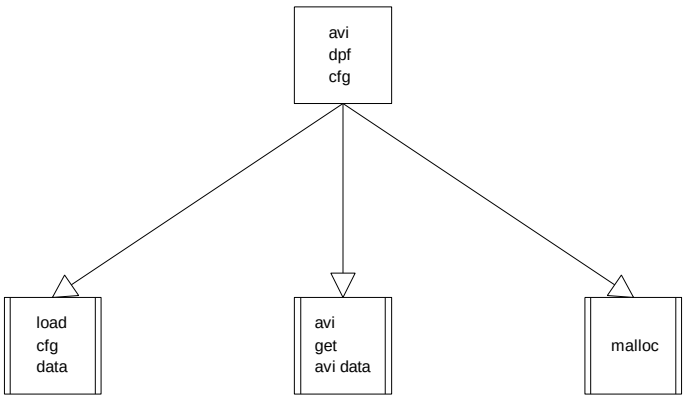


Figure 54. avi dpf cfg structure chart

Table 61. Routines called by *avi dpf cfg*

ITEM	DESCRIPTION
avi dpf cfg	Load configuration data and AVI specific data (from data files) and allocate speed and time data arrays based on the number of links.
avi get avi data	This function loads the AVI link and site data from files into dynamically allocated data structures.
load cfg data	This function loads the MDI and AVI configuration data. Data is obtained from configuration files and system function calls.
malloc	C library function to allocate memory on the heap.

4.2.3.3.4 avi dpf init log

This routine determines on which host the AVI status logger is running. The AVI status logging host is determined from the configuration data or assumed to be the same as this program's host if not defined in the configuration data. Once the hostname is determined, the routine connects to the AVI status logger process. The structure chart for *avi dpf init log* is shown in Figure 55. Descriptions of the routines called by *avi dpf init log* are provided in Table 62.

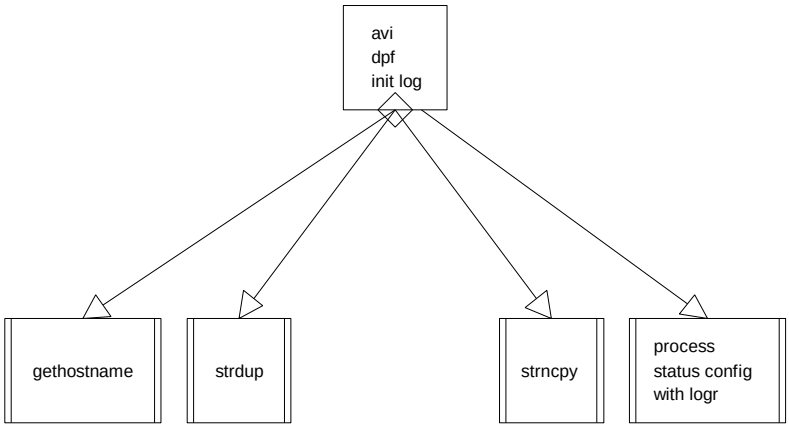


Figure 55. *avi dpf init log* structure chart

Table 62. Routines called by *avi dpf init log*

ITEM	DESCRIPTION
avi dpf init log	Initialize status logging for DPF.
gethostname	C library function to get the hostname on which the calling process is running.
process status config with logr	This routine is sets up the connection to the status logger used by the calling program.
strdup	C library function to allocate memory for and make a copy of a string.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.

4.2.3.3.5 exit handler

This routine performs exit functions for DPF. It is registered with the system using the system *atexit* function and is called automatically when a program exit condition occurs. The function logs a message to the status log indicating that DPF exited. *exit handler* calls only *process status message* so no structure chart is provided. Descriptions of the routines called by *exit handler* are provided in Table 63.

Table 63. Routines called by *exit handler*

ITEM	DESCRIPTION
exit handler	Perform exit operations.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

4.2.3.3.6 avi dpf init archive

This function creates the archive files for DPF. The archives are: tag data, speed data, and quantity data. All files are configured without time stamps. The structure chart for *avi dpf init archive* is shown in Figure 56. Descriptions of the routines called by *avi dpf init archive* are provided in Table 64.

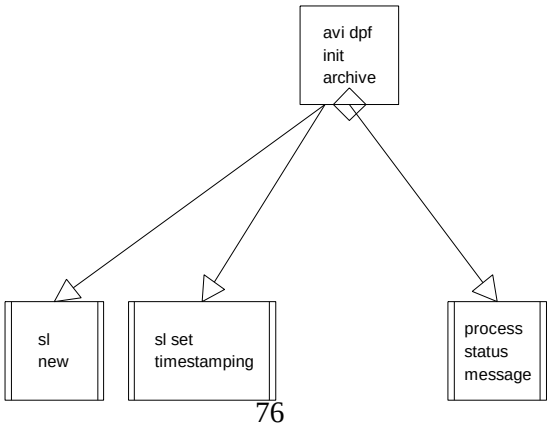


Figure 56. *avi dpf init archive* structure chart

Table 64. Routines called by *avi dpf init archive*

ITEM	DESCRIPTION
avi dpf init archive	Open DPF archive files.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
sl new	This function establishes a new archive file using the passed name and path.
sl set timestamping	This function enables/disables the timestamping feature for a log file.

4.2.3.3.7 init dpf sites

This function initializes the hash tables by calling *hash init*. Next, it allocates the space used for tag data updates to the status GUI. Finally, it initializes the encryption mechanism used to scramble the tag ids. The structure chart for *init dpf sites* is shown in Figure 57. Descriptions of the routines called by *init dpf sites* are provided in Table 65.

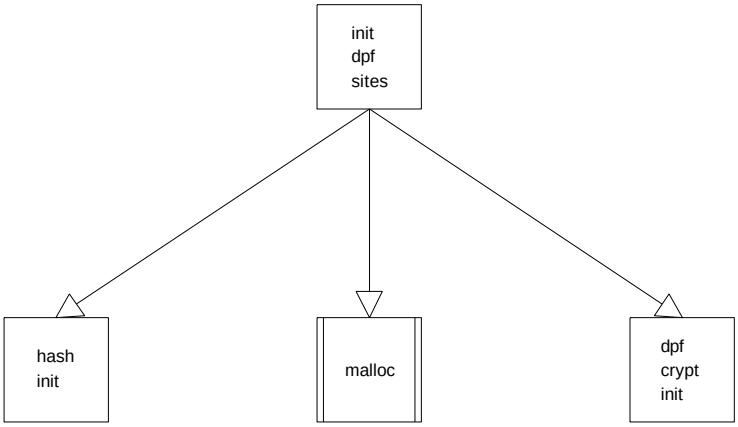


Figure 57. *init dpf sites* structure chart

Table 65. Routines called by *init dpf sites*

ITEM	DESCRIPTION
dpf crypt init	Initialize encryption algorithm so it varies with each execution of the program.
hash init	Allocate and initialize DPF tag data hash table.
init dpf sites	Initialize static data structures for DPF site information. Allocate hashing table for tag data. Initialize encryption for tag scrambling.
malloc	C library function to allocate memory on the heap.

4.2.3.3.8 hash init

This function initializes the hash tables used by DPF to store tag reads. The organization of the hash tables is described in Appendix TBD. The array of pointers to hash tables is allocated first with one entry per site. Next, the code loops to create the hash structure for each site. If any errors occur during the allocation of this data, a special free function is called to free all the constituent parts. The structure chart for *hash init* is shown in Figure 58. Descriptions of the routines called by *hash init* are provided in

Table 66.

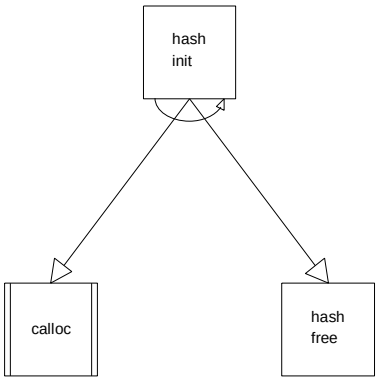


Figure 58. *hash init* structure chart

Table 66. Routines called by *hash init*

ITEM	DESCRIPTION
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
hash free	Free the DPF global tag data hash table. The table may be incomplete when this function is called.
hash init	Allocate and initialize DPF tag data hash table.

4.2.3.3.9 hash free

This function frees the hash structure used by DPF to organize tag reads. It consists of a pair of loops to free the structures on a site and group basis. *hash free* calls only *free* so no structure chart is provided. Descriptions of the routines called by *hash free* are provided in Table 67.

Table 67. Routines called by *hash free*

ITEM	DESCRIPTION
free	C Library Function used to free previously allocated memory and make it available for further allocation.
hash free	Free the DPF global tag data hash table. The table may be incomplete when this function is called.

4.2.3.3.10 dpf crypt init

This function uses the current time to initialize the encryption of tag id data. The resulting key is stored globally. *dpf crypt init* calls only *time* so no structure chart is provided. Descriptions of the routines called by *dpf crypt init* are provided in Table 68.

Table 68. Routines called by *dpf crypt init*

ITEM	DESCRIPTION
dpf crypt init	Initialize encryption algorithm so it varies with each execution of the program.
time	C library function to get the current system time in Unix format.

4.2.3.3.11 init dpf links

This function allocates and initializes the link network table and link status tables. The organization of the link network table is contained in Appendix TBD. The link/site cross-reference data file is read and used to initialize the network table. The structure chart for *init dpf links* is shown in Figure 59. Descriptions of the routines called by *init dpf links* are provided in Table 69.

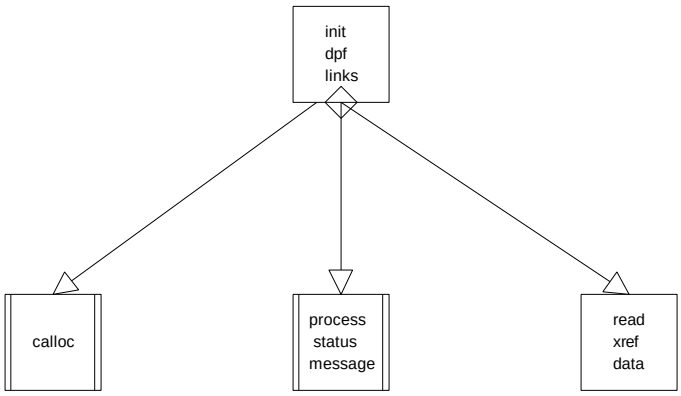


Figure 59. *init dpf links* structure chart

Table 69. Routines called by *init dpf links*

ITEM	DESCRIPTION
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
init dpf links	Initialize static data structures for DPF link information.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
read xref data	Read the site/link cross reference data from file allocating the network structure, storing the read data and performing other initializations as appropriate.

4.2.3.3.12 read xref data

This function reads the link/site cross-reference data file and uses it to initialize the network table. It reads the number of entries in the file then enters a loop to read the link data. Each line is read

then processed. Lines may be commented out with a "#" in the first column. Each line contains the source and destination site names and the threshold percentage. This is followed by one or more triples that contain the TransGuide link ID, distance over the link, and nominal speed. Each triple is read and inserted in the list of TransGuide links for this AVI link. Once all of the TransGuide links have been read, the links are traversed to calculate a weighted nominal speed for the corresponding AVI link. The structure chart for *read xref data* is shown in Figure 60. Descriptions of the routines called by *read xref data* are provided in Table 70.

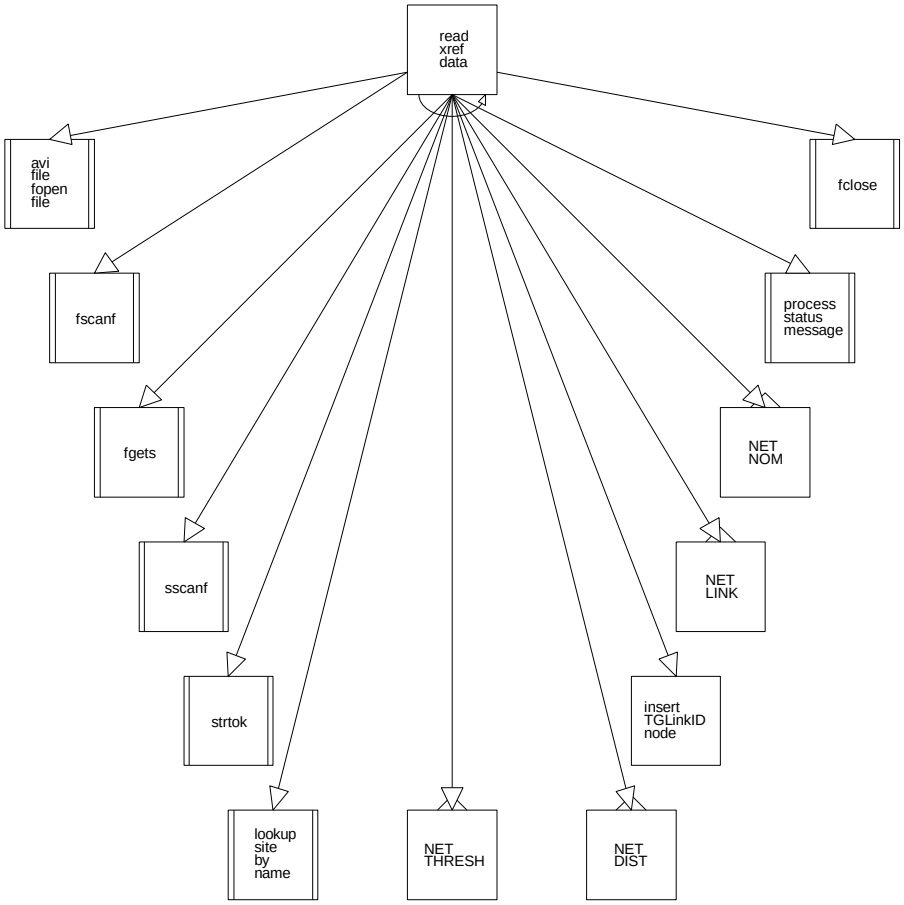


Figure 60. *read xref data* structure chart

Table 70. Routines called by *read xref data*

ITEM	DESCRIPTION
avi file fopen file	Open file identified by passed filename and path with error checking.
fclose	C library function to close a stream.
fgets	C library function to read a string from an input stream.
fscanf	C library function to scan an input stream into the referenced variables according to the specified format string.

ITEM	DESCRIPTION
insert TGLinkID node	Determine the site indices in the link network, and allocate and store the link data.
lookup site by name	Lookup the site index for a site using its name.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET LINK	Macro to ease references to the link field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET NOM	Macro to ease references to the nominal speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET THRESH	Macro to ease references to the threshold field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
read xref data	Read the site/link cross reference data from file allocating the network structure, storing the read data and performing other initializations as appropriate.
sscanf	C library function to scan a string into the referenced variables according to the specified format string.
strtok	C Library Function used to break the specified string into a sequence of tokens.

4.2.3.3.13 insert TGLinkID node

This function inserts the TransGuide link data at the appropriate index in the network (AVI link). First it uses the passed source and destination AVI link identifiers to get their corresponding site index. Next, it allocates a new node for the link id data and assigns the index corresponding to the TransGuide link id, link distance and nominal speed. Last, it inserts the new record into the list of TransGuide link ids for this AVI link. The structure chart for *insert TGLinkID node*s is shown in Figure 61. Descriptions of the routines called by *insert TGLinkID node* are provided in Table 71.

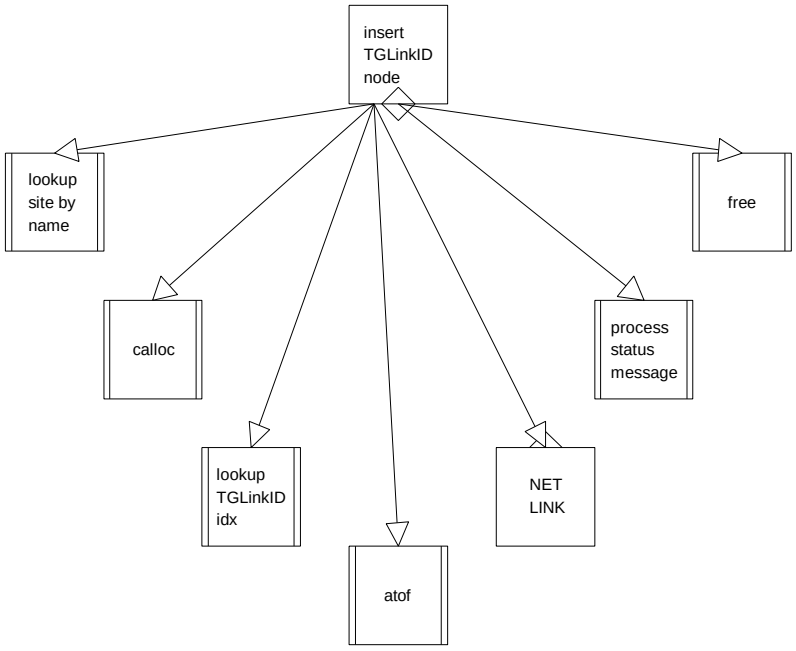


Figure 61. *insert TGLinkID node* structure chart

Table 71. Routines called by *insert TGLinkID node*

ITEM	DESCRIPTION
atof	C Library Function to convert an ASCII string to an float value.
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
free	C Library Function used to free previously allocated memory and make it available for further allocation.
insert TGLinkID node	Determine the site indices in the link network, and allocate and store the link data.
lookup site by name	Lookup the site index for a site using its name.
lookup TGLinkID idx	Look up the TransGuide link id index using its identifier.
NET LINK	Macro to ease references to the link field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

4.2.3.3.14 avi dpf init shm

DCM maintains the status data in shared memory for DPF and AVI GUI. DPF uses the information to mark link status in the link speed and time messages sent to Data Server. AVI GUI uses the information to update the user on the state of the system. This function first creates or

attaches to the shared memory depending on whether it already exists. Next, it allocates a temporary copy of the status data that is used for initialization. The initialization data is written to make sure later reads do not get garbage. The structure chart for *avi dcm init shm* is shown in Figure 62. Descriptions of the routines called by *avi dcm init shm* are provided in Table 72.

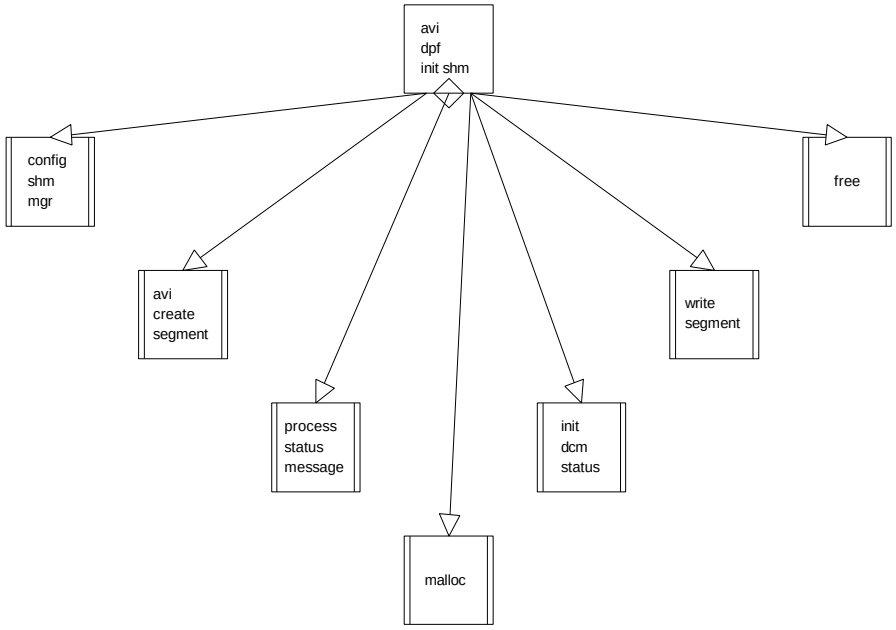


Figure 62. *avi dpf init shm* structure chart

Table 72. Routines called by *avi dpf init shm*

ITEM	DESCRIPTION
avi create segment	This function connects to or creates a shared memory segment with error checking.
avi dpf init shm	Attach/create shared memory used by DPF.
config shm mgr	MDI Shared Memory Manager routine used to initialize and configure the shared memory manager library routines for the calling program.
free	C Library Function used to free previously allocated memory and make it available for further allocation.
init dcm status	Initialize the passed DCM status structure.
malloc	C library function to allocate memory on the heap.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
write segment	This function writes the passed data into the identified shared memory segment.

4.2.3.3.15 dpf main loop

This function performs the main processing for DPF. It alternates between waiting for new tag data to arrive from DCM and performing the periodic tasks: send heartbeat, calculate new averages and send them to Data Server, and archive long term averages to file. The structure chart for *dpf main loop* is shown in Figure 63. Descriptions of the routines called by *dpf main loop* are provided in Table 73.

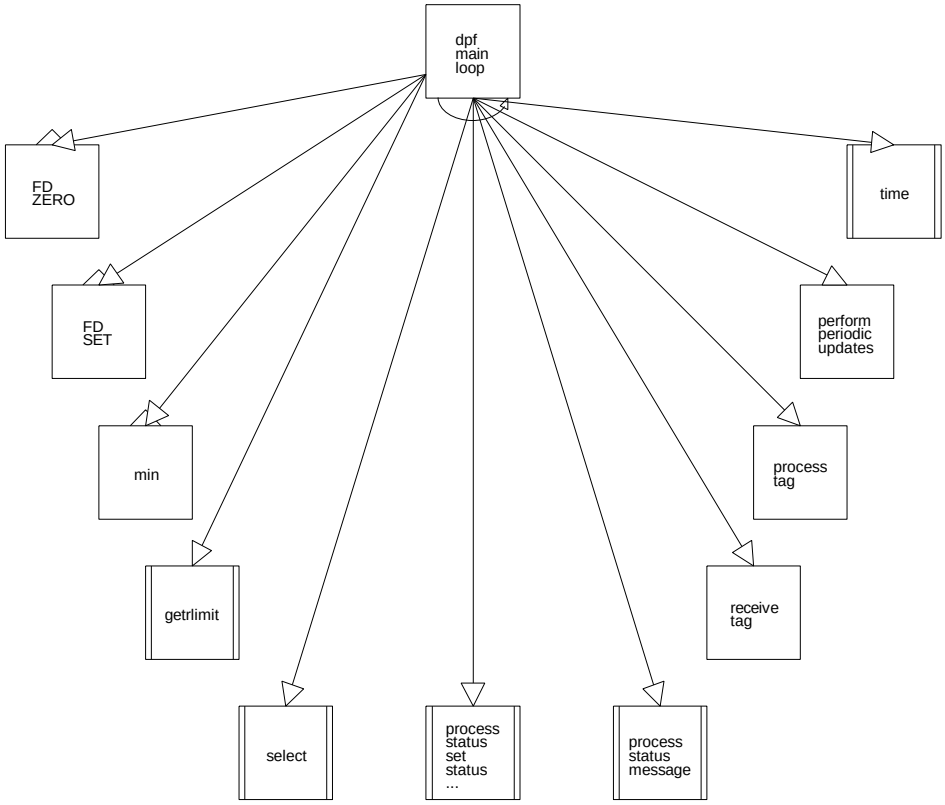


Figure 63. *dpf main loop* structure chart

Table 73. Routines called by *dpf main loop*

ITEM	DESCRIPTION
dpf main loop	This is the main loop for DPF. It alternates between waiting for new tag data to arrive from DCM and performing the periodic tasks: sending heartbeats, calculating new averages and sending them to Data Server, and archiving long term averages to file.
FD SET	C library macro to set a file descriptor in a file descriptor set.
FD ZERO	C library macro to zero a file selector set used with select().
getrlimit	C library function to get the specified system limit from the system.
min	Macro to return the minimum of two like atomic values.
perform periodic updates	Perform DPF update operations which occur on a periodic basis: heartbeat, update speed/time data, archive data, and purge old data.

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
process tag	This function takes the passed tag read data and performs processing on it: store it in the hash table, look for matches, etc.
receive tag	This function accepts connections from DCM for the tag data. It reads the tag data from the socket, and return it to the caller for further processing.
select	C Library Function used to multiplex synchronous I/O. The list of file descriptors for reading, writing, and receiving exceptions are examined and any file descriptors that are ready for reading, writing, or have an exceptional condition pending are identified.
time	C library function to get the current system time in Unix format.

4.2.3.3.16 receive tag

This function is called by *dpf main loop* when its *select* call indicates that a request is arriving on one of its sockets. This function checks the request to see if it is on the listen socket. If it is, it accepts the connection and sets the socket to be non-blocking. Otherwise, the activity must be the arrival of tag data. The tag data is read from the socket and returned to the caller. If an error occurs, this routine may disconnect the socket. The structure chart for *receive tag* is shown in Figure 64. Descriptions of the routines called by *receive tag* are provided in Table 74.

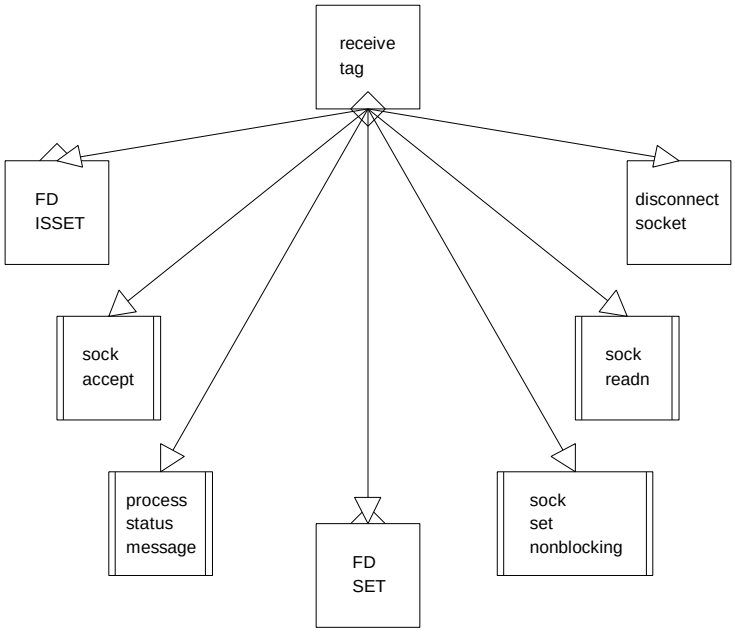


Figure 64. *receive tag* structure chart

Table 74. Routines called by *receive tag*

ITEM	DESCRIPTION
disconnect socket	Disconnect from a socket.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
FD SET	C library macro to set a file descriptor in a file descriptor set.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
receive tag	This function accepts connections from DCM for the tag data. It reads the tag data from the socket, and return it to the caller for further processing.
sock accept	MDI Socket routine that accepts connections on the specified listen socket.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.
sock set nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

4.2.3.3.17 disconnect socket

This function clears the a socket descriptor from a descriptor set and closes the socket. The structure chart for *disconnect socket* is shown in Figure 65. Descriptions of the routines called by *disconnect socket* are provided in Table 75.

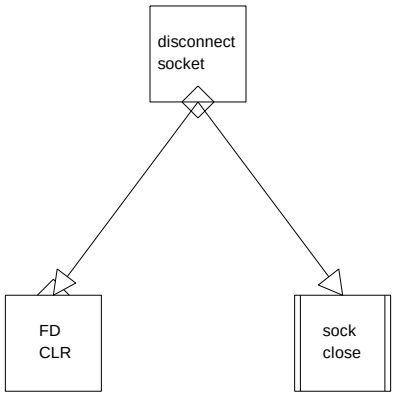


Figure 65. *disconnect socket* structure chart

Table 75. Routines called by *disconnect socket*

ITEM	DESCRIPTION
disconnect socket	Disconnect from a socket.
FD CLR	C library macro which clears a given file descriptor in a file descriptor set.
sock close	MDI Socket routine used to close the specified socket connection.

4.2.3.3.18 process tag

This function takes a new tag read and performs processing on it. First it adds the tag to the appropriate hash table and logs it in the tag archive. Next, it enters a loop that searches through the sites that are potential sources for this tag. If a match is found, and the time delta is positive, then the match is inserted with the computed travel time and speed and the source tag is deleted from the hash table. The structure chart for *process tag* is shown in Figure 66. Descriptions of the routines called by *process tag* are provided in Table 76.

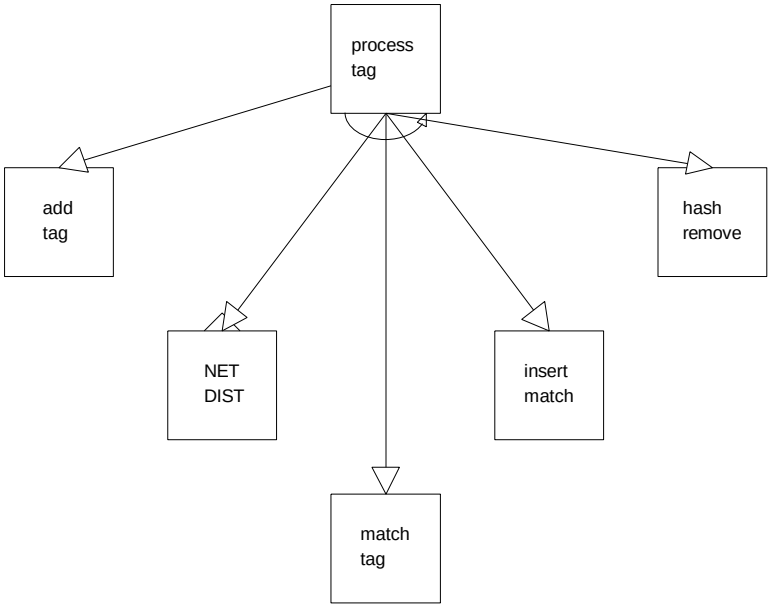


Figure 66. *process tag* structure chart

Table 76. Routines called by *process tag*

ITEM	DESCRIPTION
add tag	Take the passed tag id. Store it in the hash table, scramble it, and archive it to the tag data archive file.
hash remove	Remove the tag record indicated by the passed tag id and site index from the global tag data hash table.
insert match	Insert a tag match record (match time and speed) for the link specified by the source and destination site.
match tag	Use the passed tag id and search for matching tags in associated sites.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
process tag	This function takes the passed tag read data and performs processing on it: store it in the hash table, look for matches, etc.

4.2.3.3.19 add tag

This function takes a tag read and stores it in the hash table, scrambles the tag id, and archives it to the tag archive file. First, it looks up the site index using the RFS source number. Next, it converts the time of the tag read into a Unix format time and inserts the tag data into the hash table. Then, it updates the tag data for the status GUI. Finally, it encrypts the tag id and writes the tag data to the tag archive file. The structure chart for *add tag* is shown in Figure 67. Descriptions of the routines called by *add tag* are provided in Table 77.

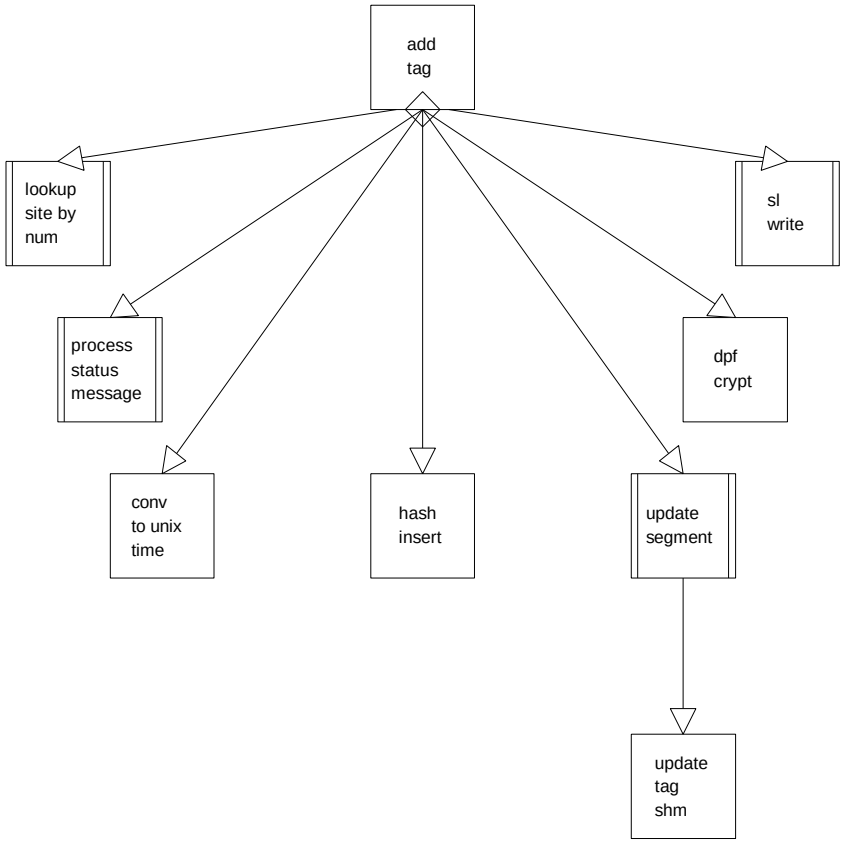


Figure 67. *add tag* structure chart

Table 77. Routines called by *add tag*

ITEM	DESCRIPTION
add tag	Take the passed tag id. Store it in the hash table, scramble it, and archive it to the tag data archive file.
conv to unix time	Convert time data in raw format into a Unix style time value.
dpf crypt	Encrypt the passed tag id data replacing the plaintext tag id with the encrypted tag id.
hash insert	Take the tag data that was passed into update segment and update the passed copy of the shared memory segment.
lookup site by num	Lookup the site index for a site using its hardware "source number".
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
sl write	This function writes the passed data to the specified archive file.
update segment	MDI library routine to update the identified shared memory.

ITEM	DESCRIPTION
update tag shm	Take the new passed tag data and update the buffer which is passed to this function. This function is called indirectly through the update segment library function.

4.2.3.3.20 conv to unix time

This function takes the time fields from the tag data and stores them into a Unix broken down time structure. The broken down time structure is then converted to a standard Unix time using *mktime*. The structure chart for *conv to unix time* is shown in Figure 68. Descriptions of the routines called by *conv to unix time* are provided in Table 78.

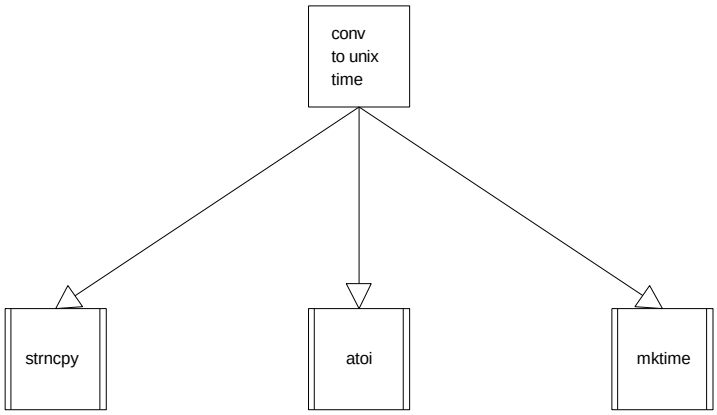


Figure 68. *conv to unix time* structure chart

Table 78. Routines called by conv to *unix time*

ITEM	DESCRIPTION
atoi	C Library Function to convert an ASCII string to an integer value.
conv to unix time	Convert time data in raw format into a Unix style time value.
mktime	C library function to convert a Unix broken down time structure into a normal Unix time value.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.

4.2.3.3.21 hash insert

This function takes the tag id and calculates a hash value for the record. The record is then inserted in the indexed group for that site's hash table or in the overflow group for that hash site. *hash insert* calls only *hash calc* so no structure chart is provided. Descriptions of the routines called by *hash insert* are provided in Table 79.

Table 79. Routines called by *hash insert*

ITEM	DESCRIPTION
hash calc	Calculate the hash value for an AVI tag.
hash insert	Take the tag data that was passed into update segment and update the passed copy of the shared memory segment.

4.2.3.3.22 hash calc

This function returns the result of performing *hash fold* on the key and dividing to generate the final hash index. *hash calc* calls only *hash fold* so no structure chart is provided. Descriptions of the routines called by *hash calc* are provided in Table 80.

Table 80. Routines called by *hash calc*

ITEM	DESCRIPTION
hash calc	Calculate the hash value for an AVI tag.
hash fold	Perform key folding for an AVI tag id (the key).

4.2.3.3.23 hash fold

This function performs key folding on the AVI tag id (the key). The folding consists of adding up the key broken up into integer chunks without overflow. The key must be an integral number of integer chunks for this to work. Neither a structure chart nor a description of routines called table is provided as *hash fold* calls no subroutines.

4.2.3.3.24 update tag shm

Take the tag data that was passed into update segment and update the passed copy of the shared memory segment. Neither a structure chart nor a description of routines called table is provided as *update tag shm* calls no subroutines.

4.2.3.3.25 dpf crypt

This routine uses the Unix *crypt* utility to encrypt the tag id in 8 character chunks. The key used by *crypt* is set when *dpf crypt init* is called. *dpf crypt* calls only *crypt* so no structure chart is provided. Descriptions of the routines called by *dpf crypt* are provided in Table 81.

Table 81. Routines called by *dpf crypt*

ITEM	DESCRIPTION
crypt	C library function used to encrypt Unix passwords. The function operates on up to eight characters at a time.
dpf crypt	Encrypt the passed tag id data replacing the plaintext tag id with the encrypted tag id.

4.2.3.3.26 match tag

This routine takes the source and destination site and a tag id and searches for entries. If both are found, the delta between them is computed and the resulting travel time is returned. *match tag*

calls only *hash find* so no structure chart is provided. Descriptions of the routines called by *match tag* are provided in Table 82.

Table 82. Routines called by *match tag*

ITEM	DESCRIPTION
hash find	Locate a tag record in the global tag data hash table returning only the tag data.
match tag	Use the passed tag id and search for matching tags in associated sites.

4.2.3.3.27 hash find

This function searches in the specified site for a tag using its id. The function returns success or failure and passes back a pointer to the tag data (if found) but not the indexing information for the tag data. *hash find* calls only *hash find idx* so no structure chart is provided. Descriptions of the routines called by *hash find* are provided in Table 83.

Table 83. Routines called by *hash find*

ITEM	DESCRIPTION
hash find	Locate a tag record in the global tag data hash table returning only the tag data.
hash find idx	Locate a tag record in the global hash table, and return its indexing information and the tag record.

4.2.3.3.28 hash find idx

This function searches in the specified site for the specified tag using its id. The function calculates the has entry within the site for this tag, then loops through the indexed group, and, if necessary, the overflow group looking for the tag. The function returns success or failure and passes back a pointer to the tag data (if found) along with the indexing information for the tag data. *hash find idx* calls only *hash calc* so no structure chart is provided. Descriptions of the routines called by *hash find idx* are provided in Table 84.

Table 84. Routines called by *hash find idx*

ITEM	DESCRIPTION
hash calc	Calculate the hash value for an AVI tag.
hash find idx	Locate a tag record in the global hash table, and return its indexing information and the tag record.

4.2.3.3.29 insert match

This function is passed the source and destination site, travel time, and match time for a tag match. A match record is allocated and loaded and placed at the head of the match list for the specified source and destination site. The structure chart for *insert match* is shown in Figure 69. Descriptions of the routines called by *insert match* are provided in Table 85.

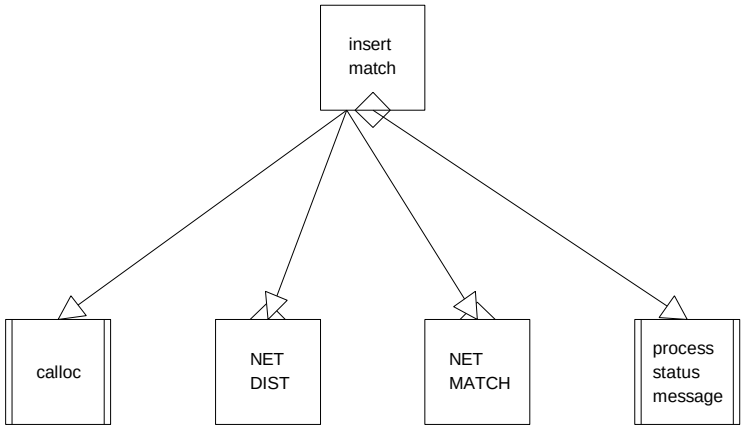


Figure 69. *insert match* structure chart

Table 85. Routines called by *insert match*

ITEM	DESCRIPTION
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
insert match	Insert a tag match record (match time and speed) for the link specified by the source and destination site.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET MATCH	Macro to ease references to the match field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

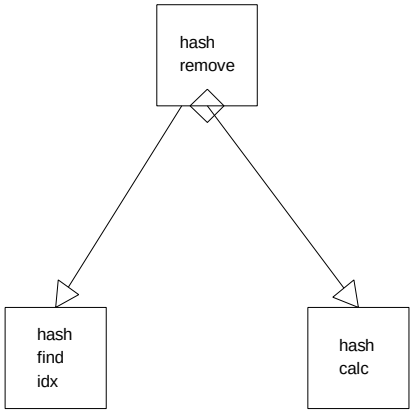


Figure 70. *hash remove* structure chart

4.2.3.3.30 hash remove

This function removes a tag record from the hash table for the indexed site. First, the indexing information for this tag is determined. If this fails, the function returns. If the entry is found in the main group, then it is removed from that group, and the first tag with the same hash index in the overflow group is copied into its place. If no tag was removed then the group is compressed. Next, if the entry was not found in the main group or an entry was copied from the overflow group, then the overflow group must be compressed. The structure chart for *hash remove* is shown in Figure 70. Descriptions of the routines called by *hash remove* are provided in Table 86.

Table 86. Routines called by *hash remove*

ITEM	DESCRIPTION
hash calc	Calculate the hash value for an AVI tag.
hash find idx	Locate a tag record in the global hash table, and return its indexing information and the tag record.
hash remove	Remove the tag record indicated by the passed tag id and site index from the global tag data hash table.

4.2.3.3.31 perform periodic updates

This function performs the periodic operations for DPF. It maintains the last time each operation was performed, and checks the current time minus the last time and compares it to the time limit to see if it is time to do that operation. The operations performed are: send heartbeat, update speed and time averages and status data and send them to the DSIF process, archive the link time/speed data, archive the read quantity data, and purge old link time/speed data. The structure chart for *perform periodic updates* is shown in Figure 71. Descriptions of the routines called by *perform periodic updates* are provided in Table 87.

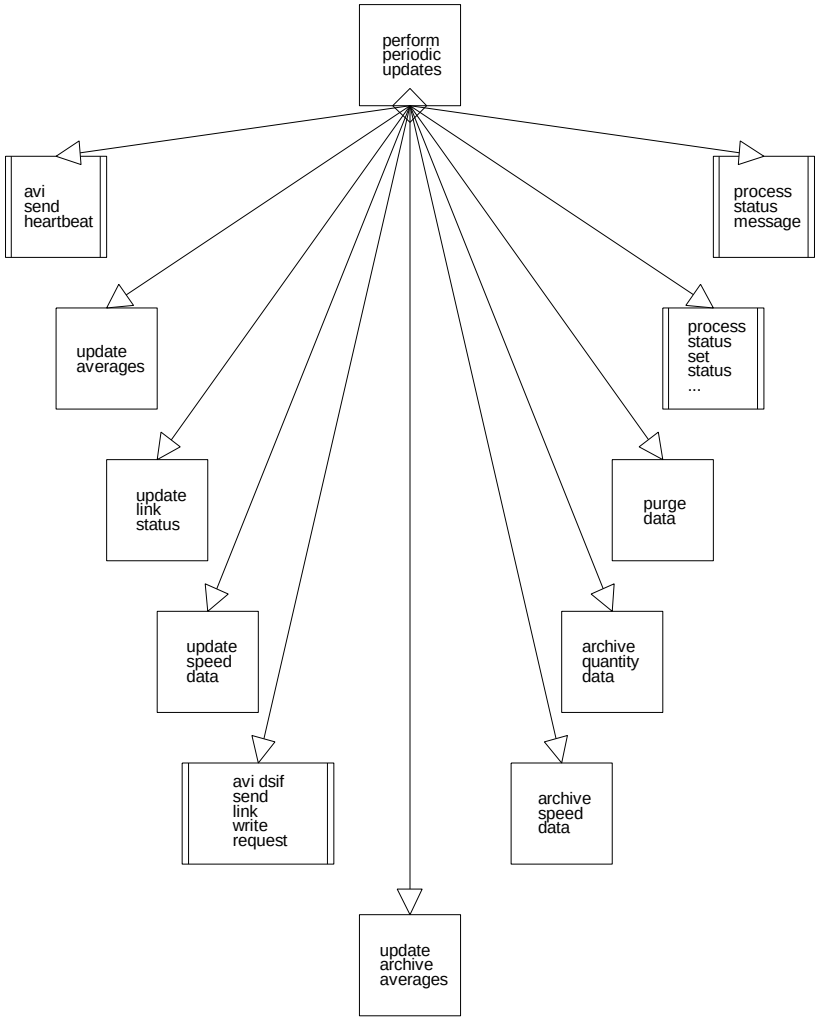


Figure 71. *perform periodic updates* structure chart

Table 87. Routines called by *perform periodic updates*

ITEM	DESCRIPTION
archive quantity data	Write the current read quantity values to the archive file.
archive speed data	Write the current archive travel time and speed values to the archive file.
avi dsif send link write request	Package the AVI link speed or time data into the correct format and send it to the Data Server.
avi send heartbeat	This function sends the heartbeat to the AVI heartbeat process with automatic reconnection and error checking.
perform periodic updates	Perform DPF update operations which occur on a periodic basis: heartbeat, update speed/time data, archive data, and purge old data.

ITEM	DESCRIPTION
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
purge data	Purge obsolete data from the global match and tag data structures.
update archive averages	This function updates the current average archive travel speeds in the DPF global link network.
update averages	This function updates the current average travel speeds in the DPF global link network.
update link status	Update the status values in the global speed and time arrays that are organized by TransGuide link id.
update speed data	Update the data values in the global speed and time arrays that are organized by TransGuide link id.

4.2.3.3.32 update averages

This function loops through all sites and destinations to update the average speeds for each defined link. If the previous speed is valid, then new average is used by calculating a threshold; otherwise, the default limits are used. The average speed is calculated using the newly calculated limits. Finally, if the resulting calculation used values, then the speed and time are updated; otherwise, if no values were used in the calculation, then the current value is retained, unless the nominal data time-out has expired. In that case, the nominal speed is reported See Appendix TBD for more details on how the speed values are calculated. The structure chart for *update averages* is shown in Figure 72. Descriptions of the routines called by *update averages* are provided in Table 88.

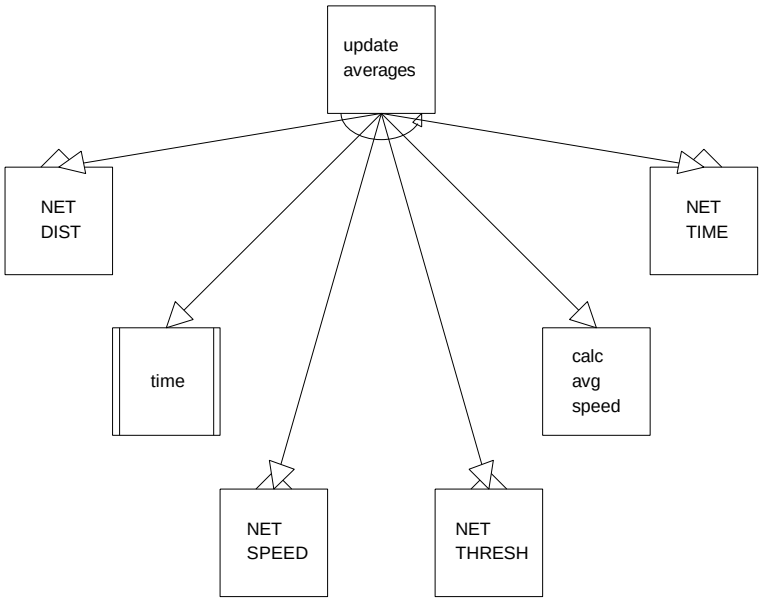


Figure 72. *update averages* structure chart

Table 88. Routines called by *update averages*

ITEM	DESCRIPTION
calc avg speed	Calculate the average speed over a link using matches which have not aged beyond their usefulness.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET SPEED	Macro to ease references to the speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET THRESH	Macro to ease references to the threshold field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET TIME	Macro to ease references to the time field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
time	C library function to get the current system time in Unix format.
update averages	This function updates the current average travel speeds in the DPF global link network.

4.2.3.3.33 calc avg speed

This function calculates the average speed for one site. The function loops through the match nodes associated with the site. Each match in the list is included in the averages until the cutoff time is reached (matches are too old). Then walking the match list ends since the matches are

included in reverse chronological order. The average is calculated and returned. If no matches were found, then the negative of the nominal value is returned. The structure chart for *calc avg speed* is shown in Figure 73. Descriptions of the routines called by *calc avg speed* are provided in

Table 89.

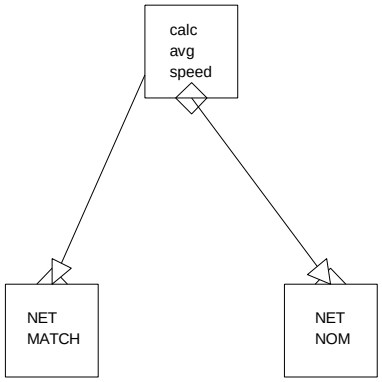


Figure 73. *calc avg speed* structure chart

Table 89. Routines called by calc avg speed

ITEM	DESCRIPTION
calc avg speed	Calculate the average speed over a link using matches which have not aged beyond their usefulness.
NET MATCH	Macro to ease references to the match field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET NOM	Macro to ease references to the nominal speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.

4.2.3.3.34 update link status

This function updates the status for the TransGuide links based on the associated physical links. A local copy of the DCM site status is allocated. Next, the data is read from shared memory. A loop is executed once for each physical site. This site is used first as a source and then as a destination within two inner loops. The inner loops sweep through the sites as well. Sites default to LINKACTIVE. In each loop, if any of the physical sites which map to a TransGuide link is not connected, then the status is set to LINKINACTIVE. The local memory is freed upon completion. The structure chart for *update link status* is shown in Figure 74. Descriptions of the routines called by *update link status* are provided in Table 90.

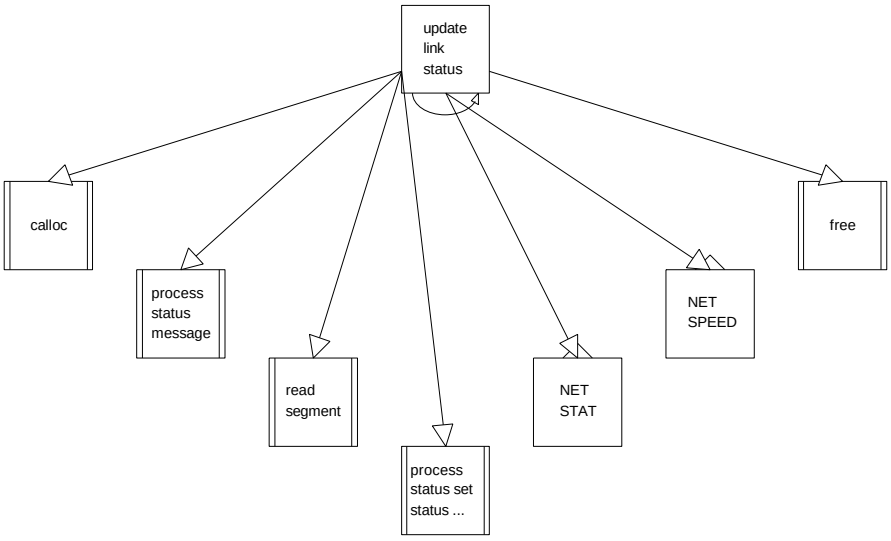


Figure 74. *update link status* structure chart

Table 90. Routines called by *update link status*

ITEM	DESCRIPTION
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
free	C Library Function used to free previously allocated memory and make it available for further allocation.
NET SPEED	Macro to ease references to the speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET STAT	Macro to ease references to the status field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
read segment	MDI library routine to read the identified shared memory segment.
update link status	Update the status values in the global speed and time arrays that are organized by TransGuide link id.

4.2.3.3.35 update speed data

This function loops through the entire link network looking for valid links. For each valid link, the list of TransGuide links is walked. Each TransGuide link is indexed in the speed and time arrays which are sent to Data Server and the corresponding status, time and speed values are updated. The structure chart for *update speed data* is shown in Figure 75. Descriptions of the routines called by *update speed data* are provided in Table 91.

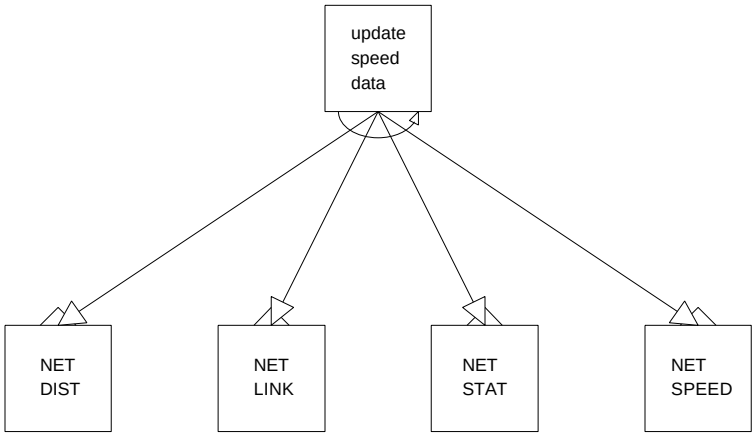


Figure 75. *update speed data* structure chart

Table 91. Routines called by *update speed data*

ITEM	DESCRIPTION
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET LINK	Macro to ease references to the link field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET SPEED	Macro to ease references to the speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET STAT	Macro to ease references to the status field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
update speed data	Update the data values in the global speed and time arrays that are organized by TransGuide link id.

4.2.3.3.36 update archive averages

This function loops through all sites and destinations to update the archive average speeds for each defined link. If the previous speed is valid, then new average is used by calculating a threshold; otherwise, the default limits are used. The average speed is calculated using the newly calculated limits. Finally, if the resulting calculation used values, then the speed and time are updated;

otherwise, if no values were used in the calculation, then the current value is retained, unless the nominal data time-out has expired. In that case, the nominal speed is reported See Appendix TBD for more details on how the speed values are calculated. The structure chart for *update archive averages* is shown in Figure 76. Descriptions of the routines called by *update archive averages* are provided in Table 92.

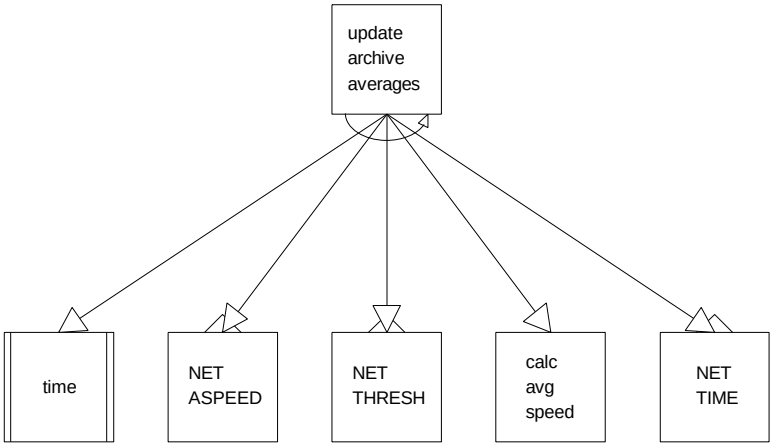


Figure 76. *update archive averages* structure chart

Table 92. Routines called by *update archive averages*

ITEM	DESCRIPTION
calc avg speed	Calculate the average speed over a link using matches which have not aged beyond their usefulness.
NET ASPEED	Macro to ease references to the archive speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET THRESH	Macro to ease references to the threshold field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET TIME	Macro to ease references to the time field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
time	C library function to get the current system time in Unix format.
update archive averages	This function updates the current average archive travel speeds in the DPF global link network.

4.2.3.3.37 archive speed data

This function loops through the entire network and writes the AVI link identifier, archive average speed, and average time to the file. The structure chart for *archive speed data* is shown in

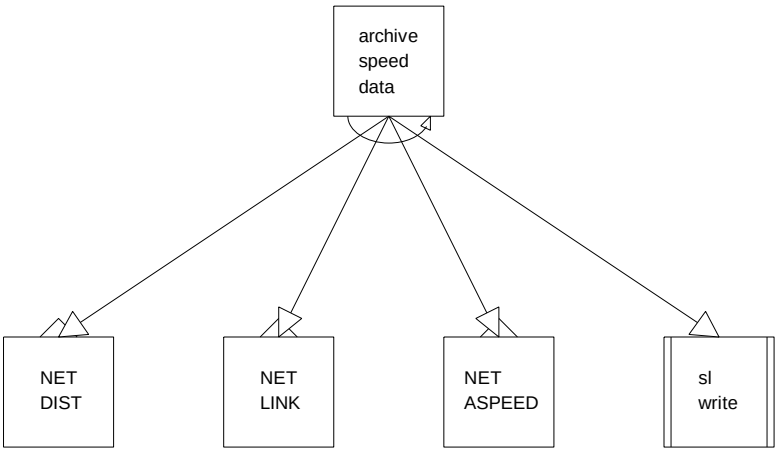


Figure 77. Descriptions of the routines called by *archive speed data* are provided in Table 93.

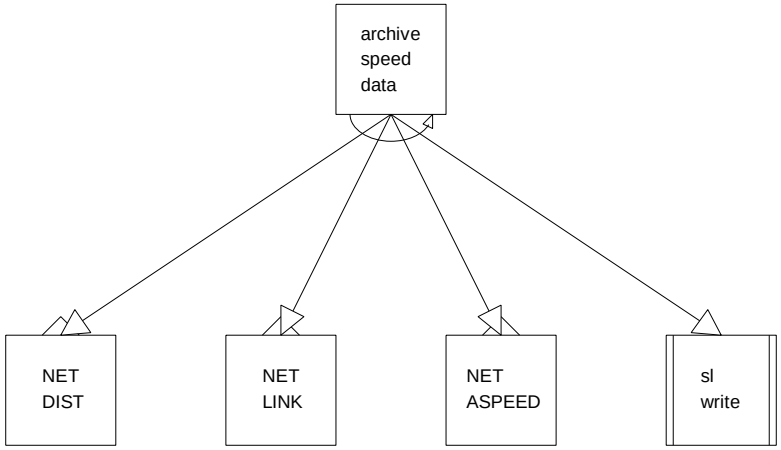


Figure 77. *archive speed data* structure chart

Table 93. Routines called by *archive speed data*

ITEM	DESCRIPTION
archive speed data	Write the current archive travel time and speed values to the archive file.
NET ASPEED	Macro to ease references to the archive speed field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.

ITEM	DESCRIPTION
NET LINK	Macro to ease references to the link field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
sl write	This function writes the passed data to the specified archive file.

4.2.3.3.38 archive quantity data

This function loops through the entire network and writes the AVI link identifier and read count for that link to the read quantity archive file. The structure chart for *archive quantity data* is shown in Figure 78. Descriptions of the routines called by *archive quantity data* are provided in Table 94.

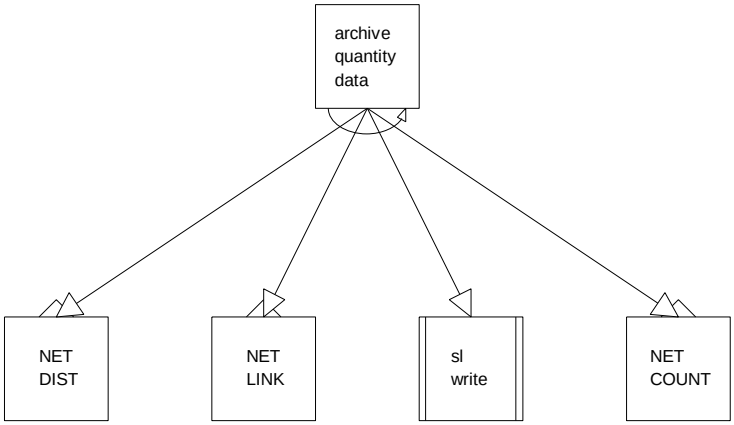


Figure 78. *archive quantity data* structure chart

Table 94. Routines called by *archive quantity data*

ITEM	DESCRIPTION
archive quantity data	Write the current read quantity values to the archive file.
NET COUNT	Macro to ease references to the count field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET DIST	Macro to ease references to the distance field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
NET LINK	Macro to ease references to the link field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
sl write	This function writes the passed data to the specified archive file.

4.2.3.3.39 purge data

This function purges obsolete data from the hash tables. Then it loops through the entire network and purges old matches from the match list for each link. The structure chart for *purge data* is shown in Figure 79. Descriptions of the routines called by *purge data* are provided in Table 95.

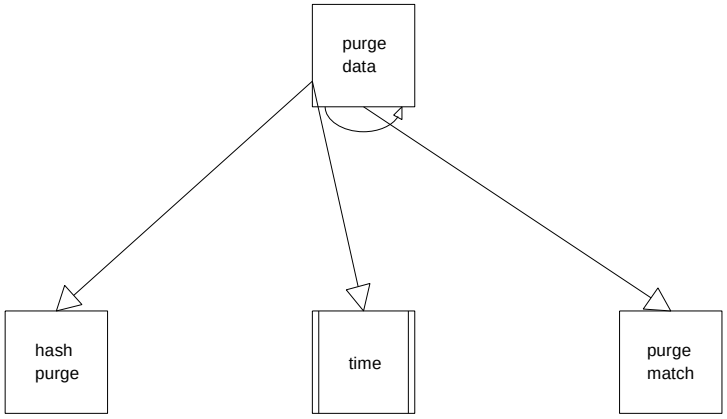


Figure 79. *purge data* structure chart

Table 95. Routines called by *purge data*

ITEM	DESCRIPTION
hash purge	Purge from the global hash table tag records older than the passed number of seconds.
purge data	Purge obsolete data from the global match and tag data structures.
purge match	Purge tag matches for a given link which have aged beyond their usefulness.
time	C library function to get the current system time in Unix format.

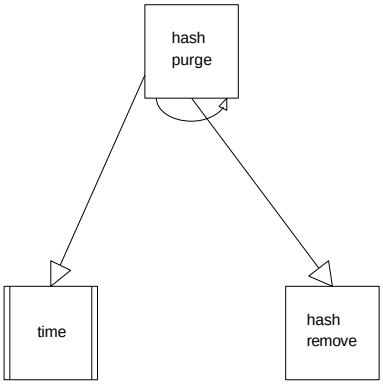


Figure 80. *hash purge* structure chart

4.2.3.3.40 hash purge

This function purges from the global hash table tag records older than the passed number of seconds. The function loops for each site. Within each site it loops through the overflow group and removes all obsolete records. Then, it loops through each site table and group within the table and removes the old entries. The structure chart for *hash purge* is shown in Figure 80. Descriptions of the routines called by *hash purge* are provided in Table 96.

Table 96. Routines called by *hash purge*

ITEM	DESCRIPTION
hash purge	Purge from the global hash table tag records older than the passed number of seconds.
hash remove	Remove the tag record indicated by the passed tag id and site index from the global tag data hash table.
time	C library function to get the current system time in Unix format.

4.2.3.3.41 purge match

This function loops through the match list for the passed site until no matches remain or some matches must be purged. Since the matches are in reverse chronological order, once one old match is found, all matches remaining in the list must also be removed. The list of valid matches is set to end with the last valid one and the obsolete match records are freed back to the system. The structure chart for *purge match* is shown in Figure 81. Descriptions of the routines called by *purge match* are provided in

Table 97.

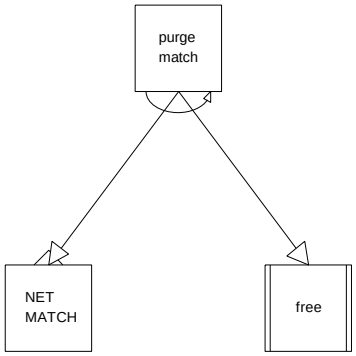


Figure 81. *purge match* structure chart

Table 97. Routines called by *purge match*

ITEM	DESCRIPTION
free	C Library Function used to free previously allocated memory and make it available for further allocation.
NET MATCH	Macro to ease references to the match field of the "two-dimensional" network structure which was dynamically allocated as a "one-dimensional" array of structures.
purge match	Purge tag matches for a given link which have aged beyond their usefulness.

4.2.3.4 Data Server Interface (AVI DSIF)

The *AVI DSIF* process provides the single point of interface between the AVI system and the Data Server. *AVI DSIF* is responsible for receiving messages from the other processes in the AVI system and directing them to the Data Server.

4.2.3.4.1 main

The AVI DSIF *main* routine is responsible for setting up the clean up routines, configuring the appropriate signals to catch and ignore, initializing the status logging and configuration data, setting up the crossing and sensor shared memory segments, connecting to the heartbeat process and the Data Server, sending periodic heartbeats to the "internal" heartbeat process, and responding to requests made by other processes within the AVI system. The structure chart for the

AVI DSIF *main* routine is shown in Figure 82. Descriptions of the routines called by AVI DSIF *main* are provided in Table 98.

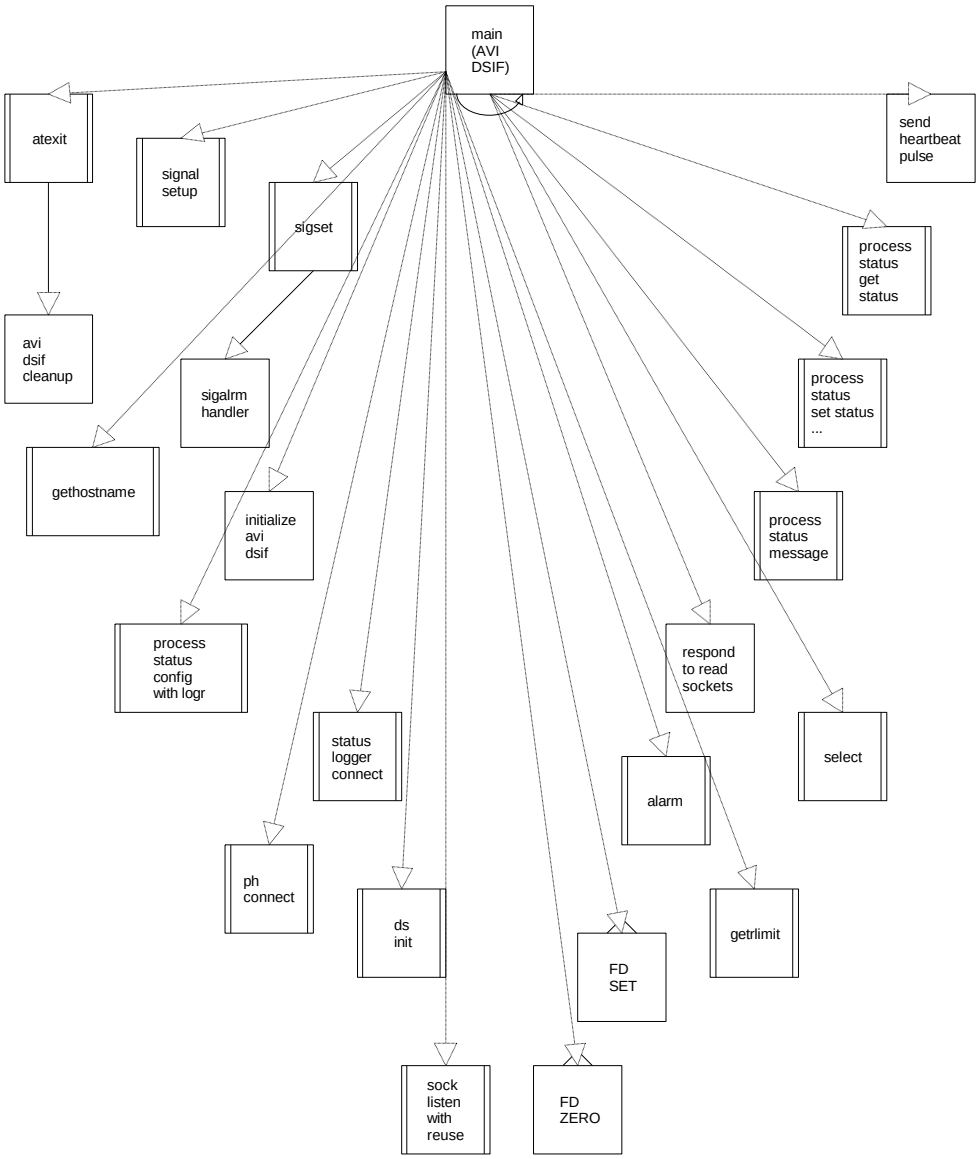


Figure 82. AVI DSIF *main* structure chart

Table 98. Routines called by AVI DSIF *main*

ITEM	DESCRIPTION
alarm	System Call used to set the alarm clock of the calling process to send a SIGALRM signal after the specified number of seconds have elapsed.
atexit	C Library Function used to register routines to be called on normal termination of a program.

ITEM	DESCRIPTION
avi dsif cleanup	This function is called on exit to do a graceful shutdown of the AVI DSIF process.
ds init	MDI Data Server library routine used to initialize the connection to the Data Server.
FD SET	C library macro to set a file descriptor in a file descriptor set.
FD ZERO	C library macro to zero a file selector set used with select().
gethostname	C library function to get the hostname on which the calling process is running.
getrlimit	C library function to get the specified system limit from the system.
initialize avi dsif	The specified configuration file is read to obtain the values to be used for the configurable items of the AVI DSIF process.
main (AVI DSIF)	This is the main routine for the AVI Data Server Interface Program (DSIF).
ph connect	MDI Process Heartbeat routine used to connect to the specified process-level heartbeat service. The host name and service name are used to make the connection.
process status config with logr	This routine is sets up the connection to the status logger used by the calling program.
process status get status	MDI Process Status routine used to obtain the most severe process-level status. This is an aggregation of the status for each of the status types defined for the process.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
respond to read sockets	This function loops through the ready sockets accepting connect requests and receiving status messages as appropriate.
select	C Library Function used to multiplex synchronous I/O. The list of file descriptors for reading, writing, and receiving exceptions are examined and any file descriptors that are ready for reading, writing, or have an exceptional condition pending are identified.
send heartbeat pulse	Sends the heartbeat pulse message to the AVI project heartbeat process.
sigalrm handler	This is the signal handler for the SIGALRM signal. The signal is used to indicate when the process level heartbeat should be sent to the AVI HEARTBEAT process.
signal setup	This function sets up the signal handler for all signals that are not currently handled within the calling process.
sigset	C Library Function used to modify the disposition of a signal. The signal can be caught, ignored, or returned to the default disposition.

ITEM	DESCRIPTION
sock listen with reuse	MDI Common Socket routine used to set up a socket to listen for connections and to make the socket address reusable.
status logger connect	This function connects to the status logger.

4.2.3.4.2 avi dsif cleanup

This routine is called when the *AVI DSIF* process performs a normal termination. This routine performs the necessary housekeeping chores to cause a graceful exit of the AVI DSIF process. The structure chart for the *avi dsif cleanup* routine is shown in Figure 83. A description of the routines called by *avi dsif cleanup* is provided in Table 99.

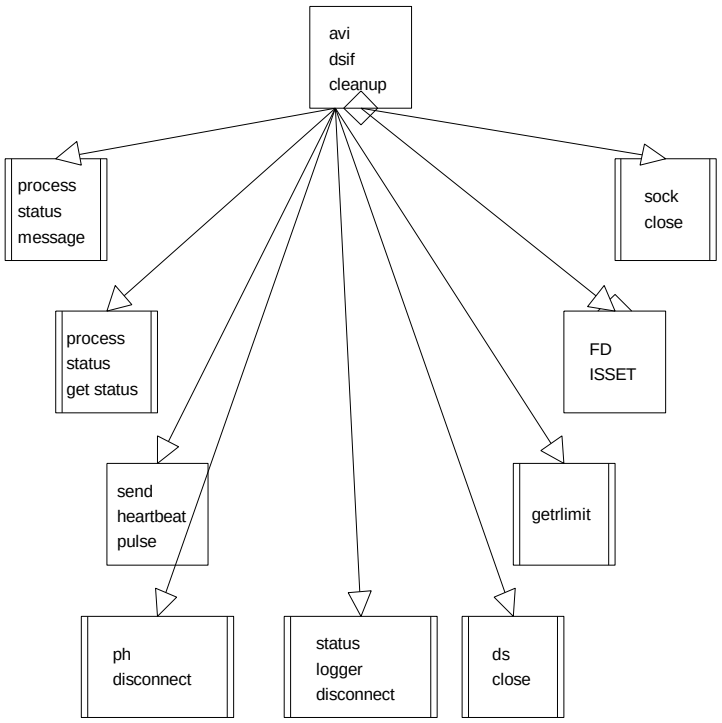


Figure 83. *avi dsif cleanup* structure chart

Table 99. Routines called by *avi dsif cleanup*

ITEM	DESCRIPTION
avi dsif cleanup	This function is called on exit to do a graceful shutdown of the AVI DSIF process.
ds close	MDI Data Server routine used to close the connection to the Data Server.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
getrlimit	C library function to get the specified system limit from the system.

ITEM	DESCRIPTION
ph disconnect	MDI Process Heartbeat routine used to disconnect from the process-level heartbeat service.
process status get status	MDI Process Status routine used to obtain the most severe process-level status. This is an aggregation of the status for each of the status types defined for the process.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
send heartbeat pulse	Sends the heartbeat pulse message to the AVI project heartbeat process.
sock close	MDI Socket routine used to close the specified socket connection.
status logger disconnect	This function disconnects from the status logger.

4.2.3.4.3 send heartbeat pulse

This routine is invoked periodically whenever the socket selection is interrupted by an alarm signal. This routine is responsible for sending the process-level heartbeat message to the project-level heartbeat process. The structure chart for *send heartbeat pulse* is shown in Figure 84. The

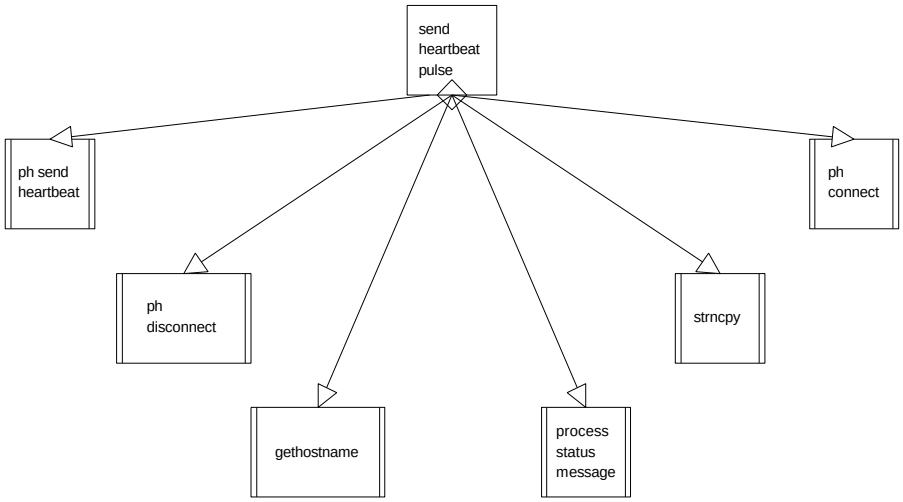


Figure 84. *send heartbeat pulse* structure chart

descriptions of the routines called by *send heartbeat pulse* are contained in Table 100.

Table 100. Routines called by *send heartbeat pulse*

ITEM	DESCRIPTION
gethostname	C library function to get the hostname on which the calling process is running.

ITEM	DESCRIPTION
ph connect	MDI Process Heartbeat routine used to connect to the specified process-level heartbeat service. The host name and service name are used to make the connection.
ph disconnect	MDI Process Heartbeat routine used to disconnect from the process-level heartbeat service.
ph send heartbeat	MDI Process Heartbeat routine used to send the specified status value to the heartbeat service configured by the ph connect call.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
send heartbeat pulse	Sends the heartbeat pulse message to the AVI project heartbeat process.
strncpy	C Library Function to copy a specified number of characters from a source string to a destination string.

4.2.3.4.4 sigalrm handler

The *sigalrm handler* routine is invoked whenever the AVI DSIF process receives an alarm signal from the process alarm clock. The routine sets a flag indicating a heartbeat message needs to be sent to the Data Server and then sets the alarm clock again so the routine will be invoked. *sigalrm handler* calls only *alarm* so no structure chart is provided.. A description of the routines called by *sigalrm handler* is provided in Table 101.

Table 101. Routines called by *sigalrm handler*

ITEM	DESCRIPTION
alarm	System Call used to set the alarm clock of the calling process to send a SIGALRM signal after the specified number of seconds have elapsed.
sigalrm handler	This is the signal handler for the SIGALRM signal. The signal is used to indicate when the process level heartbeat should be sent to the AVI HEARTBEAT process.

4.2.3.4.5 initialize avi dsif

This routine is called to read the AVI DSIF configuration file and set up configuration information for the entire process. *initialize avi dsif* calls only *load cfg data* so no structure chart is provided. Descriptions of the routines called by *initialize avi dsif* are contained in Table 102. Configurable items for the AVI DSIF process are described in Table 103.

Table 102. Routines called by *initialize avi dsif*

ITEM	DESCRIPTION
initialize avi dsif	The specified configuration file is read to obtain the values to be used for the configurable items of the AVI DSIF process.
load cfg data	This function loads the MDI and AVI configuration data. Data is obtained from configuration files and system function calls.

Table 103. AVI DSIF configuration items

CONFIGURATION ITEM	DESCRIPTION	OPT
AVI_SHM_BASE	The name of the constant or an integer value indicating the starting base for the AVI shared memory segments.	N
DATASERVER_HOST_NAME	The host name where the Data Server process resides.	Y
DATASERVER_SERVICE_NAME	The name of the service provided by the Data Server process.	N
HEARTBEAT_HOST_NAME	The host name where the AVI project-level heartbeat process resides.	Y
HEARTBEAT_PULSE	The periodic time value for sending the heartbeat to the AVI project-level heartbeat process. This is specified in seconds.	Y
HEARTBEAT_SERVICE_NAME	The name of the service provided by the AVI project-level heartbeat process.	N
NUM_SHMEM_SEGMENTS	The total number of shared memory segments used by the AVI subsystem.	N
SERVICE_NAME	The name of the service provided by the AVI DSIF process.	N
STATUS_LOGGER_HOST_NAME	The host name where the AVI subsystem status logger process resides	Y
STATUS_LOGGER_SERVICE_NAME	The name of the service provided by the AVI subsystem status logger process.	N

4.2.3.4.6 respond to read sockets

The *respond to read sockets* routine is heart of the AVI DSIF process. This routine is called when there is data pending on any of the sockets that are connected to the process. This data could be a connection request to the AVI DSIF process, a message being sent to the AVI DSIF process by another process already connected, or it could be an indication of a process that has disconnected from the AVI DSIF process. When a connection request is received, the process immediately accepts the connection. If a message is being sent then the message is read from the active socket and is then dispatch to the Data Server according the type of message received. If a connected process disconnects from the AVI DSIF process the socket connection from the AVI DSIF process to the disconnected process is closed and removed from the list of active sockets. Errors that occur are logged to the AVI subsystem status log. The structure chart for the *respond to read sockets* is shown in Figure 85. A description of the routines called by *respond to read sockets* is provided in Table 104.

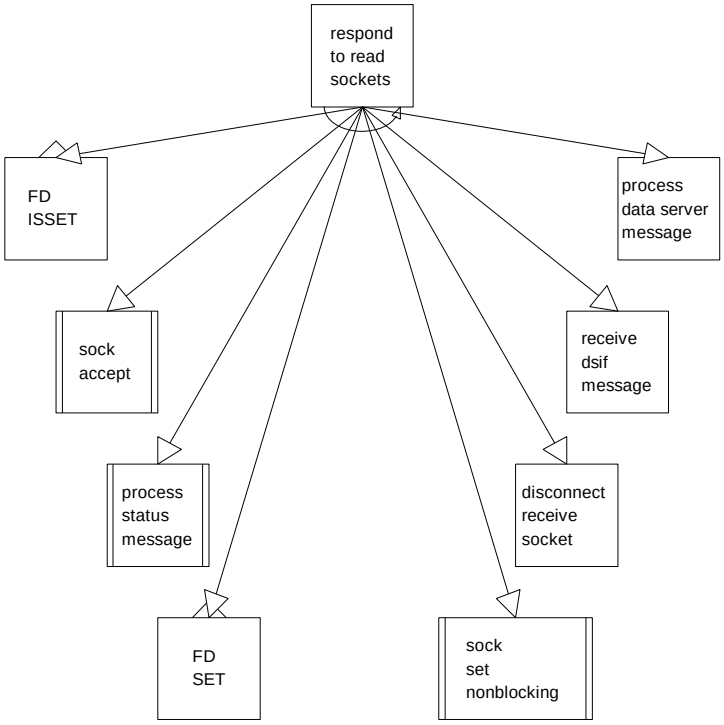


Figure 85. *respond to read sockets* structure chart

Table 104. Routines called by *respond to read sockets*

ITEM	DESCRIPTION
disconnect receive socket	On success the connection to the specified socket is shutdown and closed and the select set is modified to have the specified socket descriptor cleared from the set.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
FD SET	C library macro to set a file descriptor in a file descriptor set.
process Data Server message	This function extracts the data from the DSIF message, reads any additional data that may have been sent by the client, and sends this data to the Data Server.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
receive dsif message	This routine reads an AVI DSIF message from the specified socket.
respond to read sockets	This function loops through the ready sockets accepting connect requests and receiving status messages as appropriate.
sock accept	MDI Socket routine that accepts connections on the specified listen socket.

ITEM	DESCRIPTION
sock set nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

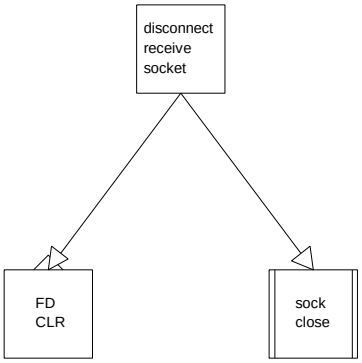


Figure 86. *disconnect receive socket* structure chart

4.2.3.4.7 disconnect receive socket

The *disconnect receive socket* routine shuts down the active socket and removes the socket from the list of sockets the AVI DSIF process listens to for data. The structure chart for *disconnect receive socket* is shown in Figure 86. The descriptions of the routines called by *disconnect receive socket* are contained in

Table 105.

Table 105. Routines called by *disconnect receive socket*

ITEM	DESCRIPTION
disconnect receive socket	On success the connection to the specified socket is shutdown and closed and the select set is modified to have the specified socket descriptor cleared from the set.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
FD SET	C library macro to set a file descriptor in a file descriptor set.
process Data Server message	This function extracts the data from the DSIF message, reads any additional data that may have been sent by the client, and sends this data to the Data Server.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
receive dsif message	This routine reads an AVI DSIF message from the specified socket.
respond to read sockets	This function loops through the ready sockets accepting connect requests and receiving status messages as appropriate.
sock accept	MDI Socket routine that accepts connections on the specified listen socket.

ITEM	DESCRIPTION
sock set nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

4.2.3.4.8 receive dsif message

The *receive dsif message* routine reads the message from the active socket and places in the received message buffer. *receive dsif message* calls only *sock readn* so no structure chart is provided. The descriptions of the routines called by *receive dsif message* are contained in Table 106.

Table 106. Routines called by *receive dsif message*

ITEM	DESCRIPTION
receive dsif message	This routine reads an AVI DSIF message from the specified socket.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.

4.2.3.4.9 process Data Server message

This function extracts the data received from another AVI process and processes it based on the type. The message may be a heartbeat or link data and these are sent by calling *ds send heartbeat* or *send write link message*, respectively. Any other code is ignored and a warning is logged. If an error occurs during processing, then the socket to Data Server is closed and reinitialized. The structure chart for *process Data Server message* is shown in Figure 87. The descriptions of the routines called by *process Data Server message* are contained in Table 107.

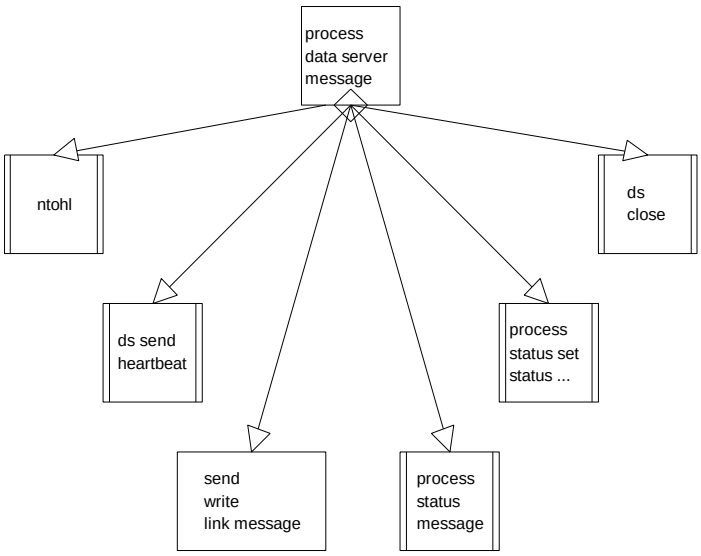


Figure 87. *process Data Server message* structure chart

Table 107. Routines called by *process Data Server message*

ITEM	DESCRIPTION
ds close	MDI Data Server routine used to close the connection to the Data Server.
ds send heartbeat	MDI Data Server routine used to send the subsystem-level heartbeat message to the Data Server. The heartbeat status is the overall status for the subsystem.
ntohl	Network Function used to convert between network and host byte order.
process Data Server message	This function extracts the data from the DSIF message, reads any additional data that may have been sent by the client, and sends this data to the Data Server.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
process status set status ...	This function is used to set the value associated with the specified process status status type.
send write link message	This function extracts the specific information from the received message and calls the Data Server write link routine.

4.2.3.4.10 send write link message

This function takes the passed message data and resizes the working data structure to accommodate the message data size. It then reads the data from the client socket. Next, it calls the Data Server library routine *ds write lane data* to write the lane data message to Data Server. Finally, it reads the return status from the Data Server socket. The structure chart for *send write link message* is shown in Figure 88. The descriptions of the routines called by *send write link message* are contained in Table 108.

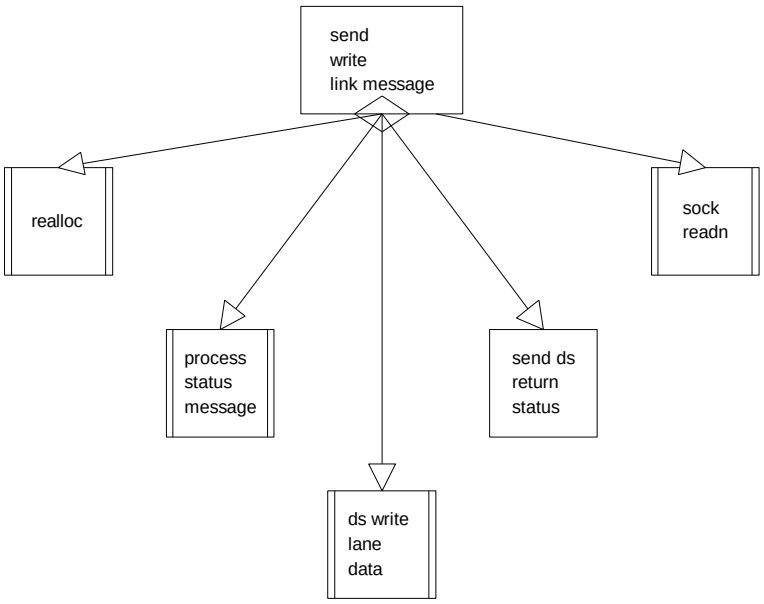


Figure 88. *send write link message* structure chart

Table 108. Routines called by *send write link message*

ITEM	DESCRIPTION
ds write lane data	This function writes the lane data message to the Data Server.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.
realloc	C Library Function to reallocate a memory block. The reallocation leaves the memory which is not affected by the size change unmodified.
send ds return status	This function sends the return status to the specified socket descriptor which should be the same one on which the request was made.
send write link message	This function extracts the specific information from the received message and calls the Data Server write link routine.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.

4.2.3.4.11 send ds return status

This function sends the return status to the specified socket descriptor. This socket descriptor should be the same socket on which the request was made. It builds the message and calls *send ds return message* to actually send the data out on the socket. The structure chart for *send ds return status* is shown in Figure 89. The descriptions of the routines called by *send ds return status* are contained in

Table 109.

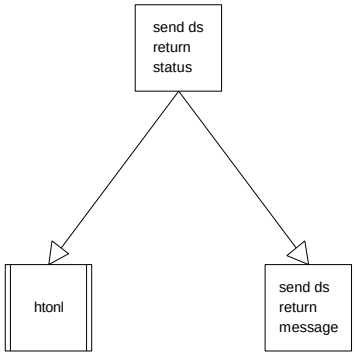


Figure 89. *send ds return status* structure chart

Table 109. Routines called by *send ds return status*

ITEM	DESCRIPTION
htonl	Network function used to convert from host to network byte formats.
send ds return message	This function sends the return message to the specified socket descriptor which should be the same one on which the request was made.
send ds return status	This function sends the return status to the specified socket descriptor which should be the same one on which the request was made.

4.2.3.4.12 send ds return message

This function sends the return message to the specified socket descriptor, which should be the same one on which the request was made. *send ds return message* calls only *sock writen*, so no structure chart is provided. Descriptions of the routines called by *send ds return message* are provided in Table 110.

Table 110. Routines called by *send ds return message*

ITEM	DESCRIPTION
send ds return message	This function sends the return message to the specified socket descriptor which should be the same one on which the request was made.
sock writen	MDI Socket routine used to write a specified number of bytes to a specified socket.

4.2.3.5 AVI DSIF Library Routines

Processes communicate to using the routines in this library and the MDI common heartbeat routines. The routines needed to send heartbeats to AVI DSIF are described in MDI Common Code

Software Design Document and are not a part of this library. The AVI DSIF library contains routines that are used to interact with the AVI DSIF process for operations other than heartbeating. This library includes a function to send link data to and receive status from the AVI DSIF process. These routines are discussed in more detail in the following subsections.

4.2.3.5.1 avi dsif send link write request

This function packages the passed link data into the correct format and sends it to the AVI DSIF process. First, it verifies the socket connection and reestablishes it if not connected. Next, the message data is formatted before sending to AVI DSIF. Next, the data is send to AVI DSIF. Finally, the code waits for the status from the send. The structure chart for *avi dsif send link write request* is shown in Figure 90. The descriptions of the routines called by *avi dsif send link write request* are contained in Table 111.

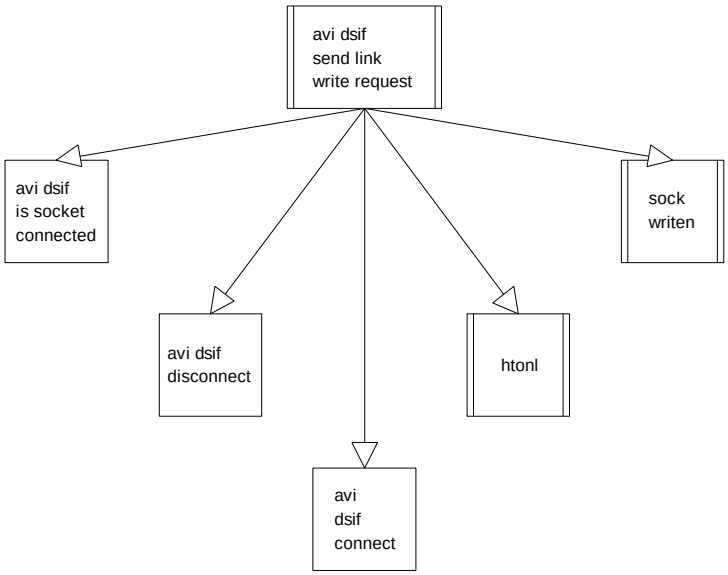


Figure 90. *avi dsif send link write request* structure chart

Table 111. Routines called by *avi dsif send link write request*

ITEM	DESCRIPTION
avi dsif connect	This function is used to connect to the AVI Data Server Interface (DSIF) process specified by the service name passed to this routine.
avi dsif disconnect	Close the socket connection to AVI DSIF.
avi dsif is socket connected	This function is used to determine if there is a valid socket connection with the DSIF process.
avi dsif send link write request	Package the AVI link speed or time data into the correct format and send it to the Data Server.
htonl	Network function used to convert from host to network byte formats.

ITEM	DESCRIPTION
sock writen	MDI Socket routine used to write a specified number of bytes to a specified socket.

4.2.3.5.2 avi dsif is socket connected

This function determines if there is a valid socket connection to the AVI DSIF process. The function sets the socket descriptor in a read descriptor set. The socket is supposed to be write only and should not be set after the select. If it is, then the socket is not connected. The structure chart for *avi dsif is socket connected* is shown in Figure 91. The descriptions of the routines called by *avi dsif is socket connected* are contained in Table 112.

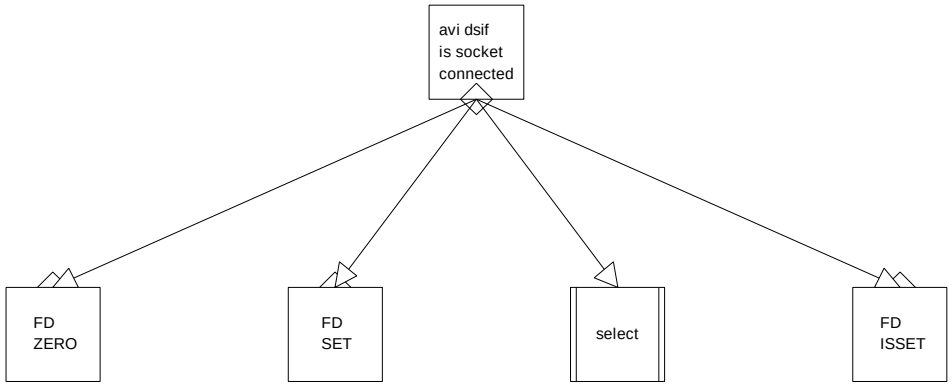


Figure 91. *avi dsif is socket connected* structure chart

Table 112. Routines called by *avi dsif is socket connected*

ITEM	DESCRIPTION
avi dsif is socket connected	This function is used to determine if there is a valid socket connection with the DSIF process.
FD ISSET	C library macro to check to see if a given socket descriptor in a set is set (requires processing).
FD SET	C library macro to set a file descriptor in a file descriptor set.
FD ZERO	C library macro to zero a file selector set used with select().
select	C Library Function used to multiplex synchronous I/O. The list of file descriptors for reading, writing, and receiving exceptions are examined and any file descriptors that are ready for reading, writing, or have an exceptional condition pending are identified.

4.2.3.5.3 avi dsif disconnect

This function disconnects from the AVI DSIF socket. *avi dsif disconnect* calls only *sock close* so no structure chart is provided. Descriptions of the routines called by *avi dsif disconnect* are provided in Table 113.

Table 113. Routines called by *avi dsif disconnect*

ITEM	DESCRIPTION
avi dsif disconnect	Close the socket connection to AVI DSIF.
sock close	MDI Socket routine used to close the specified socket connection.

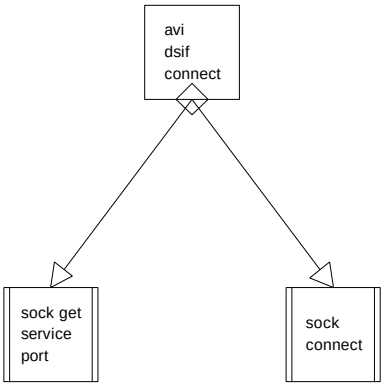


Figure 92. *avi dsif connect* structure chart

4.2.3.5.4 avi dsif connect

This routine is responsible for establishing the communications path to the Data Server from the calling process. This is accomplished by obtaining the port number associated with the specified service and then connecting a socket to the port on the specified host computer. The structure chart for *avi dsif connect* is shown in Figure 92. The descriptions of the routines called by *avi dsif connect* are contained in

Table 114.

Table 114. Routines called by *avi dsif connect*

ITEM	DESCRIPTION
avi dsif connect	This function is used to connect to the AVI Data Server Interface (DSIF) process specified by the service name passed to this routine.
sock connect	MDI Socket routine used to create a socket connection to the specified host and port.
sock get service port	MDI Socket routine that returns the port number associated with the specified service name.

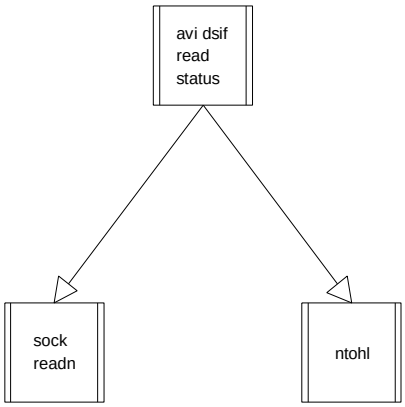


Figure 93. *avi dsif read status* structure chart

4.2.3.5.5 avi dsif read status

This function reads the return status from a Data Server request (write link data). The code loops waiting for a successful read. Once received, the data is converted from network order to the usual byte ordering. The structure chart for *avi dsif read status* is shown in Figure 93. The descriptions of the routines called by *avi dsif read status* are contained in

Table 115.

Table 115. Routines called by *avi dsif read status*

ITEM	DESCRIPTION
avi dsif read status	This function reads the return status from the Data Server request.
ntohl	Network Function used to convert between network and host byte order.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.

4.2.3.6 AVI GUI

4.2.3.7 AVI Common Libraries

This section contains descriptions of the utility functions contained in the AVI unique libraries that are part of AVI.

4.2.3.8 AVI Interface

The AVI interface routines provide access to the AVI shared memory and send configuration change messages to DCM for the AVI GUI.

4.2.3.8.1 avi alloc site status

This function allocates a site status structure for the GUI. The allocated memory includes space for both the detailed status data and the tag data. The structure chart for *avi alloc site status* is shown in Figure 94. Descriptions of the routines called by *avi alloc site status* are provided in Table 116.

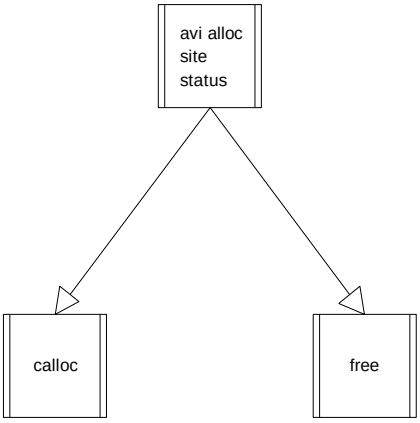


Figure 94. *avi alloc site status*
structure chart

Table 116. Routines called by *avi alloc site status*

ITEM	DESCRIPTION
avi alloc site status	Allocate a site status structure for the GUI including both status and tag data.
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
free	C Library Function used to free previously allocated memory and make it available for further allocation.

4.2.3.8.2 avi dealloc site status

This function deallocates a site's status structure for the GUI including both the detailed site status data and tag data. avi dealloc site status calls only *free*, so no structure chart is provided. Descriptions of the routines called by *avi dealloc site status* are provided in Table 117.

Table 117. Routines called by *avi dealloc site status*

ITEM	DESCRIPTION
avi dealloc site status	Deallocate a site status structure for the GUI including both the detailed site status and tag data.
free	C Library Function used to free previously allocated memory and make it available for further allocation.

4.2.3.8.3 avi get detailed site status

This function reads the global site status and tag data from shared memory and stores it in the passed structure. *avi get detailed site status* calls only *read segment element* so no structure chart is provided. Descriptions of the routines called by *avi get detailed site status* are provided in Table 118.

Table 118. Routines called by *avi get detailed site status*

ITEM	DESCRIPTION
avi get detailed site status	Read the global site status and tag data from shared memory for the GUI.
read segment element	MDI Shared Memory Manager routine to read the contents of one element of the specified shared memory segment. The contents are stored in a memory area allocated by the caller.

4.2.3.8.4 avi get site status

This function reads the global site status for the indexed site from shared memory and sets the site status into the passed variable. *avi get site status* calls only *read segment element* so no structure chart is provided. Descriptions of the routines called by *avi get site status* are provided in Table 119.

Table 119. Routines called by *avi get site status*

ITEM	DESCRIPTION
avi get site status	Read the global site status for the indexed site from shared memory.
read segment element	MDI Shared Memory Manager routine to read the contents of one element of the specified shared memory segment. The contents are stored in a memory area allocated by the caller.

4.2.3.8.5 send dcm command

This function sends a 4-byte configuration change command to DCM. First, the data is written to the socket using *avi sock write* which will automatically open the socket if necessary. Next, the socket is set non-blocking so that the result may be read without locking up. The status is read from the socket. This is repeated at one second intervals for up to ten tries, if necessary. The status of the command is returned. The structure chart for *send dcm command* is shown in Figure 95. Descriptions of the routines called by *send dcm command* are provided in Table 120.

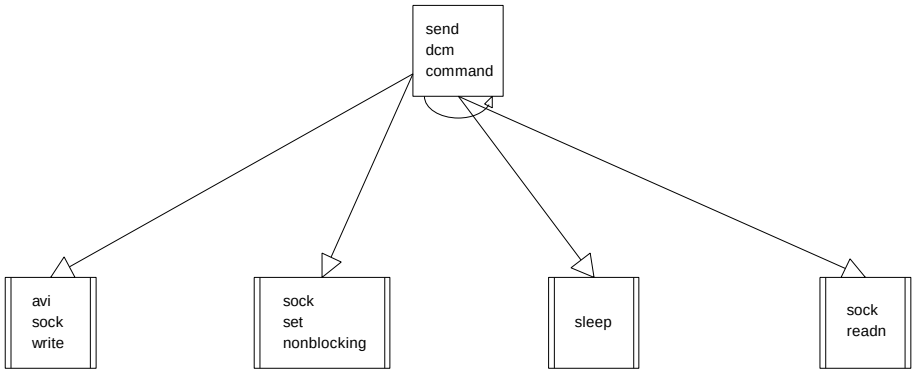


Figure 95. *send dcm command* structure chart

Table 120. Routines called by *send dcm command*

ITEM	DESCRIPTION
avi sock write	Write to a socket with retry.
send dcm command	Send a configuration change command to DCM.
sleep	C library function to suspend a process for the specified number of seconds.
sock readn	MDI Socket routine that reads a specified number of bytes from the specified socket.
sock set nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

4.2.3.9 AVI Configuration

The AVI Configuration library provides a common routine for reading AVI and MDI configuration data for the AVI processes.

4.2.3.9.1 load cfg data

This function loads the configuration data. The configuration data is obtained from configuration files and system function calls. The data is returned to the caller in a structure that is passed as a parameter. The data in the structure is only valid if the function returns a successful condition.

The function calls *init cfg data* to initialize the fields of the structure. Next, the configuration data is loaded from the MDI configuration file and the AVI configuration file. Once these values are obtained, the *cfg get value* routine is used to get the specific values and place them into the AVI configuration structure. Port numbers are determined by using the Unix *getservbyname* utility. The structure chart for *load cfg data* is shown in Figure 96. Descriptions of the routines called by *load cfg data* are provided in Table 121.

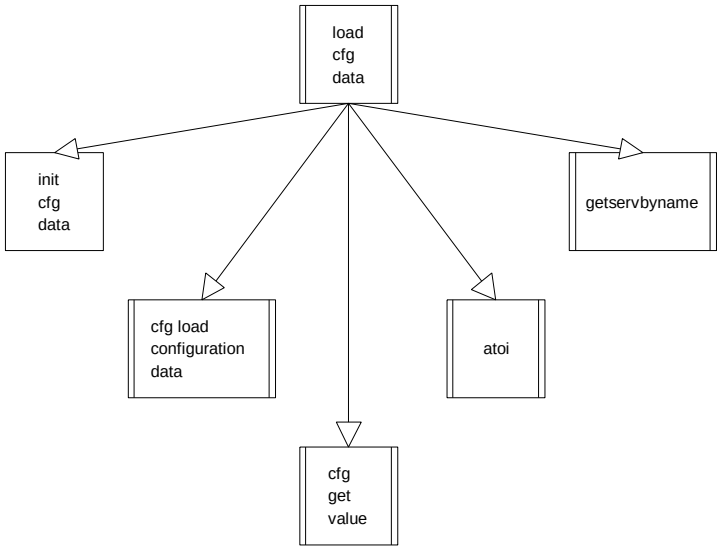


Figure 96. *load cfg data* structure chart

Table 121. Routines called by *load cfg data*

ITEM	DESCRIPTION
atoi	C Library Function to convert an ASCII string to an integer value.
cfg get value	MDI Configuration File routine used to return the value of the specified configuration name.
cfg load configuration data	MDI Configuration File routine used to read the configuration name-value pairs from the specified configuration file. These name-value pairs are loaded into memory so they can be accessed on demand by the calling program.
getservbyname	C library function which searches the system for an entry which matches the passed name returning the structure associated with the entry.
init cfg data	This function initializes the passed configuration data structure.
load cfg data	This function loads the MDI and AVI configuration data. Data is obtained from configuration files and system function calls.

4.2.3.9.2 init cfg data

This function initializes the fields of the passed AVI configuration structure to default values. Neither a structure chart nor a description of routines called table is provided as *init cfg data* calls no subroutines.

4.2.3.10 AVI Heartbeat

The AVI Heartbeat library provides a routine to send the "internal" heartbeat to the AVI heartbeat process. The AVI heartbeat process sends the summary heartbeat to DSIF which in turn sends it to Data Server.

4.2.3.10.1 avi send heartbeat

This function sends an "internal" heartbeat to the AVI heartbeat process. The routine first tries to send the heartbeat; if it fails, it tries to disconnect, reconnect and retransmit the heartbeat. The structure chart for *avi send heartbeat* is shown in Figure 97. Descriptions of the routines called by *avi send heartbeat* are provided in Table 122.

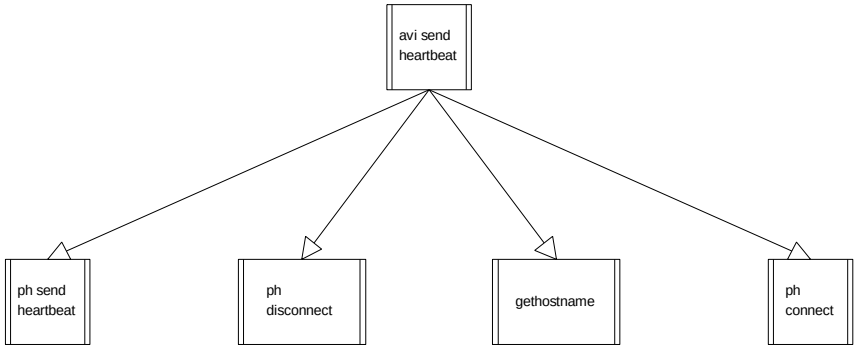


Figure 97. *avi send heartbeat* structure chart

Table 122. Routines called by *avi send heartbeat*

ITEM	DESCRIPTION
avi send heartbeat	This function sends the heartbeat to the AVI heartbeat process with automatic reconnection and error checking.
gethostname	C library function to get the hostname on which the calling process is running.
ph connect	MDI Process Heartbeat routine used to connect to the specified process-level heartbeat service. The host name and service name are used to make the connection.
ph disconnect	MDI Process Heartbeat routine used to disconnect from the process-level heartbeat service.
ph send heartbeat	MDI Process Heartbeat routine used to send the specified status value to the heartbeat service configured by the ph connect call.

4.2.3.11 AVI Data

The AVI Data library provides routines to load the data files that are used by AVI. This includes getting the AVI link identifiers from the common set for MDI, reading the AVI site data file, and functions to look up site/link information.

4.2.3.11.1 avi get avi data

This function loads the AVI data from the data files into the AVI data data structure passed by the caller. File access and memory allocations are performed by this function The structure chart for *avi get avi data* is shown in Figure 98. Descriptions of the routines called by *avi get avi data* are provided in Table 123.

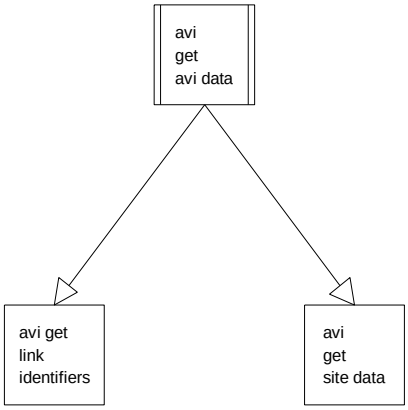


Figure 98. *avi get avi data* structure chart

Table 123. Routines called by *avi get avi data*

ITEM	DESCRIPTION

ITEM	DESCRIPTION
avi get avi data	This function loads the AVI link and site data from files into dynamically allocated data structures.
avi get link identifiers	This function obtains the AVI link identifiers and returns an array which contains them.
avi get site data	Read the site data file and store the data into dynamically allocated memory.

4.2.3.11.2 avi get link identifiers

This function uses MDI common library routines to load the AVI link identifiers from the MDI database returning the data in an array allocated by the common routines. The structure chart for *avi get link identifiers* is shown in Figure 99. Descriptions of the routines called by *avi get link identifiers* are provided in Table 124.

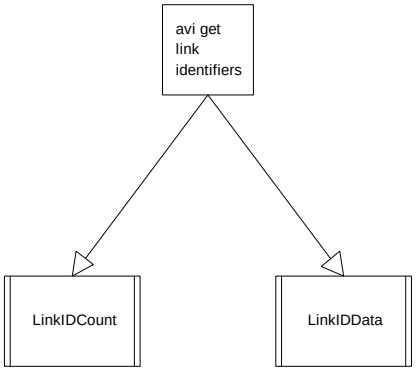


Figure 99. *avi get link identifiers* structure chart

Table 124. Routines called by *avi get link identifiers*

ITEM	DESCRIPTION
avi get link identifiers	This function obtains the AVI link identifiers and returns an array which contains them.
LinkIDCount	Get the number of TransGuide link IDs which are used by the passed link type (e.g., AVI).
LinkIDData	Get the TransGuide link ID data for the passed link type (e.g., AVI).

4.2.3.11.3 avi get site data

This function opens the site data file, reads the record count, allocates the memory for the AVI site data, and calls *avi read site data file* to load the site data from the file and store it in the allocated memory. The structure chart for *avi get site data* is shown in Figure 100. Descriptions of the routines called by *avi get site data* are provided in Table 125.

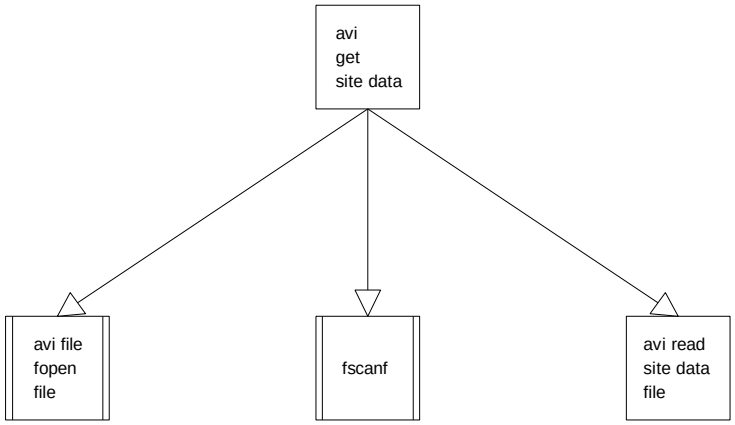


Figure 100. *avi get site data* structure chart

Table 125. Routines called by *avi get site data*

ITEM	DESCRIPTION
avi file fopen file	Open file identified by passed filename and path with error checking.
avi get site data	Read the site data file and store the data into dynamically allocated memory.
avi read site data file	This function reads the site data file into the passed array of site data structures.
fscanf	C library function to scan an input stream into the referenced variables according to the specified format string.

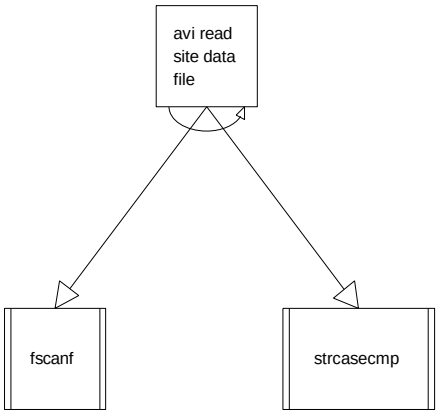


Figure 101. *avi read site data file* structure chart

4.2.3.11.4 avi read site data file

This function reads the records from the site data file into an array of site data data structures. The data read is the site (source) number, site identifier, and enable flag. Records are read until the count of sites is reached. The structure chart for *avi read site data file* is shown in Figure 101. Descriptions of the routines called by *avi read site data file* are provided in

Table 126.

Table 126. Routines called by *avi read site data file*

ITEM	DESCRIPTION
avi read site data file	This function reads the site data file into the passed array of site data structures.
fscanf	C library function to scan an input stream into the referenced variables according to the specified format string.
strcasecmp	C library to do a case insensitive compare between two strings.

4.2.3.11.5 lookup TGLinkID idx

This function looks up the index of a TransGuide link id using the passed list of link identifiers. It returns the located index or -1 if it is not found. *lookup TGLinkID idx* calls only *strcmp* so no structure chart is provided. Descriptions of the routines called by *lookup TGLinkID idx* are provided in Table 127.

Table 127. Routines called by *lookup TGLinkID idx*

ITEM	DESCRIPTION
lookup TGLinkID idx	Look up the TransGuide link id index using its identifier.
strcmp	C library to do a case sensitive compare between two strings.

4.2.3.11.6 lookup site by name

This function looks up the index of the AVI site using the passed site name and site data. It returns the located index or -1 if it is not found. *lookup site by name* calls only *strcmp* so no structure chart is provided. Descriptions of the routines called by *lookup site by name* are provided in Table 128.

Table 128. Routines called by *lookup site by name*

ITEM	DESCRIPTION
lookup site by name	Look up the index for an AVI site using the site's name.
strcmp	C library to do a case sensitive compare between two strings.

4.2.3.11.7 lookup site by num

This function looks up the site index for a field site using it hardware "source number." This is a number that is arbitrarily assigned during hardware installation. It is mapped through a data file which is read into the global AVI site data that must be passed in to this routine. Neither a structure chart nor a description of routines called table is provided as *lookup site by num* calls no subroutines.

4.2.3.12 AVI File

The AVI File library provides functions to open a file with pathname assembly and error checking and to get the number of entries from the first record of an AVI data file.

4.2.3.12.1 avi file fopen file

This function builds a pathname from the passed path and file components, opens the file with the flags passed by the caller, and obtains information about the file. The file pointer is returned by the function and the file information is returned in an argument. The structure chart for *avi file fopen file* is shown in Figure 102. Descriptions of the routines called by *avi file fopen file* are provided in Table 129.

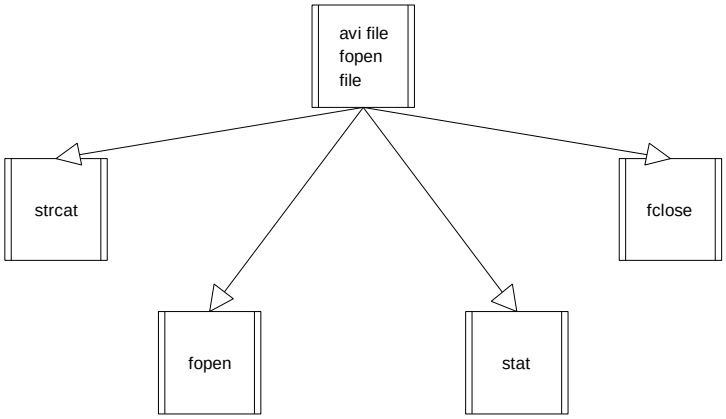


Figure 102. *avi file fopen file* structure chart

Table 129. Routines called by *avi file fopen file*

ITEM	DESCRIPTION
avi file fopen file	Open file identified by passed filename and path with error checking.
fclose	C library function to close a stream.
fopen	C library function to open a stream file.
stat	C library file to obtain information about the specified file.
strcat	C library function which concatenates a copy of the second argument to the first.

4.2.3.12.2 avi file get num entries

This function uses the passed pathname to open an AVI data file and read the first record which by convention contains the number of entries in the file. The number of entries in the file is returned or -1 if an error occurred. The structure chart for *avi file get num entries* is shown in Figure 103. Descriptions of the routines called by *avi file get num entries* are provided in Table 130.

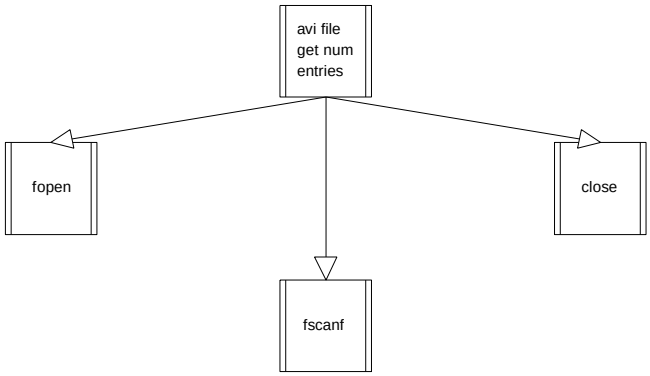


Figure 103. *avi file get num entries* structure chart

Table 130. Routines called by *avi file get num entries*

ITEM	DESCRIPTION
avi file get num entries	Get the number of entries in a configuration file from the first record of the file.
close	C library function to close a file or device.
fopen	C library function to open a stream file.
fscanf	C library function to scan an input stream into the referenced variables according to the specified format string.

4.2.3.13 AVI Util

The AVI Util library contains routines that provide a variety of useful functions. Functions can attach to a shared memory segment, write data to a socket with retry, initialize and handle signals, and initialize a DCM status structure.

4.2.3.13.1 avi create segment

This function checks to see if the shared memory segment exists and is of the correct size. If it already exists with the correct size, the it attaches to the segment. If it does not exist, then it is created. If it exists but is not the correct size, then no other action is taken and an error is returned. The structure chart for *avi create segment* is shown in Figure 104. Descriptions of the routines called by *avi create segment* are provided in Table 131.

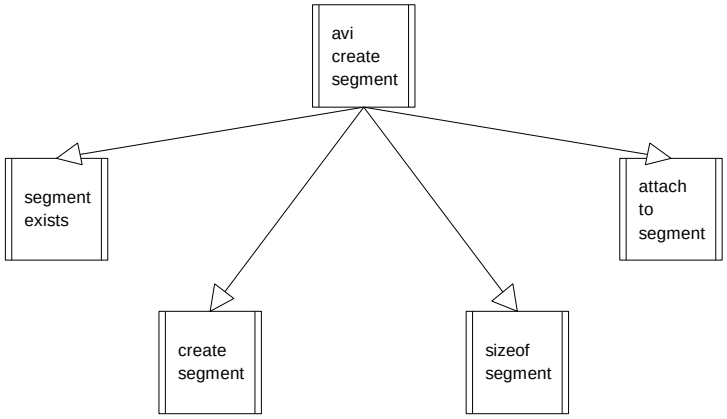


Figure 104. *avi create segment* structure chart

Table 131. Routines called by *avi create segment*

ITEM	DESCRIPTION
attach to segment	MDI Shared Memory Manager routine used to attach the calling process to the specified shared memory segment.
avi create segment	This function connects to or creates a shared memory segment with error checking.
create segment	MDI Shared Memory Manager routine used to create a shared memory segment of the specified size. The shared memory segment is automatically attached to the calling process.
segment exists	MDI Shared Memory Manager routine to test for the existence of the specified shared memory segment.
sizeof segment	MDI Shared Memory Manager routine used to obtain the size in bytes of the specified shared memory segment.

4.2.3.13.2 avi sock write

This function writes to a socket with retries. It checks the socket descriptor on entry, and if it is invalid, then it connects to the socket using the port and hostname. Next, it sends the message to the socket. If the send fails, then the socket is closed and reopened. The loop is repeated until the send succeeds or the retry limit is exceeded. The structure chart for *avi sock write* is shown in Figure 105. Descriptions of the routines called by *avi sock write* are provided in Table 132.

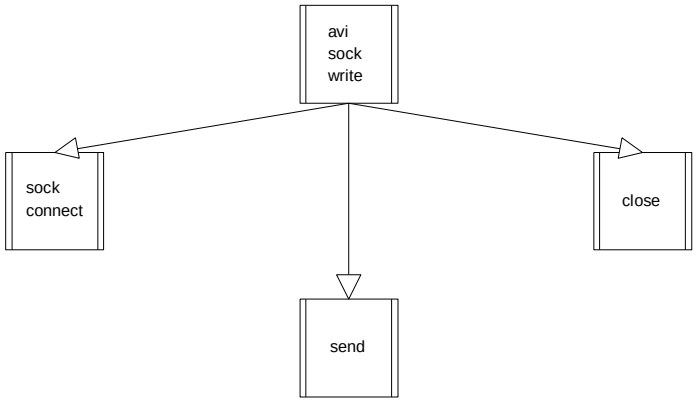


Figure 105. *avi sock write* structure chart

Table 132. Routines called by *avi sock write*

ITEM	DESCRIPTION
avi sock write	Write to a socket with retry.
close	C library function to close a file or device.
send	C library function to send data over a socket.
sock connect	MDI Socket routine used to create a socket connection to the specified host and port.

4.2.3.13.3 catch signal

This function catches signals that can kill this process. Any signal that makes it to this routine indicates an unhandled, fatal condition. This routine logs a message and performs an error exit for the process. *catch signal* calls only *process status message* so no structure chart is provided. Descriptions of the routines called by *catch signal* are provided in Table 133.

Table 133. Routines called by *catch signal*

ITEM	DESCRIPTION
catch signal	Catch signals which can kill this process.
process status message	MDI Process Status routine used to log a status message for the specified status type. If the process status library was configured to use a status logger then the message is forwarded to the status logger. Otherwise the message is written to the configured status log file.

4.2.3.13.4 signal setup

This function sets up the signal handler for all signals that are not currently handled within the calling process. Set up a signal set and register the action and handler for each one. Some signals can be ignored and these are set up accordingly. The structure chart for *signal setup* is shown in Figure 106. Descriptions of the routines called by *signal setup* are provided in Table 134.

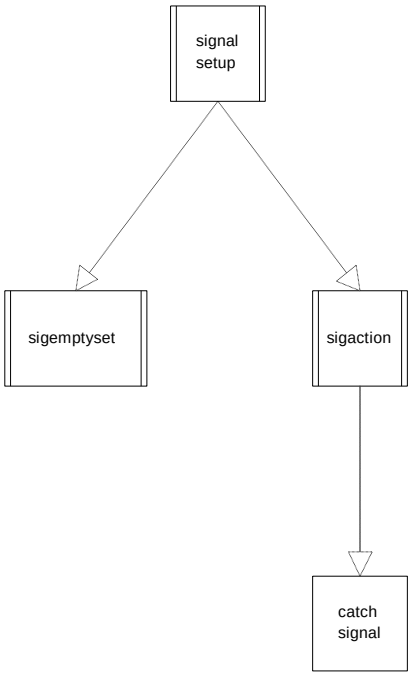


Figure 106. *signal setup* structure chart

Table 134. Routines called by *signal setup*

ITEM	DESCRIPTION
catch signal	Catch signals which can kill this process.
sigaction	C library function which allows the calling process to examine or specify the action to be taken on delivery of a specific signal.
sigemptyset	C library function that initializes the passed set to exclude all signals defined by the system.
signal setup	This function sets up the signal handler for all signals that are not currently handled within the calling process.

4.2.3.13.5 init dcm status

This function initializes the passed DCM status structure. The structure contains one entry for each AVI site. Neither a structure chart nor a description of routines called table is provided as *init dcm status* calls no subroutines.

4.2.3.14 CRC

This library contains functions that implement 16-bit CRC routines. The library includes the code to calculate the following CRCs:

- CCITT
- Reverse CCITT
- CRC-16

- Reverse CRC-16

Additional 16-bit CRCs may be created by using the lower level functions with other generator polynomials. For a discussion of the CRCs and their implementation see the *C Programmer's Guide to Serial Communications* by Joe Campbell, Chapters 3 and 23.

4.2.3.14.1 crc init

This function builds tables for the CCITT and CRC-16 CRCs and their reverses. The global tables are initialized and using macros that expand to calls to *crc make table* with the appropriate parameters for each CRC. The structure chart for *crc init* is shown in Figure 107. Descriptions of the routines called by *crc init* are provided in Table 135.

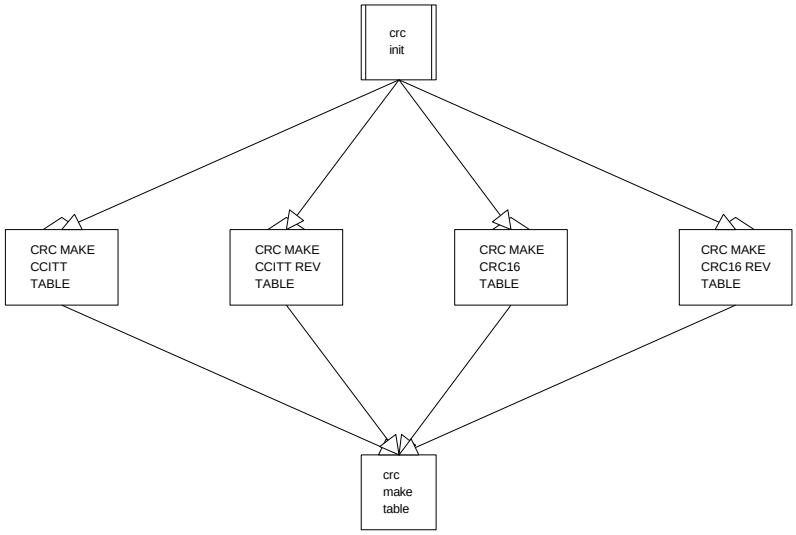


Figure 107. *crc init* structure chart

Table 135. Routines called by *crc init*

ITEM	DESCRIPTION
crc init	This function builds tables for the CCITT and CRC-16 CRCs and their reverses.
CRC MAKE CCITT REV TABLE	Macro to make the reverse CCITT CRC lookup table.
CRC MAKE CCITT TABLE	Macro to make the CCITT CRC lookup table.
CRC MAKE CRC16 REV TABLE	Macro to make the CRC-16 CRC lookup table.
CRC MAKE CRC16 TABLE	Macro to make the CRC-16 CRC lookup table.
crc make table	This function builds a table of CRC values for 8-bit bytes. The generator polynomial and CRC function is passed into the function; thus, it can be used for any 16-bit CRC, normal or reversed.

4.2.3.14.2 crc make table

This function builds the CRC lookup table using the passed generator function and polynomial. Two-hundred fifty-six values (all the values possible for a single byte) are stored into the passed table. Only the passed function may be called so neither a structure chart nor a table of called functions is provided.

4.2.3.14.3 crc calc

This function calculates the new value for a 16-bit CRC for byte data. The generator polynomial is passed into the function; thus, it can be used for any 16-bit CRC. Neither a structure chart nor a description of routines called table is provided as *crc calc* calls no subroutines.

4.2.3.14.4 crc calc rev

This function calculates the new value for a reverse 16-bit CRC for byte data. The generator polynomial is passed into the function; thus, it can be used for any 16-bit CRC. The generator variable must be reversed before it is passed to this function. Neither a structure chart nor a description of routines called table is provided as *crc calc rev* calls no subroutines.

4.2.3.14.5 crc msg ccitt

This function calculates the CCITT CRC on the passed message. It uses a macro that results in a call to *crc calc* with the appropriate parameters. *crc msg ccitt* calls only *crc calc* so no structure chart is provided. Descriptions of the routines called by *crc msg ccitt* are provided in Table 136.

Table 136. Routines called by *crc msg ccitt*

ITEM	DESCRIPTION
crc calc	This function calculates the new value for a 16-bit CRC for byte data.
CRC CCITT	Macro to do the lookup of one byte using the CCITT CRC.
crc msg ccitt	This function calculates the CCITT CRC on the passed message.

4.2.3.14.6 crc msg rev ccitt

This function calculates the reverse CCITT CRC on the passed message. It uses a macro that results in a call to *crc calc rev* with the appropriate parameters. *crc msg rev ccitt* calls only *crc calc rev* so no structure chart is provided. Descriptions of the routines called by *crc msg rev ccitt* are provided in Table 137.

Table 137. Routines called by *crc msg rev ccitt*

ITEM	DESCRIPTION
crc calc rev	This function calculates the new value for a 16-bit reverse CRC for byte data.
crc msg rev ccitt	This function calculates the reverse CCITT CRC on the passed message.
CRC REV CCITT	Macro to do the lookup of one byte using the CCITT reverse CRC.

4.2.3.14.7 crc msg crc16

This function calculates the CRC-16 CRC on the passed message. It uses a macro that results in a call to *crc calc* with the appropriate parameters. *crc msg crc16* calls only *crc calc* so no structure chart is provided. Descriptions of the routines called by *crc msg crc16* are provided in Table 138.

Table 138. Routines called by *crc msg crc16*

ITEM	DESCRIPTION
crc calc	This function calculates the new value for a 16-bit CRC for byte data.
CRC CRC16	Macro to do the lookup of one byte using the CRC-16 CRC.
crc msg crc16	This function calculates the CRC-16 on the passed message.

4.2.3.14.8 crc msg rev crc16

This function calculates the reverse CRC-16 CRC on the passed message. It uses a macro that results in a call to *crc calc rev* with the appropriate parameters. *crc msg rev crc16* calls only *crc calc rev* so no structure chart is provided. Descriptions of the routines called by *crc msg rev crc16* are provided in Table 139.

Table 139. Routines called by *crc msg rev crc16*

ITEM	DESCRIPTION
crc calc rev	This function calculates the new value for a 16-bit reverse CRC for byte data.
crc msg rev crc16	This function calculates the reverse CRC-16 CRC on the passed message.
CRC REV CRC16	Macro to do the lookup of one byte using the CRC-16 reverse CRC.

5. TRACEABILITY MATRIX

The traceability matrix for the AVI Data Processing System is presented in this section. It lists the requirements of the system that were presented in Section 3 of this document. Along with each requirement is the source of the requirement, the design element it was assigned to, the level at which it will be tested, and the method that will be used to verify the requirement.

This table will be used throughout the design, development, and test of the system to ensure that the requirements have been met. It will continually be updated as requirements and design elements are refined. During development of the Acceptance Test Plan (ATP), sections of the test plan will be referenced in the *TEST LEVEL* column of this table to cross-reference to the ATP.

The columns of the traceability table have the following meanings:

- Requirement Number: A unique identifier (with embedded level) assigned to the requirement
- Requirement: A brief description of the requirement
- Source: A paragraph reference in RFO, proposal or other source for this requirement
- Allocated To: The design element to which the requirement is allocated
- Test Level: The level at which testing of the requirement will take place
- Verify: The verification technique for the requirement

REQUIREMENT NUMBER	REQUIREMENT	SOURCE	ALLOCATED TO
	GENERAL REQUIREMENTS		
AVI-GN-1	The program shall have its developmental progress documented every four weeks.	P 2.2.2.4.1 P1 S1	Program
AVI-GN-2	The program shall have a final report prepared at the end of the project.	P 2.2.2.4.1 P1 S1	Program
AVI-GN-3	The program shall have its status reported in interim reports, as necessary.	P 2.2.2.4.1 P1 S1	Program
AVI-GN-4	The program shall have a Software Development Plan.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-5	The program shall have a Software Requirements Specification.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-6	The program shall have a Software Design Document.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-7	The program shall have a Software Acceptance Test Plan.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-8	The program shall have a Software User Manual.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-9	The program shall have a Software Version Description Document.	P 2.2.2.4.2 P1 S1	Program
AVI-GN-10	The program shall have training provided on AVI Reader Field Sites.	P 2.2.2.4.5 P1 S1	Program
AVI-GN-11	The program shall have training provided on the AVI Data Processing System.	P 2.2.2.4.5 P1 S1	Program
AVI-GN-12	The program shall have a Tag specification developed.	R 24 S4	Program
	SYSTEM REQUIREMENTS		
	INTERFACE REQUIREMENTS		
AVI-IF-1	The system shall interface to the MDI Data Server.	Design	AVI Master Con System
AVI-IF-2	The system shall interface to the Reader Field Site.	Design	AVI Master Con System

REQUIREMENT NUMBER	REQUIREMENT	SOURCE	ALLOCATED TO
AVI-IF-3	The system shall interface with the user.	Design	AVI Master Com System
	FUNCTIONAL REQUIREMENTS		
AVI-FN-1	The AVI Data Processing System shall gather AVI Tag read data from Reader Field Sites.	R 24 S8 P 2.2 P6 S2	AVI Data Proce Software
AVI-FN-2	The AVI Data Processing System shall process AVI Tag read data.	R 24 S8 P 2.2 P6 S2	AVI Data Proce Software
AVI-FN-3	The AVI Data Processing System shall process status data.	P 2.2.1 P1 S3	AVI Data Proce Software
AVI-FN-4	The AVI Data Processing System shall allow for configuration of system components.	P 2.2.1 P1 S6	AVI Data Proce Software
AVI-FN-5	The AVI Data Processing System shall handle errors.	P 2.2.1 P1	AVI Data Proce Software
	PHYSICAL CHARACTERISTIC REQUIREMENTS		
AVI-PY-1	The AVI Data Processing System shall have a dedicated master computer.	R 25.4.6 S1	AVI Master Com System
AVI-PY-2	The AVI Data Processing System shall have a modem pool.	Design	AVI Modem Server Sy
	MISCELLANEOUS REQUIREMENTS		
AVI-MS-1	The AVI Data Processing System shall have software designed using either structured or object-oriented methodologies.	P 2.2.2 P1 S2	AVI Data Proce Software
AVI-MS-2	The AVI Data Processing System shall have software written in C or C++.	P 2.2.2 P1 S3	AVI Data Proce Software
AVI-MS-3	The AVI Data Processing System shall have software tested at the unit and system integration levels, as appropriate.	P 2.2.2 P1 S3	AVI Data Proce Software
	SUB-SYSTEM REQUIREMENTS		
	MASTER COMPUTER SUBSYSTEM REQUIREMENTS		

REQUIREMENT NUMBER	REQUIREMENT	SOURCE	ALLOCATED TO
	INTERFACE REQUIREMENTS		
AVI-IF-1.1	The AVI Master Computer System shall interface with the MDI Data Server.	AVI-IF-1	AVI Master Com System
	PHYSICAL CHARACTERISTIC REQUIREMENTS		
AVI-PY-1.1	The AVI MC shall be a Sun Microsystems Ultra SPACRStation or better	R 25.4.6 S1 P 2.2.2.3.4 P1 S2 AVI-PY-1	AVI Master Com System
AVI-PY-1.2	The AVI MC shall have, at a minimum, the following items: • 167mhz SPARC CPU • 4.2 GB Hard Disk • 128 MB RAM • Floppy Disk Drive • Sun CD-ROM drive • Turbo GX+ Graphics card • 20” Sun color monitor • 2 Ethernet cards • 2 SCSI channels • 2 RS-232 ports • Keyboard • 2-Button mouse • 96 serial ports (SCSI attached) • 1 modem/site • 1 disl-up line/site	R 25.4.7 S1 P 2.2.2.3.4 P1 S2 AVI-PY-1	AVI Master Com System
	MODEM POOL SUBSYSTEM REQUIREMENTS		
	INTERFACE REQUIREMENTS		
AVI-IF-2.1	The AVI Modem Server System shall interface with the AVI Reader Field Sites.	AVI-IF-2	AVI Modem Server Sy

REQUIREMENT NUMBER	REQUIREMENT	SOURCE	ALLOCATED TO
	MASTER COMPUTER SOFTWARE SUBSYSTEM REQUIREMENTS		
	INTERFACE REQUIREMENTS		
AVI-IF-1.2	The AVI Data Processing Software shall interface with the MDI Data Server using protocol TBD.	AVI-IF-1	AVI Data Proce Software
AVI-IF-2.2	The AVI Data Processing Software shall interface with the AVI Reader Field Sites using protocol TBD.	AVI-IF-2	AVI Data Proce Software
AVI-IF-3.1	The AVI Data Processing Software shall interface with the user.	AVI-IF-3	AVI Data Proce Software
	FUNCTIONAL REQUIREMENTS		
AVI-FN-1.1	The AVI Data Processing Software shall gather AVI Tag read data from the AVI Reader Field Site.	P 2.2 P6 S1 P 2.2.2.3 AVI-FN-1	AVI Data Proce Software
AVI-FN-2.1	The AVI Data Processing Software shall store the AVI Tag read data collected from the Reader Field Sites .	R 24 S8 R 25.4.2 S1 R 25.4.2 S2 AVI-FN-2	AVI Data Proce Software
AVI-FN-2.2	The AVI Data Processing Software shall provide the AVI Tag read data to the MDI Data Server.	R 25.4.2 S1 R 25.4.2 S2 P 2.2 P6 S1 AVI-FN-2	AVI Data Proce Software
AVI-FN-2.3	The AVI Data Processing Software shall process operational data gathered from the Reader Field Sites.	R 25.4.2 S1 P 2.2 P6 S1 P 2.2.2.3.1 AVI-FN-2	AVI Data Proce Software
AVI-FN-3.1	The AVI Data Processing Software shall determine AVI Reader Field Site status.	P 2.2.1 P1 S7 AVI-FN-3	AVI Data Proce Software
AVI-FN-3.2	The AVI Data Processing Software shall monitor its own status.	P 2.2.1 P1 S3 AVI-FN-3	AVI Data Proce Software
AVI-FN-4.1	The AVI Data Processing Software shall accept configuration data.	P 2.2.2.1 P1 S6 AVI-FN-4	AVI Data Proce Software

REQUIREMENT NUMBER	REQUIREMENT	SOURCE	ALLOCATED TO
AVI-FN-4.2	The AVI Data Processing Software shall store configuration data.	P 2.2.2.1 P1 S6 AVI-FN-4	AVI Data Proce Software
AVI-FN-4.3	The AVI Data Processing Software shall perform configuration operations.	P 2.2.2.1 P1 S6 AVI-FN-4	AVI Data Proce Software
AVI-FN-5.1	The AVI Data Processing Software shall detect communications errors between itself and the AVI Reader Field Site.	P 2.2.1 P1 S3 AVI-FN-5	AVI Data Proce Software
AVI-FN-5.2	The AVI Data Processing Software shall log communications errors between itself and the AVI Reader Field Site.	P 2.2.1 P1 S3 AVI-FN-5	AVI Data Proce Software
AVI-FN-5.3	The AVI Data Processing Software shall report communications errors between itself and the AVI Reader Field Site.	P 2.2.1 P1 S3-4 AVI-FN-5	AVI Data Proce Software