Railroad Delay Advance Warning System

(Advance Warning to Avoid Railroad Delays)
(AWARD)

Model Deployment Initiative

System Design Document

Version 1.0

March 25, 1998

SwRI Project No. 10-8684

P.O. No. 7-70030
Req. No. 50115-7-700-30

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

AWARD Project Manager	Date
SwRI MDI Project Manager	Date
Automation Engineering Director	Date

Acronym List

ATMS	Advanced Traffic Management System
AWARD	Advance Warning to Avoid Railroad Delays
ETA	Estimated Time of Arrival
ETTA	Estimated Time Until Arrival
ITS	Intelligent Transportation Systems
MDI	Model Deployment Initiative
RRS	Railroad Road Software Subsystem
SwRI	Southwest Research Institute
TBD	To be Determined
TOS	TransGuide Operational Software
TxDOT	Texas Department of Transportation
VMS	Variable Message Sign

Table of Contents

List of Figures

List of Tables

AWARD

(Advance Warning to Avoid Railroad Delays)

System Design Document

1. Introduction

The Advance Warning to Avoid Railroad Delays (AWARD) system is an Advanced Traveler Information Service (ATIS) implementation designed to help motorists and emergency response vehicles avoid delays due to railroad operations that cross freeway access frontage roads. Railroad operations in urban areas are usually carried out at low speeds which can result in grade crossings being closed to vehicular traffic for several minutes. In high traffic areas and during peak traffic times, closing a frontage road for several minutes can prevent traffic from entering a freeway and can also block exiting traffic. Eventually this can result in traffic congestion on feeder roads and in the exiting lanes of the freeway. The AWARD system includes sensors to detect the presence and characteristics of trains operating in affected areas and computer algorithms to predict the time and duration of blockage of grade crossings at or near freeway exits and entrances. The AWARD system is integrated with TransGuide operations to provide advance knowledge of train operations and allow motorists and emergency vehicles to select different freeway exits or entrances or choose alternate routes to avoid congestion.

The AWARD system includes sensors at selected locations along the Union Pacific Kerrville line track near IH 10. Trains on this section of track operate at speeds of approximately 10 mph and can block freeway access at several frontage road locations for periods of over five minutes. The sensors measure the speed of trains approaching grade crossings and transmit speed information to a central computer at the TransGuide facility. Computer algorithms predict the time and duration that selected grade crossings will be blocked and provide this information to TransGuide operators, motorists, and emergency operations through other Model Deployment Initiative (MDI) components including the Automatic Route Guidance System, the Traveler Advisory Information System, and the Area Wide Database. The system hardware and software are designed to allow expansion for additional sensor and grade crossing locations in the future.

1.1 Purpose of System

The purpose of the AWARD system is to predict when specified grade crossings will be closed by train operations and provide this information to TransGuide operators, the motoring public and emergency vehicle operators in time for them to select alternate routes and avoid traffic congestion at the closed grade crossing. This has the capability to reduce congestion, reduce hazards that can cause accidents and reduce delays in travel time.

The interactions between different modes of transportation often present particular difficulties in traffic management and are the cause of traffic hazards, motorist delays, and obstructions to emergency vehicle operations. In the case of freeway-railroad interaction addressed here, the Kerrville line of the

Union Pacific Railroad runs nearly parallel to IH 10 from south of Culebra Road to near Basse Road where IH 10 turns westerly. The railroad line continues northward, crossing under IH 410 at Jackson Keller Road. In this interval of slightly over five miles, the railroad track crosses three major roads where crossing blockages can affect freeway traffic.

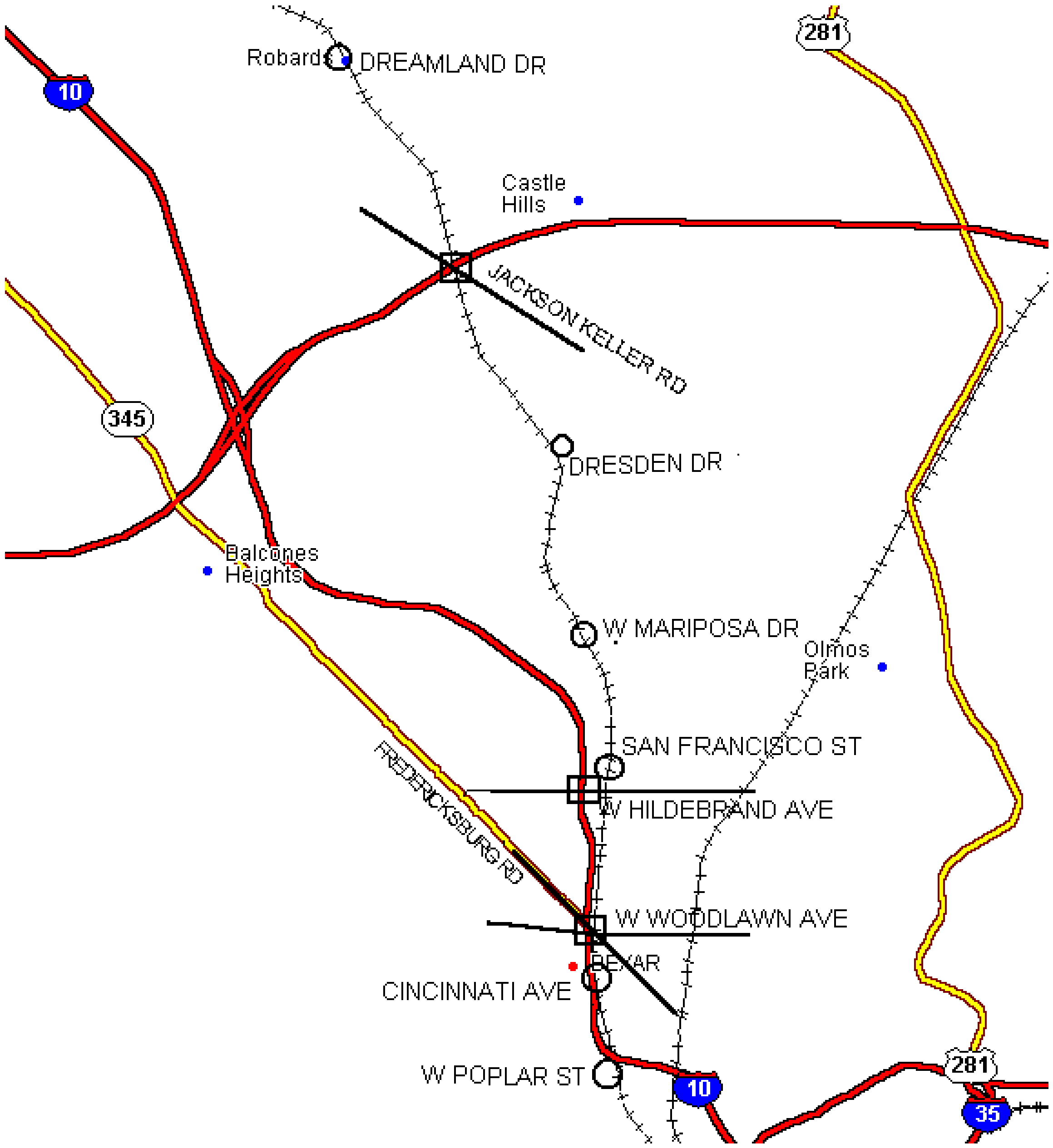


Figure 1. Award Sensor and Crossing Sites

Figure 1 shows the locations of sensors and major railroad grade crossings from Dreamland Road, north of IH 410 to Poplar Street on the west side of IH 10. This section of track crosses three major intersections that directly affect traffic on IH 10 and IH 410:

- Jackson-Keller at IH 410. Traffic on Jackson-Keller as well as both directions on the IH 410 access road is blocked by trains at this crossing. Traffic exiting IH 410 East can be blocked by a train causing congestion to back up onto the freeway lanes.
- Fredericksburg Road and Woodlawn Avenue at IH 10. At this point, traffic exiting from IH 10 and attempting to turn east on either Woodlawn or Fredericksburg is blocked by passing trains. If the grade crossing is blocked for several minutes, traffic can back up until it interferes with the exit lanes and eventually causes congestion and hazards on the freeway.
- Hildebrand and IH 10. Traffic exiting from IH 10 and turning east on Hildebrand is blocked by trains in grade crossings.

This track is used primarily to haul gravel and rock from a quarry north of town. Due to the congestion of the area and the condition of the track, trains operating on this section of track are limited to speeds of ten mph (discussions with Mr. Doug Woods of Union Pacific). Since the trains are moving so slowly it takes a long time for them to clear a grade crossing and delays of up to ten minutes have been reported by residents and motorists in the area. The trains do not maintain a fixed schedule but five or more trains may move across the track on some days (according to businesses located next to the track). As a result of all these factors, train operations in this section have an appreciable effect on freeway traffic and on the travel time of motorists, including emergency vehicles.

1.2 Operational Concept

The AWARD system predicts grade crossing blockage by detecting approaching trains a distance from the crossing. Sensors are placed at selected distances from the three chosen grade crossings. Trains are detected by acoustic sensors consisting of directional microphones sensitive to the sound of railroad cars moving on the track. The presence of a train energizes a doppler radar unit aimed at the tracks. The doppler radar measures the speed of the passing train and transmits speed data to a central computer located at the TransGuide center as shown in the Architectural Block Diagram. A workstation in the TransGuide center calculates an equation of motion for the train and predicts the time of arrival and the duration of closure for grade crossings ahead of the train.

The predicted time and duration of crossing closures is provided to TransGuide operators in the form of a “railroad grade crossing event” patterned after traffic events currently used in TransGuide operations. TransGuide operators may respond to the railroad grade crossing event by initiating variable message sign displays or other appropriate actions. Information on grade crossing closures is also placed in the Area Wide Database for use by other Model Deployment activities.

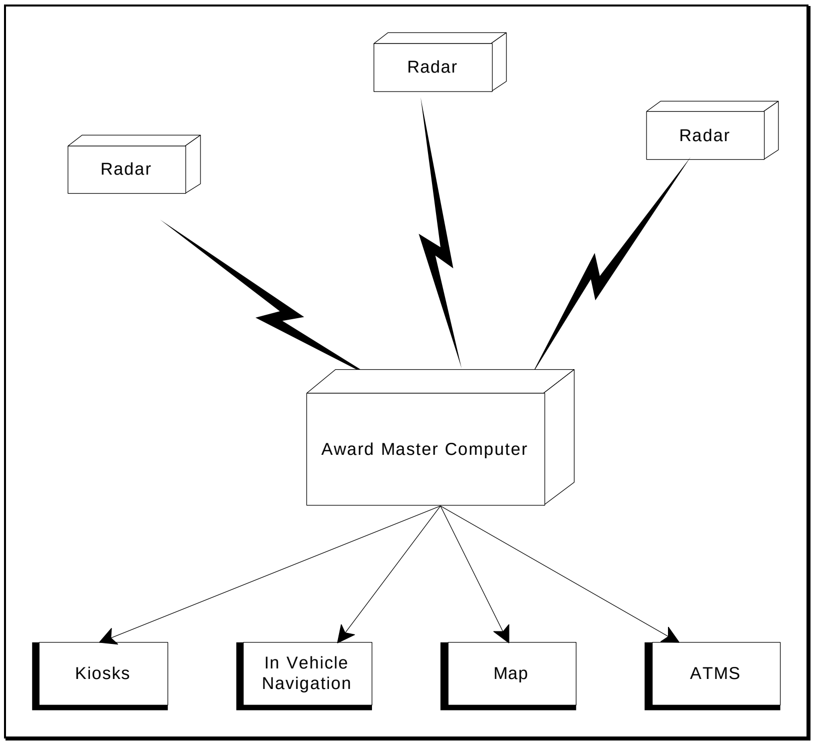


Figure 2. Architectural Block Diagram

1.3 Goals and Objectives

The immediate goal of the AWARD system is to provide information on predicted grade crossing closures early enough to allow motorists and emergency vehicle operators to select alternate routes to avoid the congested areas. This results in reducing travel time for motorists, reducing congestion on freeway exit lanes at the affected crossings and reducing delays in emergency vehicle response.

1.4 Referenced Documents

The following documents are referenced in this design:

- Texas Department of Transportation Request for Offer (RFO) for the Model Deployment Initiative System Integration, 60115-7-70030; TxDOT Specification No. 795-SAT-01.
- San Antonio Advanced Traffic Management Software Requirements Document (December, 1995).
- Object Oriented Analysis and Design by Martin and Odell, Prentice Hall, 1992.
- Southwest Research Institute Proposal No. 10-20352, A Proposal for 60115-7-70030: Request for Offer, Model Deployment Initiative System Integration.
- Installation and Operations Manual - SmartSonic™ TSS-1, International Road Dynamics, Saskatoon, Sask., 1997
- MDI Common Code Software Design Document

2. External Interfaces

AWARD system interfaces include electrical power and signal connections between hardware subsystems (electrical interfaces), data interfaces between computer programs (software interfaces), and between programs and operators (user interfaces). Each of these is described briefly below.

2.1 Remote Sensor Site

Interfaces at the remote sensor site include:

- Electrical power connection to the field unit is through liquid tight strain relief fittings. A surge suppresser and circuit breaker is provided in the input power circuit.
- A power supply at each sensor site provides 12 VDC to the SmartSonic acoustic detector and to the O'Conner radar unit.
- A signal from the SmartSonic controller card is wired directly to a time delay relay located on the SmartSonic enclosure. The output of the time delay relay is wired directly to the transmit power circuit of the radar unit.
- The Radar to Modem connection is EIA-232 also known as RS-232 at 2400 baud, 8 bit, no parity, 1 stop bit. A DB 24 connector on a cable from the radar connects to a DB 24 connector on the modem. EIA-232 establishes the electrical connections.
- The Modem to Telephone line connection includes a surge suppresser. RJ11 connectors are used for the telephone line connections.
- Field modem to TransGuide modem: Each modem provides an RJ11 Socket for connecting to the commercial telephone system through a surge suppresser. The electrical connection using RJ-11 connectors is standard U.S. telephone practice. The communication protocol used is standard, uncompressed, serial, modulated signal technique.

2.2 TransGuide Location

Interfaces at the TransGuide location include:

- Telephone Lines to Modems at TransGuide: the telephone hook-up connects to computer modems using RJ-11 connectors and is standard U.S. telephone connections. The communication protocol is standard, uncompressed, serial, modulated signal technique.
- Modems at TransGuide to Workstation: the modem pool provides a SCSI-2 cable connection to the workstation. The SCSI-2 standard defines the pertinent electrical and communication protocols.

2.3 Software Interfaces

External interfaces associated with the AWARD software are:

- AWARD software to TransGuide ATMS: The AWARD system provides the train delay information to the TransGuide ATMS. This information is sent to the existing TransGuide ATMS as an external event using a socket communication protocol. This event is similar to the alarm incidents currently handled by the TransGuide ATMS. Crossing location and train delay duration are transmitted as part of this event. The crossing location and train delay duration are used by the TransGuide ATMS to perform automatic scenario searches and as an aid to the TransGuide operators in determining the execution time for the selected scenario.

- AWARD Software to MDI Data Server: The AWARD system provides raw sensor information and train delay information to the MDI Data Server.

3. Requirements

This section presents the attributes that are required for a successful implementation of the AWARD Railroad Delay Advance Warning System. Most of these requirements are derived directly from TxDOT Specification 795-SAT-01, "Model Deployment Initiative System Integration Request for Offer" and the Southwest Research Institute Proposal 10-20352. Listed in Section 3.1, General Requirements, are the programmatic requirements that specify what items are to be delivered and the schedule for specific events. Section 3.2, System Level Requirements, lists requirements that apply to the overall AWARD system and Sections 3.3 through 3.8 list requirements that apply to each of the subsystems individually.

The requirement definition number provides a unique code for reference and tracking. The first two characters identify the Railroad Delay Project of MDI. The next 3 characters designate the group or subsystem associated with the requirement. The numeric entry is a sequence number within the group.

3.1 General Requirements

The following section lists requirements for delivery of specific items during the project.

RR-GEN-01	An 80% System design document shall be delivered on February 14, 1997. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-02	A 100% design document shall be delivered on December 31, 1997. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-03	A Software Acceptance Test Plan shall be delivered. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-04	A Version Description Document shall be delivered. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-05	Monthly status reports shall be provided via a presentation with the customer. (SwRI Proposal 10-20352 stated a report will be delivered. An alternative was negotiated.)
RR-GEN-06	A training program shall be presented. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-07	A videotape of the training program shall be delivered. (SwRI Proposal 10-20352, section 2.1.2.8.3)
RR-GEN-08	A final report shall be delivered. (SwRI Proposal 10-20352, section 2.1.2.8.3)

3.2 System Level Requirements

The following requirements specify the overall operation and performance of the system. Each requirement is identified by a unique code and is referenced to the source from which the requirement was derived.

RR-SYS-01	The system shall deliver advance warning to motorists of expected delays at railroad crossings. (TxDOT 795-SAT-01, Paragraph 28)
RR-SYS-02	The system shall determine the speed and length of a train engine and attached railroad cars. (TxDOT 795-SAT-01, Paragraph 28)
RR-SYS-03	The system shall determine expected delay times at selected grade crossings. (TxDOT 795-SAT-01, Paragraph 28)
RR-SYS-04	The system shall transmit an expected delay to TransGuide Operators as an alarm through a software interface with the existing TransGuide ITS system. (TxDOT 795-SAT-01, Paragraph 28)
RR-SYS-05	Expected railroad delays shall be transmitted to the traveling public by use of existing variable message signs and also to the San Antonio Area Wide Database. (TxDOT 795-SAT-01, Paragraph 28)
RR-SYS-06	The system shall provide warnings for grade crossings at IH 10 and Fredericksburg Road, IH 10 and Hildebrand Road, and IH 10 and Vance Jackson Road. (TxDOT 795-SAT-01, Paragraph 29.1.3)
RR-SYS-07	The field equipment shall be mounted on a suitable structure at some location along the railroad line in advance of the crossing for which warnings are to be given. (TxDOT 795-SAT-01, Section 29.2.1 refers to an ”existing structure” along the track. Investigation has determined that there are no existing structures at some locations where sensors will be needed.)
RR-SYS-08	Field equipment shall be located in TxDOT or the City of San Antonio right of way. (TxDOT 795-SAT-01, Section 29.2.1)
RR-SYS-09	The field equipment shall determine length and speed of trains through observation only. No connection to the railroad tracks or controlling equipment is allowed. ((TxDOT 795-SAT-01, Paragraph 29.2.2)

3.3 Sensor Subsystem Requirements

RR-SNS-01	The train speed sensor shall have a range to allow measurement of the train speed from a location outside the railroad right-of-way. This distance is normally 50 feet on either side of the track center line but may vary in some locations. (RR-SYS-04)
RR-SNS-02	The detector unit shall measure locomotive speed within 2 miles per hour (+/-) at the maximum train speeds allowed for the section of track where sensors are installed. (TxDOT 795-SAT-01, Paragraph 29.2.4 specifies ±2 mph for trains traveling at 60 mph. Trains on the selected section of track operate at less than 10 mph. Specifying a sensor that operates at 60 mph will reduce the measurement accuracy at low speeds.)

3.4 Communications Subsystem Requirements

RR-COM-01	The field unit shall communicate to the TransGuide equipment using a non-proprietary protocol. (TxDOT 795-SAT-01, Paragraph 29.1.2)

3.5 Electrical Subsystems Requirements

RR-ELC-01	The field unit shall operate on standard line power. (nominal 120 VAC) (RR-545-07)

3.6 Mechanical Subsystem Requirements

RR-MEC-01	The equipment will be designed to operate within an ambient temperature range of -12°C to 49°C (10°F to 120°F) and will not allow condensation accumulations which would interfere with its operation. (RR-SYS-07)
RR-MEC-02	The system enclosure will be able to be mounted to a pole or other suitable structure. (RR-SYS-07 and RR-SYS-08)
RR-MEC-03	The system will provide an internal mechanism for accurate pointing of the sensor. (RR-SNS-01)

3.7 Railroad Software Subsystem Requirements

RR-RRS-01	The RR-Delay Master Computer shall calculate the length of the train from measured train speed integrated over time. (SwRI Proposal 10-20352, Section 2.4.1) The RFO stated (Paragraph 29.2.2) "The detector unit shall measure the length of the locomotive and all attached cars within 10 feet (+/-)." The detector itself does not measure train length directly. The specified accuracy is possible for trains traveling at speeds of 50 mph or more but can only be done for slow trains (10 mph) if acceleration or deceleration is constant.

RR-RRS-02	The RR-Delay Master Computer shall calculate the expected time of arrival of the first element of the train and the last element of the train at selected downrail crossings. (SwRI Proposal 10-20352, Section 2.4.1)
RR-RRS-03	The RR-Delay Master Computer shall determine expected delay times at railroad crossings. The RR-Delay Master Computer shall estimate delay time within ±30 seconds. (TxDOT 795-SAT-01, Paragraph 29.3.3)
RR-RRS-04	The RR-Delay Master Computer shall transmit the railroad delay data to the existing TransGuide ITS system. (TxDOT 795-SAT-01, Paragraph 29.3.3)

3.8 TransGuide Operational Software Subsystem Requirements

Software modifications to the TransGuide ATMS provide the system software interface and operational control required to incorporate train delay information generated by the AWARD system into the existing traffic management system software. The requirements related to these software changes are in the table below.

RR-TGS-01	The TransGuide Operational Software shall interface with and receive railroad delay data from the Railroad Operational Software. (RR-RRS-04)
RR-TGS-02	The TransGuide Operational Software shall transmit expected delay information to TransGuide operators as an alarm. (TxDOT 795-SAT-01, Paragraph 28)
RR-TGS-03	The TransGuide Operational Software shall be capable of performing a scenario search for a RR delay incident. (SwRI Proposal 10-20352, Section 2.4.1)

3.8.1 TransGuide Alarm/Incident Handler Requirements

The Alarm/Incident Handler (AIH) subsystem of the TransGuide ATMS is responsible for the handling of traffic incidents or alarms. This subsystem was be modified as part of the AWARD project in order to handle the RR delay information being sent by the RSS. The requirements related to the modifications made to the AIH are listed below.

RR-TGS-02.01	The AIH shall accept a RR delay alarm from the RSS. (RR-TGS-02)
RR-TGS-02.02	The AIH shall indicate the RR delay alarm as an update alarm if the RR delay alarm is related to a current RR delay incident. (RR-TGS-02)
RR-TGS-02.03	The AIH shall create a new AIH RR incident if the RR delay alarm is not related to a current RR delay incident. (RR-TGS-02)
RR-TGS-02.04	The AIH RR incident shall contain data from the railroad delay information contained in the RR delay alarm. (RR-TGS-02)

RR-TGS-02.05 The AIH shall build the AIH RR incident screen for new RR delay alarms. (RR-TGS-92)

RR-TGS-02.06 The AIH shall display the AIH RR incident screen, as an icon, on the workstation of the manager responsible for the sector containing the RR incident. (RR-TGS-02)

RR-TGS-02.07 The AIH shall generate an audio notification of new RR incident alarms at the workstation of the manager responsible for the sector containing the RR incident. (RR-TGS-02)

RR-TGS-02.08 The AIH shall update the railroad delay information for an existing incident using the railroad delay information contained in the associated RR delay update alarm. (RR-TGS-02)

RR-TGS-02.09 The AIH RR incident screen shall provide the same actions currently provided by the AIH-NewIncidentScreen. (RR-TGS-02)

3.8.2 TransGuide Scenario Manager Requirements

The Scenario Manager (SCM) subsystem of the TransGuide ATMS is responsible for the searching and execution of pre-defined and operator generated incident scenarios. This subsystem will need to be modified as part of the AWARD project in order to search and execute scenarios related to incidents occurring as a result of a RR delay. The requirement related to the modifications made to the SCM is listed below.

RR-TGS-03.01 The SCM-ScenarioSearchScreen shall contain the RR incident type for selection by a TransGuide operator. (RR-TGS-03)

4. Sensor System Design

The AWARD system described in this design document is the initial implementation and test of a new approach to handling intermodal traffic problems. It includes a limited number of sensors to predict train activity at three specific grade crossings where blocked intersections affect freeway traffic. The results of this limited implementation are being used to assess the effectiveness and benefits of the concept. The effectiveness of the system will serve as the basis for future expansion of the system and for implementation of advance warning methods at additional intersections. The design of the AWARD system is based on the following goals:

- Provide advance information on train crossings to allow motorists to plan and take alternate routes which avoid blocked intersections. This will reduce congestion at intersections and on freeways and reduce traffic hazards.

- Provide advance information on train crossings to TransGuide operators to allow them to respond to predicted crossing blockages. This will allow operators to include train information in planning VMS messages and in responding to traffic incidents.

- Provide advance information to emergency services to allow route planning that avoids congested intersections. This will lead to faster response time of emergency vehicles.

- Provide a system architecture that can be expanded to include additional sensor locations and additional grade crossings in the future. This will allow expansion of the system and implementation at other locations.

4.1 System Architecture

The AWARD system from an operational standpoint is depicted in Figure 3. Equipment in the field is primarily composed of an acoustic detector and a radar speed gun connected to a modem. This unit relays train velocity information using standard telephone communications through a second modem to the Award Master Computer located at the TransGuide facility. Software running on the Award Master Computer monitors the remote radar units to determine train locations and speeds. After calculating where street blockages will be occurring, the data is relayed to the TransGuide ATMS and the Area Wide Database. From the Area Wide Database, the data is available for use by other elements of the Model Deployment Initiative.

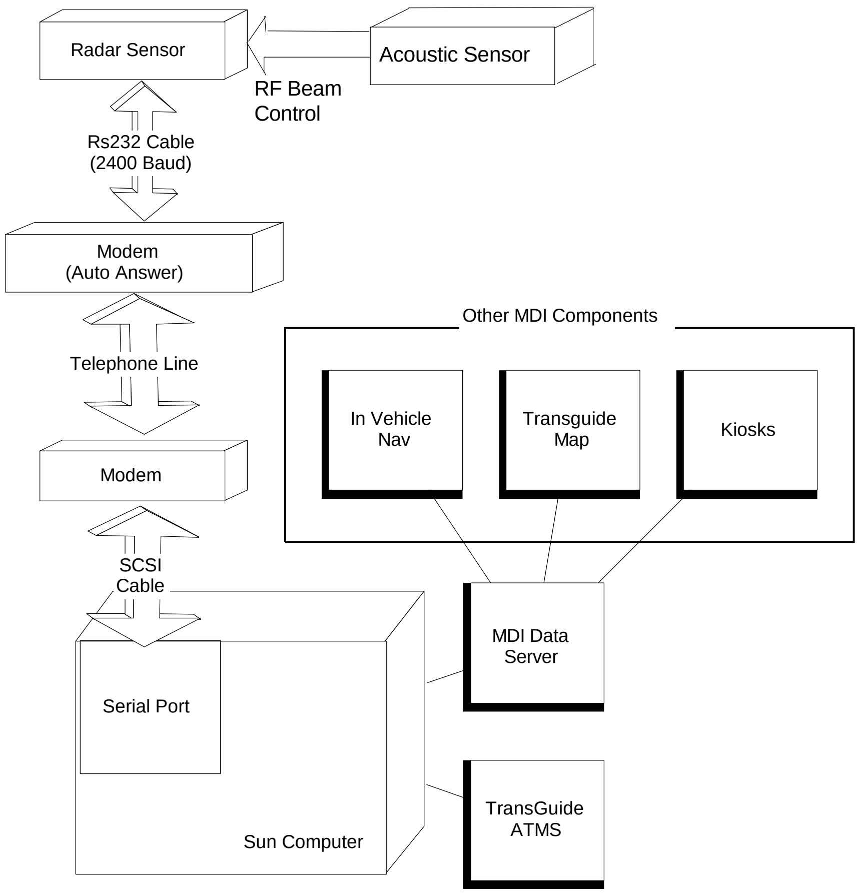


Figure 3. System Block Diagram

The flow of information related to grade crossing closures is illustrated in Figure 4. Train speed measurements are transmitted to the Railroad Master Computer Subsystem where estimated time and duration of crossing blockages are calculated. If any blockages are predicted to occur within specified time intervals a railroad incident event is generated. This information is provided to the Scenario Management Subsystem which communicates with TransGuide operations personnel through graphical user interfaces.

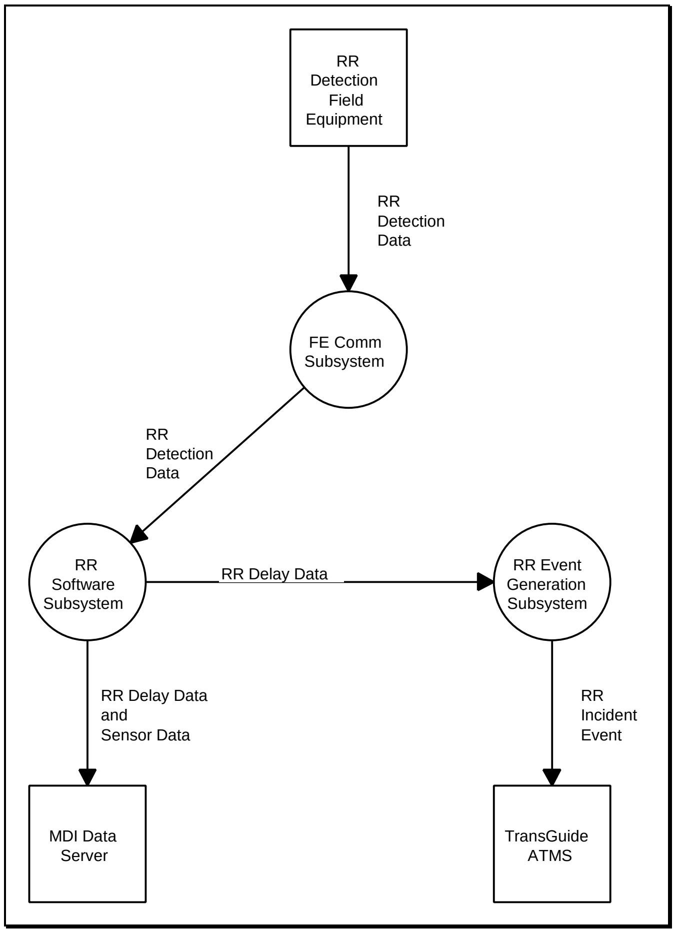


Figure 4. AWARD Process Flow

4.2 System Geographic Layout

One important consideration in the design of the AWARD system was the location of sites for train speed sensors. Sensors are located far enough from the grade crossings to provide enough advance warning to allow motorists to decide on an alternate route, possibly change freeway lanes, and take an earlier or later exit from the freeway. On the other hand, the sensors must be close enough to the crossings so that train speed is relatively constant and accurate predictions of crossing closure can be made. For this section of freeway, an advance warning time of 4 to 8 minutes was selected to provide motorists with several miles of driving in which to make alternate route plans and execute them.

Discussions with Union Pacific determined that the particular section of track (the Kerrville line) has a posted speed of 10 mph, the lowest in the city. The distance from sensors to grade crossings was then calculated based on the nominal train speed.

*distance = time*speed*
distance = 6min(1hr / 60min)*10mi / hr*
Distance = 1mile

Based on this distance, six sensors (one on each side of each crossing) are used to provide accurate time and duration estimates.

Regions of the track approximately one mile on each side of the three grade crossings were investigated to determine the availability of sites for mounting train sensors. In addition to the correct distance from grade crossings, acceptable sites must provide opportunities for mounting the sensor, an un-obstructed view of the railroad track and the availability of power and communications.

Sites were identified at each required location which provided the required characteristics.

- SITE 1 - One mile south of the Fredericksburg-Woodlawn crossing the track runs through a mixed residential/warehouse area. The sensor was mounted on a new 25-ft utility pole at Poplar Street between a warehouse and a paved area adjacent to a loading dock. The acoustic detector and the radar are aimed toward the track in the northward direction. There is very little background noise at this location.

- SITE 2 - One mile south of the Hildebrand Street crossing the track runs beside IH 10 which is double-decked at this location. The sensor at this location is mounted on an existing utility pole on the north side of Cincinnati Street. The acoustic detector and the radar are aimed toward the track in the northward direction. The acoustic sensor also picks up traffic noise from IH 10 which can be significant during heavy traffic.

- SITE 3 - One mile north of the Fredericksburg-Woodlawn crossing the track runs between the edge of Martinez Creek and Capitol Street in an area of light industry. At this location the sensor is mounted on a newly installed 30-foot utility pole located on the south side of San Francisco Street. This pole is taller than the others since the sensor is directed northward looking across San Francisco Street and the sensor must be high enough to clear any obstructing traffic on the street.

- SITE 4 - One mile north of the Hildebrand Street crossing the track runs along Martinez creek between Mardell Boulevard and Wildwood Drive. The sensor is mounted on a 25-foot pole on the north side of Mariposa Street and is directed northward.

- SITE 5 - One mile south of the Jackson-Keller grade crossing the track runs parallel to Arroya Vista Drive in a residential area. This sensor is located on a 25-foot utility pole on the north side of Dresden Drive.

- SITE 6 - North of the Jackson-Keller grade crossing the track runs along Olmos Creek through a wooded floodplain area without much development. The sensor is mounted on a 25-foot utility pole on the south side of Dreamland Drive.

Sensor Code	Site Location Telephone #	Location Coordinates	Orientation (angle to tracks)	Distance from Sensor Location to	
				Jackson-Keller	Hildebrand
A6	Dreamland (10689 TCM) 341-9615	Lat: -98 32 22 Lon: 29 32 11	44° facing south	8400 ft. South	
A5	Dresden (379 #TCM) 340-4689	Lat: -98 31 05 Lon: 29 29 55	39° facing north	7100 ft North	
A4	Mariposa 733-6041	Lat: -98 30 56 Lon: 29 28 50	20/ facing north		5600 ft South
A3	San Francisco 735-0124	Lat: -98 30 47 Lon: 29 28 06	25/ facing north		900 ft South
A2	Cincinnati (431 #TCM) 738-3894	Lat: -98 30 52 Lon: 29 26 54	29/ facing north		6600 ft North
A1	Poplar 733-6021	Lat: -98 30 45 Lon: 29 26 21	22/ facing north		10000 ft North

Table 1. Sensor Locations and Orientations

4.3 Sensor Subsystem Design

Each remote site includes an acoustic sensor to detect the presence of a train and a speed sensor to measure the speed of trains approaching a crossing.

Doppler radar was chosen as the most appropriate among the speed measurement technologies although it has some certain limitations. No commercial off-the-shelf (COTS) radar unit was found which was designed for viewing trains and communicating with a remote computer. Therefore, a vendor was found who was willing to quote such a system (the quote was part of the proposal for AWARD). Normal frequency bands of the radar units are X, K, and Ka. The least affected by rainfall is the X band, however, the vendor had no X band radar's which could measure vehicles approaching and receding, therefore K band was chosen. Radar gun technology is based on measuring frequency shift of a returned signal (i.e. Doppler shift) and is not very sensitive to slow speeds because these result in small frequency shifts. Radar units, however, may be adjusted so that they are more sensitive to slower or faster speeds than the nominal 60 miles per hour. One final limitation of using radar gun technology is that the FCC does not want radar guns to remain transmitting when unattended; therefore the radar gun was used with an acoustic sensor to activate the RF transmit beam only when a train is present.

4.3.1 Specifications for Selected Sensors

The SmartSonic Traffic Surveillance System (TSS-1) is a non-contact sensor designed for highway use. It is capable of detecting the acoustic emissions of a vehicle and providing vehicle presence signals to a traffic controller or other system. Each SmartSonic sensor is comprised of a microphone array which listens continuously to sound energy emitted from vehicles or other sources within its detection zone. The signals from the microphones are processed to provide sensor directivity, creating an effective detection beam of only a few degrees. Only sounds coming from within a specific detection zone are retained. Sounds from locations outside the detection zone (such as an adjacent highway or freeway) are severely attenuated and are ignored. The detection zone size and shape is determined by the sensor installation geometry. For the typical installation along the Union Pacific railroad track, the detection zone is approximately a 12' x 12' area.

When a train enters the detection zone, an increase in sound energy is detected and a train presence signal is generated. This signal is used to close a relay, providing power to the transmitter of the radar. When the train leaves the detection zone, the sound energy level drops below the detection threshold, and the train presence signal becomes inactive. During tests it was noted that the sound level may drop below the detection level for short periods of time. In order to prevent the radar power from being turned on and off intermittently as a train passes, a time delay relay was used to hold the radar on for the short quiet intervals as a train passes.

The SmartSonic TSS-1 microphone array is mounted overhead on utility poles just beside the enclosure holding the radar and modem. Each TSS-1 sensor has a single detection zone, and is aimed at the railroad tracks at approximately the same location as the radar.

Each TSS-1 sensor is small and lightweight to facilitate easy installation using off-the-shelf mounting hardware. The sensor utilizes a fully programmable digital signal processor (DSP) to process the microphone signals. Selectable processing bands provide flexibility to control detection zone size for different installation geometries. For AWARD, the sensors have been set to operate in the lowest frequency band to provide the greatest sensitivity to the low frequency sounds generated by the trains.

The cable (termed the ? home run? cable) from the TSS-1 acoustic sensor is brought through the bottom of the pole mounted enclosure at a sealed fitting. The cable terminates at screw connectors on the transition module which serves as the junction between the home run cable and the six conductor modular cables connected to the TSS-1 controller card. The Transition Module is mounted on the case containing the controller. Signal probe points on the transition module are readily accessible to facilitate trouble shooting if it is necessary.

The TSS-1 controller uses a programmable microprocessor to implement detection processing. The controller provides vehicle presence relay signals at the controller edge connector and visual LED detection indicators on the front panel. A complete list of SmartSonic TSS-1 specifications is provided in Table 1.

The time delay relay used to control the radar RF beam when the acoustic signal of a train is present is a Syrelec Chronos multifunction timer. It operates from a 12 volt DC supply and can provide time delays from 0.1 second to 10 hours. For this use, the time delay is set to 15 seconds.

SENSOR	
Detection Method	Passive (Non-Emitting)
Detection Frequency Band	1 of 4 bands
Detection Beam Pattern	3dB Beamwidth of 8 degrees
Mounting Position	Overhead or side mount
Detection Range	(20ft to 40ft) or (6.1 m to 12.2m)
Temperature	(-30°F to 160°F) or (-34°C to 71°C)
Wind Load Design	(120mph) or (190Km/hr)
Weight	Less than (8lb) or (3.62Kg)
Size	15.0" square x 3.0" deep
Color	Gray
Enclosure	Aluminum with baked enamel
Current Requirement	Less than 55mA @ 24VDC
CONTROLLER	
System Interface	Type 170/NEMA cardfile/RS-232C
Relay Contacts	Solid state coupled
Size of Controller Card	4.5"H x 6.875"D
Size (Shelfmount)	6.75"H x 7.94"D x 2.85"W
Power Requirements	Less Than 90 mA @24VDC for Controller Card
SYSTEM	
Power Requirements	12 to 24 VDC
Total Power Consumption	7 watts @24VDC (4 Sensors & Controller)
Sensor Interconnect Cable	Twisted pair cable(4 pairs)
Sensor Home run Cable	Twisted pair cable(4 Pairs)

Table 2. SmartSonic TSS-1 Specifications

The selected doppler radar sensor is Model 3004 from MPH Industries, Inc. manufactured by O’Conner Engineering. Sensor specifications are listed in the table below.

Criteria	Specification
Sensor Type	Doppler Radar
Frequency	24.125 Ghz
Power Output	0.005 Watt
Power Required	2.4 Watts, 10.8-24V, 250 mA at 12V
Size	Approx: 4” x 4” x 9”
Antenna Type	100 mm (4 inch) sealed lens horn
Beam Width	7º
Range	up to 2 miles
Train Response	35 Hz to 15,000 Hz weighted response (0.5 to 185 mph)
Accuracy	± 0.50 mph across response range
Output Signals	target velocity target closing target receding timer output timed relay output
External Communications/ Connector	RS 232 signals (EIA 232) DB 9 connector
System Control	Unit may be placed intro transmit mode by an external RS 232 command. Time to awaken transmitter until transmitting data may take a few seconds.
Environment	Designed for all weather, day/night, continuous operation. -30°C to +60°C (-22°F to 140°F) 90% Relative Humidity at 37°C (99°F)
Lifetime	Designed to last 7 to 10 years. Actual lifetime may vary.
Manufacturers Warranty	2 years
Approval	FCC, Part 15 (No license required.)

Table 3. Doppler Radar Speed Sensor Specifications

The radar communicates over a bidirectional RS-232 interface which is set for 2400 baud, 8 data bits, no parity, I stop bit. When power is first turned on to the radar it is in a dormant state and commands must be issued to start the desired operation. The setup commands for specific actions are:

Desired Action	Command (ASCII)
Turn microwave transmitter on	D40
Turn microwave transmitter off	D41
Send a single speed sample	S
Begin sending continuous samples	B1
Discontinue sending continuous samples	B0
Set sending rate to 4/sec (250 mS period)	T0
Set sending rate to 2/sec (500 mS period)	Tl
Set sending rate to 1/second	T3

Table 4. Doppler Radar Command Codes

Commands sent to the radar unit must be separated by a minimum of 50 milliseconds for proper processing.

Output data format

The radar transmits coded data over the RS-232 port to the modem. Each speed sample is in the form of an output string consisting of 5 groups of ASCII numbers separated by spaces. Groups are defined as follows:

Group 1 - Receding target speed when receding flag = 1
Group 2 - Approaching target speed when approaching flag = 1
Group 3 - Don't care (appears to be uncorrected measurement)
Group 4 - Don't care (normally all l's)
Group 5 - Data Flags (don't care(normally 1), beam off, valid data, don't care (normally 1), approaching target, receding target)

RRR	AAA	XXX	255	Don't Care	Beam Off	Valid Data	Don't Care	Appr. Target	Recd. Target	<cr> <lf>

The target speed data transmitted by the speed sensor is a digital value which must be converted to speed in miles per hour for processing. This is done by the following equation:

Target Speed (MPH) $=.55 + (.318 * N) + (.000257 * N^2)$

Where N is the transmitted speed data (RRR or AAA in the table above)

4.3.2 System Placement

The radar sensor requires a clear view of trains on the track. The unit is most accurate when placed near the track so that the radar beam is aimed directly at an approaching train. In this application, the sensors are located away from the RR track and the beam is aimed at an angle to the sides of the cars.

Since the unit is mounted in a box, a lexan window is provided for the sensor so that the radar unit is relatively protected and the radar beam is barely attenuated.

4.3.3 Power Requirements

The radar unit requires approximately 250mA at 12 V. The supply of power for the unit is described in the Electrical Subsystems Design.

4.3.4 Environmental Design Requirements

The radar unit requires that temperature be maintained between -30°C and +60°C (-22°F and 140°F). The relative humidity at 37°C (99°F) may not exceed 90%. The enclosure providing the environmental conditions is described in the Mechanical Subsystem Design.

4.4 Communications Subsystem Design

The communications subsystem transmits train speed measurements from the train sensor to the workstation located in the TransGuide operational center. Several alternative communication techniques were considered as a part of the system design with the design goals of:

- reliability
- economical installation and operation
- adaptability to installation in multiple locations
- expandability to allow use of many sensors for many grade crossings

Based on system trade-off analysis, dial-up telephone lines were selected for communications between the remote train sensors. This option also allows future inclusion of cellular telephone links if the system is expanded. Wireless telephone service will be the lowest cost for locations where telephone service is not available.

Telephone line communications between the computer located at TransGuide and each MPH train sensor in the field is depicted in Figure 2 (see page 3). The radar unit provides an RS-232 (EIA-232) connection at 2400 baud (8 bit, no parity, 1 stop bit, DB 9 connector). Per the MPH specification, the radar unit is either placed into transmit mode or disabled by an external command. The total process of awakening the radar transmitter, reading the train speed, and transmitting the data takes place in only a few seconds.

A modem configured to automatically answer (and establish communications) is attached to the radar unit and the telephone lines. The selected modem is a Maxtech Net Pacer, selected because it is rated for environmental extremes that may be encountered.

At the TransGuide facility, a Practical Peripherals 28.8 modem provides the other telephone connection. The modem interfaces to the Award Master Computer (a Sun Workstation) via a SCSI interface and provides a standard serial port application interface to the Award Software.

4.4.1 Modem at Remote Site

A number of modems were considered for placement at the remote site. The required specifications include:

Operation to 54° C (130°F)
External modem configuration (i.e. not a computer board)
Normal telephone line operation
Auto-pickup (auto-answer)
2400 baud

4.4.1.1 Specifications

Criteria	Specification
Communication Rate	Up to 33 KB
Type	Stand Alone/External
Digital Interface	RS-232
Telephone Interface	RJ11, RJ45
Environmental	0°C to 55° (32° to 131°F) Humidity 95% non-condensing
Power	120 VAC, 10W nominal
Auto Answer	Yes

Table 5. Modem Specifications

4.4.1.2 Power

The modem requires 115 V AC. The supply of power for the unit is described in the Electrical Subsystems Design.

4.4.1.3 Environmental

The modem requires that temperature be maintained between 0°C and +55°C (32°F to 131°F). The relative humidity must not allow condensation accumulations which would interfere with equipment operation. The enclosure providing the environmental conditions is described in the Mechanical Subsystem Design.

4.5 Electrical Subsystems Design

The remote train speed sensor installation is connected to 115 VAC single phase powre A terminal strip provides a circuit breaker and surge protection. The terminal strip provides 120 VAC power to the modem and a power supply. The power supply provides 15 VDC power for the acoustic detector and the doppler radar sensor. These are commercial, off-the-shelf components and have been selected to have wide operating temperature ranges. The power supply is a Solo model 85-15-2150, chosen to have low heat dissipation to minimize internal heating in the enclosure. Specific electrical parameters are provided in Table 6.

4.6 Mechanical Subsystem Design

The mechanical design includes the enclosure for the remote sensor and electronics.

4.6.1 Enclosure for Sensors

The physical enclosure for the sensing system is based upon current enclosure designs selected and used for traffic control installations in the City of San Antonio. Typical enclosures considered as representative examples are those used at traffic lights to house the signal electronics and power.

4.6.1.1 Design Assumptions

- The control volume is defined about the enclosure.
- The enclosure and internal components are at best at ambient temperature.
- Internal sources of heat are as specified in Table 6.

Component	No Signal Power (W)	Signal Power (W)	Operating Temperature Range
Radar	3	5	-30°C to +60°C (-22°F to +140°F)
SmartSonic TSS-1	2	2	-34°C to 71°C (-30°F to 160°F)
Modem	10	10	0°C to +55°C (32°F to +131°F)
Power Supply	5	5	-25°C to +70°C (-13°F to 158°F)
Total	20	22	

Table 6. Enclosure Internal Power Loading

- Maximum ambient temperature is 49°C (120°F). During summer months at extreme conditions a sun-shield is required. The sun-shield provides protection from the elements, as well. Analysis shows vents will always be needed.

- Minimum ambient temperature is -12°C (10°F). During extreme winter months, low temperatures in San Antonio may exceed component design limits. Analysis shows that internal heating of components in the sealed enclosure will maintain a suitable internal temperature.

4.6.1.2 Mechanical Layout

The dimensions of the enclosure are 12” x 12” x 12”. These dimensions provide ready access to components and ample room for additional components if needed.

The radar window is lexan. The thickness and clarity are designed to prevent thermal loading, and damage due to elements and/or vandalism.

4.6.1.2.1 Alignment for Sensor

Sensor alignment is provided for by a two degree-of-freedom mount. This mount allows for rotation about the pitch and yaw axis of the enclosure. All components are designed for rigid mounting within the enclosure.

4.6.1.3 External Mounting

The enclosure and sun-shield are designed for pole mount. Pole brackets are considered the primary system for installation.

4.6.2 Electrical Connections

Power and communication access are being provided by industry standard electrical and communication external connections. These provide for cabling isolation from the elements.

5. Software System Design

The AWARD Software monitors the field sensors, filtering data from them and detecting the speed and length of trains. The software is cognizant of crossings and sensors downstream from the current sensor of interest. Crossing blockage times and durations are calculated. This information is provided to the Area Wide Database and to the TransGuide ATMS. The discussion which follows describes the software in two sections: the TransGuide Operational Software related to receiving events through the TransGuide system to motorists, the public and emergency vehicles and the Railroad Operational Software related to determining the speed of trains and predicting arrival times at grade crossings and.

5.1 External Interfaces

The AWARD subsystem has eight external interfaces as shown in Figure 5. The following sections describe these external systems in more detail.

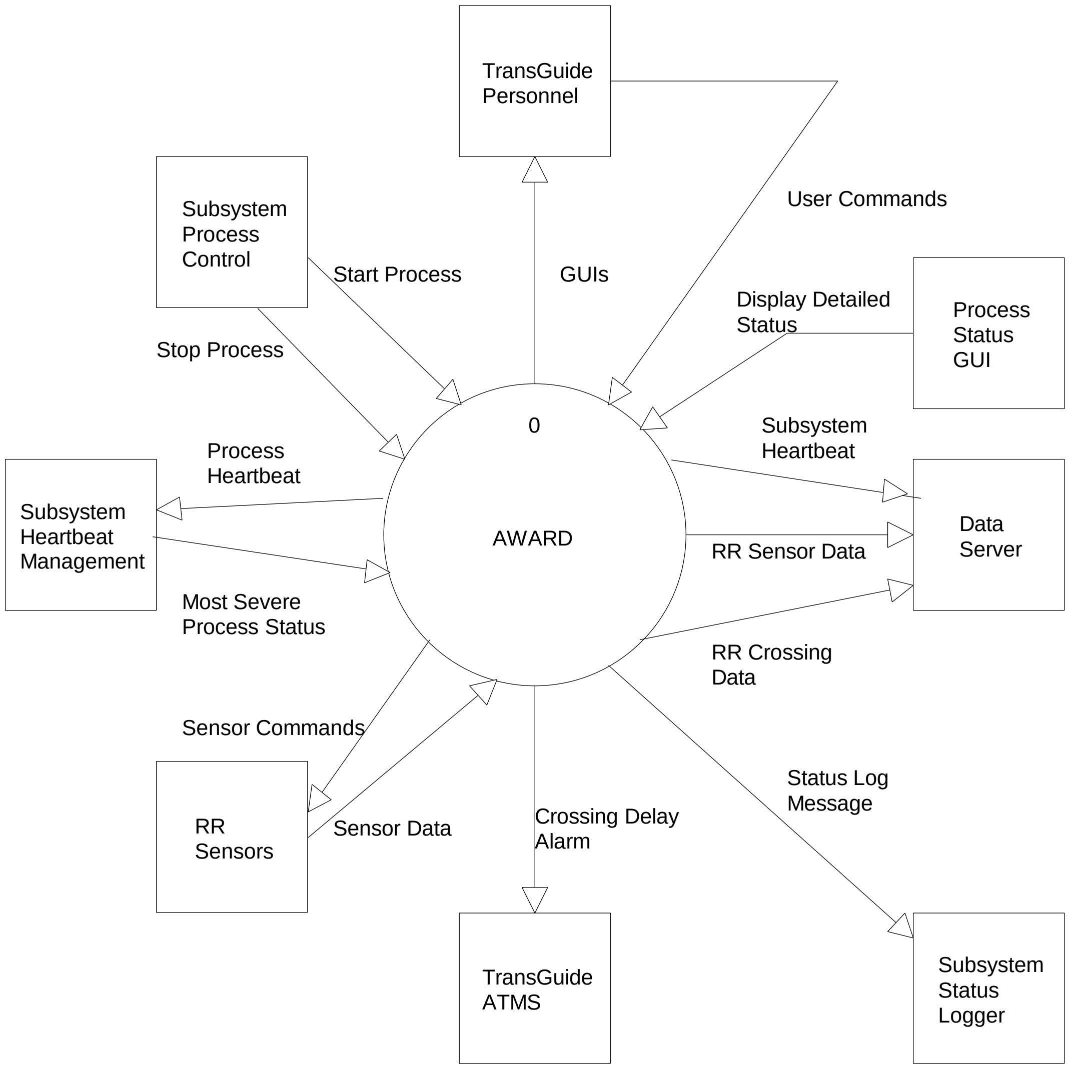


Figure 5. AWARD Context Diagram

5.1.1 TransGuide Personnel

TransGuide Personnel represents the operations and system adminstration personnel assigned to the TransGuide ATMS. These are the end-users of the AWARD subsystem and will interact with the AWARD subsystem via graphical user interfaces associated with the detailed status GUI.

5.1.2 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.

5.1.3 Data Server

Data Server is the central repository of information generated and maintained by the MDI subsystems. The AWARD subsystem sends sensor data and crossing data to the Data Server. The Data Server also receives the subsystem-level heartbeat which includes the overall status of the AWARD subsystem.

5.1.4 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.

5.1.5 TransGuide ATMS

TransGuide ATMS is the existing Advanced Traffic Management System currently in place at the TransGuide facility. Modifications to the TransGuide ATMS have been made to support the AWARD subsystem. These modifications are described in the TransGuide ATMS maintenance manual.

5.1.6 RR Sensors

RR Sensors represent the physical hardware deployed in the field. These sensors are used to detect trains travelling on the monitored tracks.

5.1.7 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.

5.1.8 Subsystem Process Control

Subsystem Process Control is responsible for starting and automatically restarting the processes associated with the AWARD subsystem.

5.1.9 External Data Flows

Several data flows exist between the AWARD subsystem and the external interfaces described above. These data flows are described in more detail inTable 7.

Data Flow	Description
Crossing Delay Alarm	Crossing Delay Alarm is an external alarm (external to TransGuide ATMS) generated by the AWARD subsystem. This alarm indicates changes in delays at a specified crossing. The alarm could represent a crossing being blocked or a clearing of a previously blocked crossing.
Display Detailed Status	Display Detailed Status is an event used to trigger the display of the subsystem's detailed status GUI.
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.
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.
RR Crossing Data	RR Crossing Data is the data associated with the railroad crossing being monitored by the AWARD subsystem. This data includes the railroad crossing identifier, the expected arrival times of the front and rear of a train, and the expected duration of the crossing delay.
RR Sensor Data	Railroad Sensor Data is the data associated with the sensor field equipment. This data includes the sensor identifiers, the current status of the sensors, and the current readings obtained from the sensors.
Sensor Commands	Sensor Commands are commands sent to the field equipment. Initialization sequences are an example of Sensor Commands.
Sensor Data	Sensor Data represents the actual data stream from the Sensors. This data stream is used to detect trains and calculate railroad crossing delays.
Start Process	Start Process is an event used to start the execution of a process.
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.
Stop Process	Stop Process is an event used to stop the execution of a process.
Subsystem Heartbeat	Subsystem Heartbeat is the heartbeat message containing the overall status of the AWARD 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.
User Commands	User Commands are the commands selected by the user from the graphical user interfaces. These commands are generated through push buttons, radio buttons, text boxes, and other user interface components.

Table 7. External Data Flows

5.2 Subsystem Design

The AWARD subsystem software resides on the AWARD master computer and the TransGuide ATMS master computer. The AWARD subsystem consists of four data processes shown inFigure 6. These data processes and associated data flows are described in the following sections.

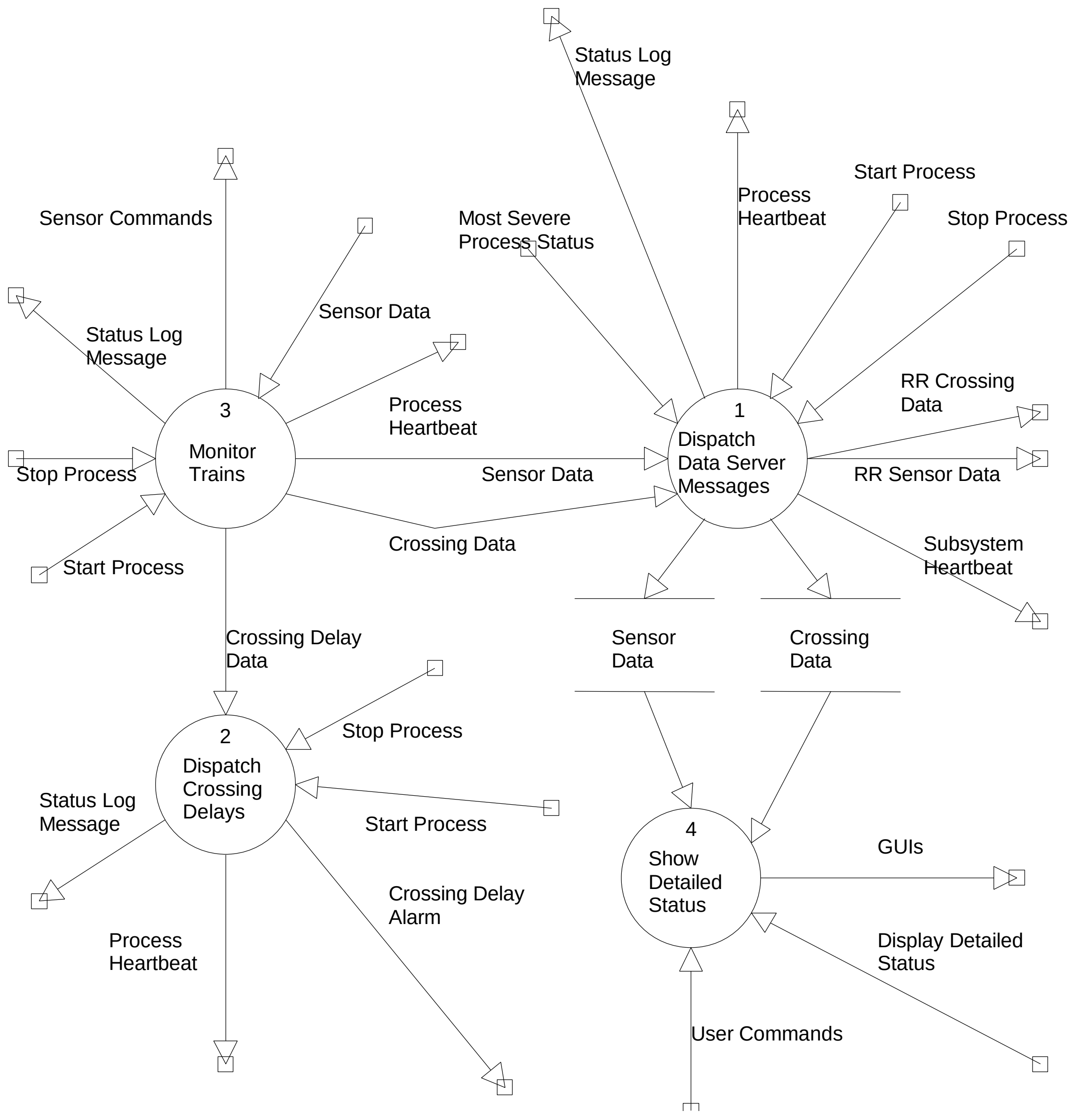


Figure 6. AWARD Subsystem Processes

5.2.1 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 containing the status of the process. This process is also responsible for setting up the Sensor Data and Crossing Data to be displayed by the detail status GUI.

The data flow diagram for the Dispatch Data Server Messages data process is shown inFigure 7. The data processes and associated data flows are described in the subsections that follow.

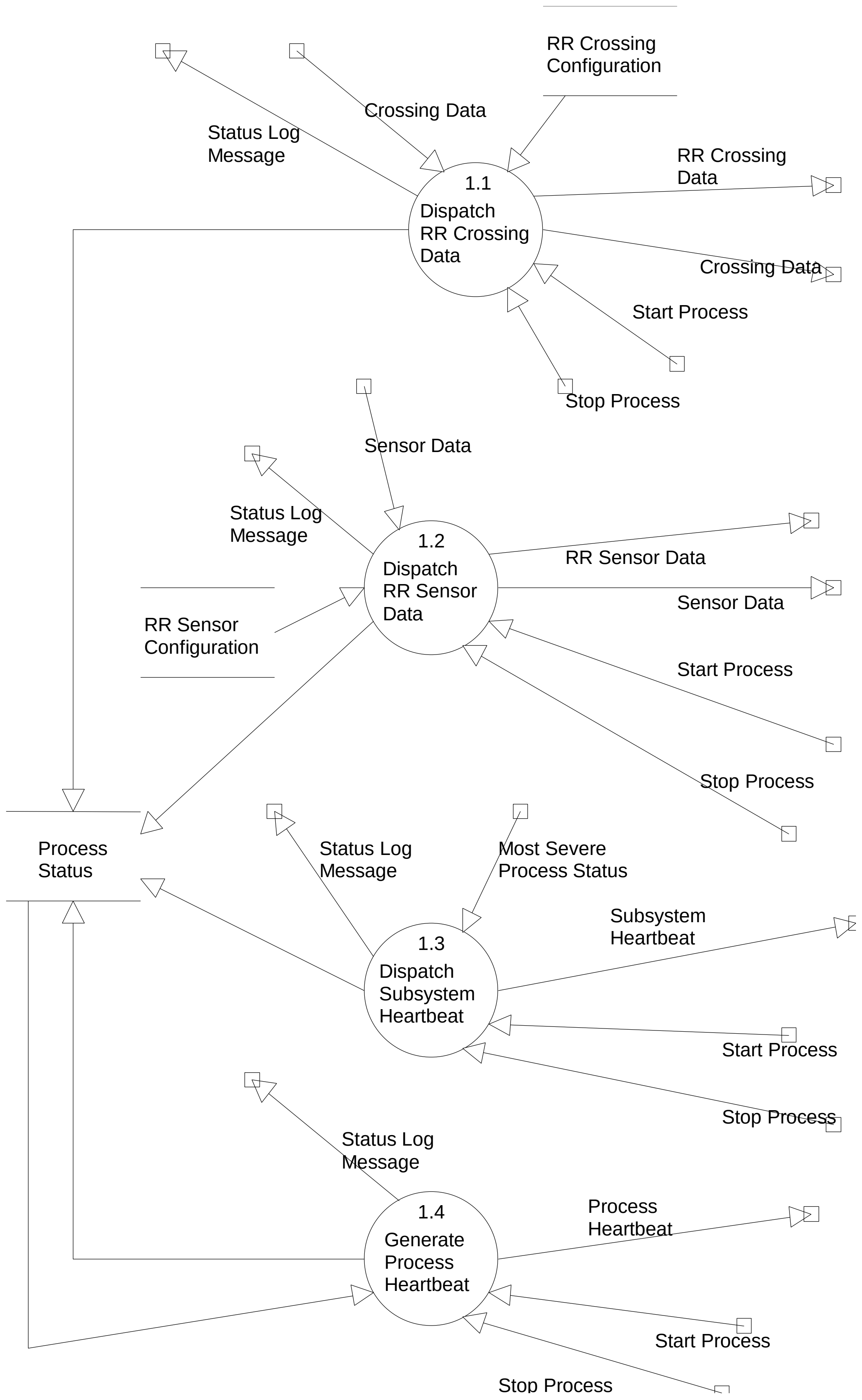


Figure 7. Dispatch Data Server Messages Data Flow

5.2.1.1 Dispatch RR Crossing Data

Dispatch RR Crossing Data is responsible for receiving the Crossing Data from the Monitor Trains data process and sending the RR Crossing Data to the Data Server and storing the Crossing Data for viewing by the detail status GUI. Errors that occur sending the data to the Data Server or in storing the information are logged using Status Log Messages.

The input data flows are described in Table 8 and the output data flows are described in Table 9.

Data Flow	Description
Crossing Data	Crossing Data is the data associated with a specific railroad crossing being monitored by the AWARD subsystem. This information is updated whenever a change is detected at the specific railroad crossing. This information is maintained within the subsystem for the Detailed Status GUI as well as being dispatched to the Data Server to be made available to other MDI subsystems.
RR Crossing Configuration	RR Crossing Configuration contains the equipment IDs for each of the crossings defined for the AWARD system. This information is used to initialize the Crossing Data data store for use by the detail status GUI.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 8. Dispatch RR Crossing Data Input Data Flows

Data Flow	Description
Crossing Data	Crossing Data is the data associated with a specific railroad crossing being monitored by the AWARD subsystem. This information is updated whenever a change is detected at the specific railroad crossing. This information is maintained within the subsystem for the Detailed Status GUI as well as being dispatched to the Data Server to be made available to other MDI subsystems.
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.
RR Crossing Data	RR Crossing Data is the data associated with the railroad crossing being monitored by the AWARD subsystem. This data includes the railroad crossing identifier, the expected arrival times of the front and rear of a train, and the expected duration of the crossing delay.
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.

Table 9. Dispatch RR Crossing Data Output Data Flows

5.2.1.2 Dispatch RR Sensor Data

Dispatch RR Sensor Data is responsible for receiving the Sensor Data from the Monitor Trains Data process and sending the RR Sensor Data to the Data Server and storing the Sensor Data for viewing by the detail status GUI. Errors that occur sending the data to the Data Server or in storing the information are logged using Status Log Messages.

The input data flows are described in Table 10 and the output data flows are described in Table 11.

Data Flow	Description
RR Sensor Configuration	RR Sensor Configuration contains the equipment IDs for each of the sensors being monitored by AWARD. This information is used to initialize the Sensor Data data store for use by the detail status GUI.
Sensor Data	Sensor Data represents the actual data stream from the Sensors. This data stream is used to detect trains and calculate railroad crossing delays.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 10. Dispatch RR Sensor Data Input Data Flows

Data Flow	Description
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.
RR Sensor Data	Railroad Sensor Data is the data associated with the sensor field equipment. This data includes the sensor identifiers, the current status of the sensors, and the current readings obtained from the sensors.
Sensor Data	Sensor Data is the shared information between the AWARD subsystem and the Detailed Status GUI. The Dispatch Data Server Messages process maintains this information and the Detailed Status GUI uses the information to display the current status to the TransGuide ATMS personnel.
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.

Table 11. Dispatch RR Sensor Data Output Data Flows

5.2.1.3 Dispatch Subsystem Heartbeat

Dispatch Subsystem Heartbeat is responsible for receiving the Most Severe Process Status from the Subsystem Heartbeat Management data process and sending the Subsystem Heartbeat to the Data Server. Errors that occur sending the Subsystem Heartbeat to the Data Server are logged using Status Log Messages.

The input data flows are described inTable 12 and the output data flows are described inTable 13.

Data Flow	Description
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.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 12. Dispatch Subsystem Heartbeat Input Data Flows

Data Flow	Description
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.
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 AWARD 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.

Table 13. Dispatch Subsystem Heartbeat Output Data Flows

5.2.1.4 Generate Process Heartbeat

Generate Process Heartbeat periodically sends the Process Heartbeat to the Subsystem Heartbeat Management process. The current Process Status is read and sent as part of the Process Heartbeat. The time interval for sending the Process Heartbeat is specified by the Heartbeat Interval configuration item. Errors and other status information is logged using the Status Log Message.

The input data flows are described in Table 14 and the output data flows are described in Table 15.

Data Flow	Description
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.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 14. Generate Process Heartbeat Input Data Flows

Data Flow	Description
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.
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.

Table 15. Generate Process Heartbeat Output Data Flows

5.2.2 Dispatch Crossing Delays

Dispatch Crossing Delays is responsible for receiving Crossing Delay Data from the Monitor Trains process and generating a Crossing Delay Alarm which is sent to the TransGuide ATMS. This process periodically sends a heartbeat message containing its current status.

The data flow diagram for the Dispatch Crossing Delays data process is shown in Figure 8. The data processes and associated data flows are described in the subsections that follow.

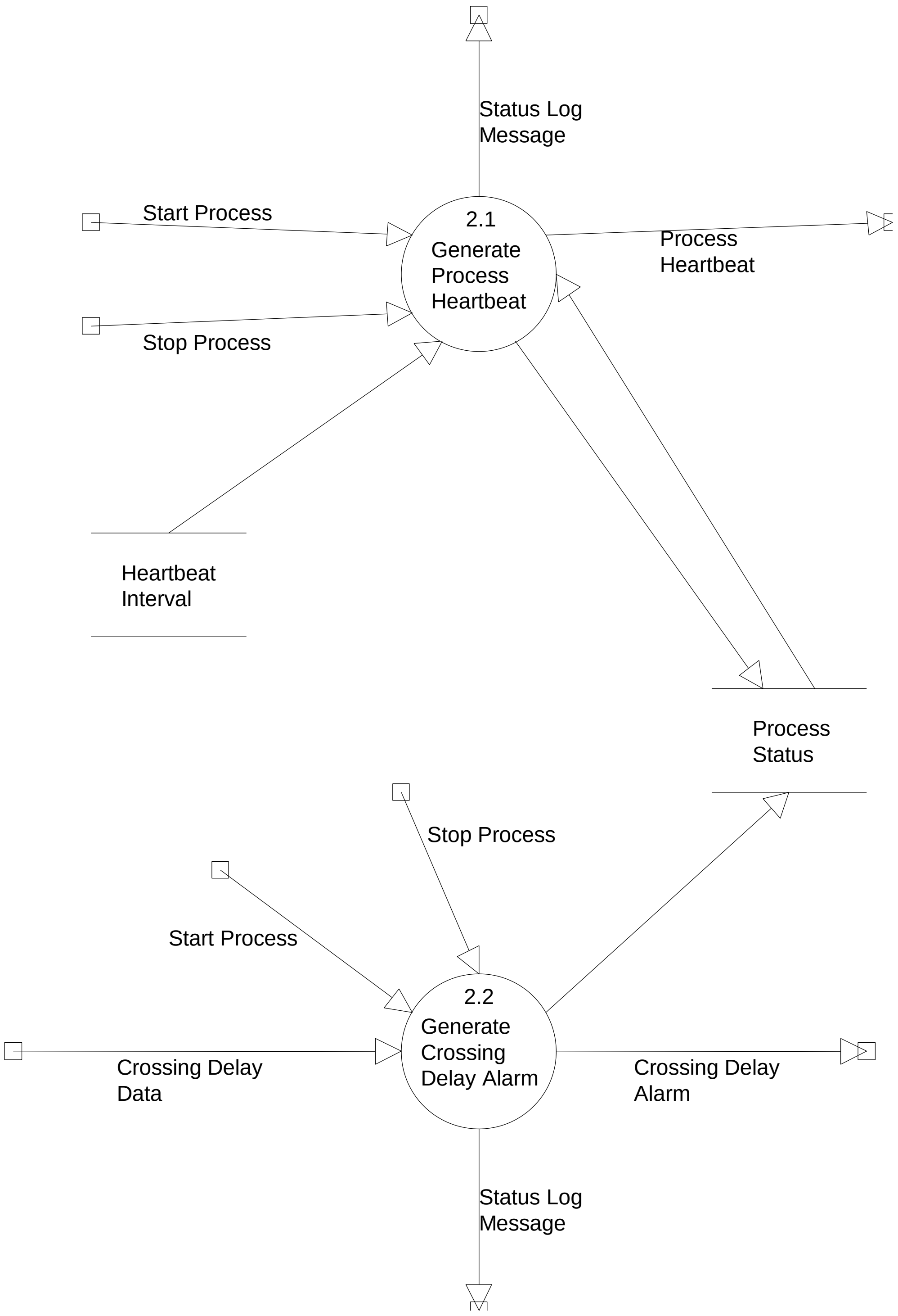


Figure 8. Dispatch Crossing Delays Data Flow

5.2.2.1 Generate Process Heartbeat

Generate Process Heartbeat periodically sends the Process Heartbeat to the Subsystem Heartbeat Management process. The current Process Status is read and sent as part of the Process Heartbeat. The time interval for sending the Process Heartbeat is specified by the Heartbeat Interval configuration item. Errors and other status information is logged using the Status Log Message.

The input data flows are described in Table 16 and the output data flows are described in Table 17.

Data Flow	Description
Heartbeat Interval	Heartbeat Interval is a configuration item that indicates how often the process-level heartbeat message is sent to the Subsystem Heartbeat Management process. This value is specified in seconds.
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.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 16. Generate Process Heartbeat Input Data Flows

Data Flow	Description
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.
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.

Table 17. Generate Process Heartbeat Output Data Flows

5.2.2.2 Generate Crossing Delay Alarm

Generate Crossing Delay Alarm receives the Crossing Delay Data and submits the Crossing Delay Alarm to the TransGuide ATMS. Errors and other status information are logged using the Status Log Message.

The input data flows are described in Table 18 and the output data flows are described in Table 19.

Data Flow	Description
Crossing Delay Data	Crossing Delay Data contains the information needed to generate the Crossing Delay Alarm to be sent to the TransGuide ATMS. This data is used to create new Crossing Delay Alarms for the TransGuide ATMS as well as update existing Crossing Delay Alarms.
Start Process	Start Process is an event used to start the execution of a process.
Stop Process	Stop Process is an event used to stop the execution of a process.

Table 18. Generate Crossing Delay Alarm Input Data Flows

Data Flow	Description
Crossing Delay Alarm	Crossing Delay Alarm is an external alarm (external to TransGuide ATMS) generated by the AWARD subsystem. This alarm indicates changes in delays at a specified crossing. The alarm could represent a crossing being blocked or a clearing of a previously blocked crossing.
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.
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.

Table 19. Generate Crossing Delay Alarm Output Data Flows

5.2.3 Monitor Trains

Monitor Trains is the process which communicates directly with the sensor field equipment and uses the sensor information to predict locations of trains and expected crossing delays within the monitored railroad sections. This process periodically sends a heartbeat message containing the current status of the process. Full details of this process can be found in Section 5.4.2.

5.2.4 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 railroad sensors and crossings being monitored by the AWARD subsystem.

The data flow diagram for the Show Detailed Status data process is shown inFigure 9. The data processes and associated data flows are described in the subsections that follow.

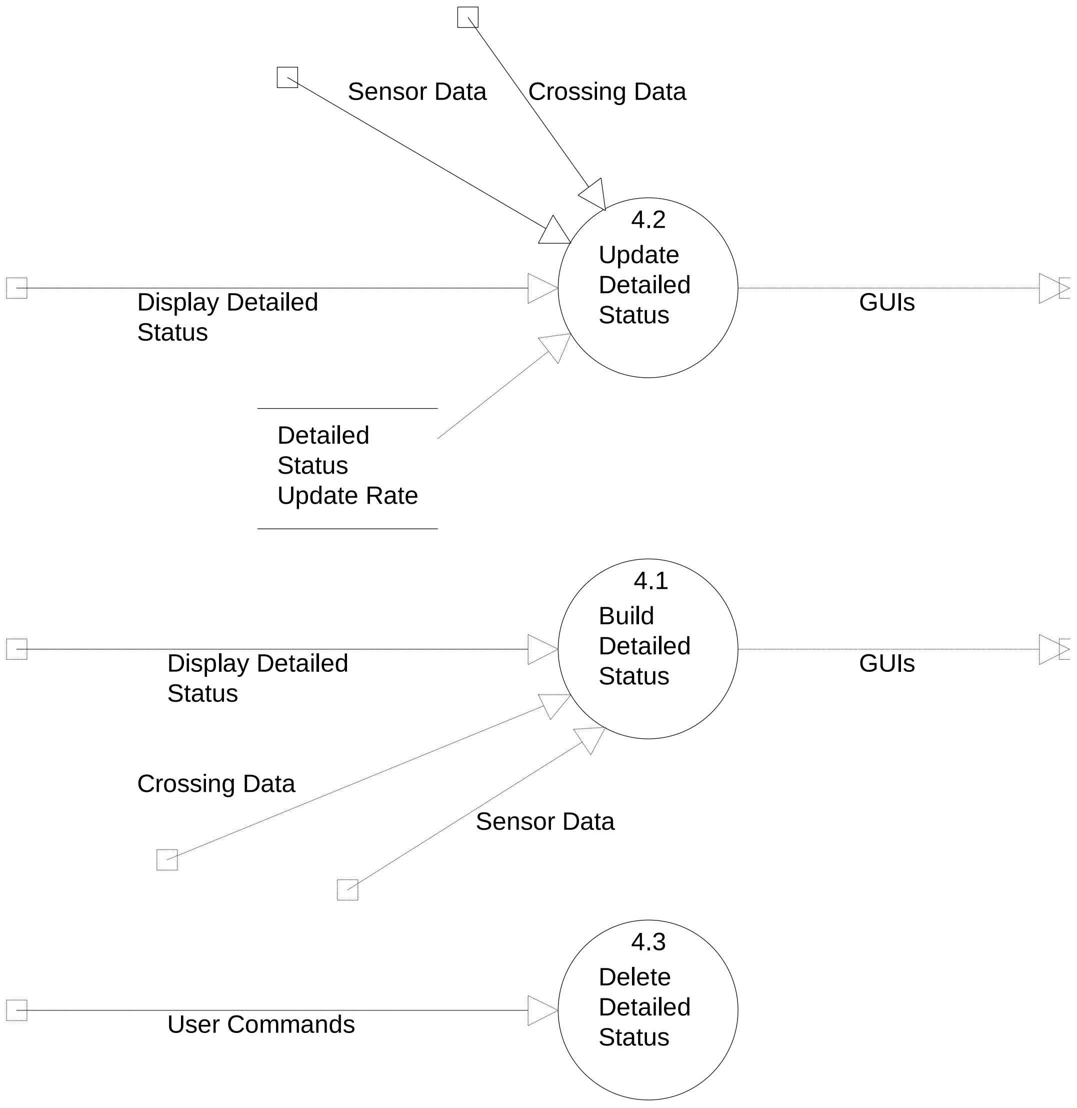


Figure 9. Show Detailed Status Data Flow

5.2.4.1 Build Detailed Status

Build Detailed Status is responsible for generating the initial graphical user interface displaying the AWARD subsystem detailed status. The detailed status includes the current status and readings of each of the sensor field equipment and the current values associated with each of the railroad crossings being monitored by the AWARD subsystem. The current Sensor Data and Crossing Data is used to fill in the details displayed to the TransGuide Personnel.

The input data flows are described in Table 20 and the output data flows are described in Table 21.

Data Flow	Description
Crossing Data	Crossing Data is the shared information between the AWARD subsystem and the Detailed Status GUI. This information is maintained by the Dispatch Data Server Messages process and is used by the Detailed Status GUI to provide the current information to the TransGuide Personnel.
Display Detailed Status	Display Detailed Status is an event used to trigger the display of the subsystem's detailed status GUI.
Sensor Data	Sensor Data is the shared information between the AWARD subsystem and the Detailed Status GUI. The Dispatch Data Server Messages process maintains this information and the Detailed Status GUI uses the information to display the current status to the TransGuide ATMS personnel.

Table 20. Build Detailed Status Input Data Flows

Data Flow	Description
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.

Table 21. Build Detailed Status Output Data Flows

5.2.4.2 Update Detailed Status

Update Detailed Status is responsible for periodically updating the status information within the detailed status GUI. The current Crossing Data and Sensor Data is read and used to display the status within the GUI. The Detailed Status Update Rate is used to cause the periodic update of the GUI.

The input data flows are described in Table 22 and the output data flows are described in Table 23.

Data Flow	Description
Crossing Data	Crossing Data is the shared information between the AWARD subsystem and the Detailed Status GUI. This information is maintained by the Dispatch Data Server Messages process and is used by the Detailed Status GUI to provide the current information to the TransGuide Personnel.
Detailed Status Update Rate	Detailed Status Update Rate is the configuration item that specifies how often the contents of the detailed status GUI are updated. This update rate is specified in seconds.
Display Detailed Status	Display Detailed Status is an event used to trigger the display of the subsystem's detailed status GUI.
Sensor Data	Sensor Data is the shared information between the AWARD subsystem and the Detailed Status GUI. The Dispatch Data Server Messages process maintains this information and the Detailed Status GUI uses the information to display the current status to the TransGuide ATMS personnel.

Table 22. Update Detailed Status Input Data Flows

Data Flow	Description
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.

Table 23. Update Detailed Status Output Data Flows

5.2.4.3 Delete Detailed Status

Delete Detailed Status deletes the detailed status GUI from the display. This process is invoked when the TransGuide personnel issue the "close" command for the detailed status GUI.

The input data flows are described in Table 24. Delete Detailed Status removes the detailed status GUI from the display so there are no associated output data flows.

Data Flow	Description
User Commands	User Commands are the commands selected by the user from the graphical user interfaces. These commands are generated through push buttons, radio buttons, text boxes, and other user interface components.

Table 24. Delete Detailed Status Input Data Flows

5.3 TransGuide Subsystem Software Architecture

The AWARD subsystem is composed of four processes that interact in order to

- monitor the sensors and railroad crossings,
- disseminate the current sensor readings and railroad crossing data to the Data Server,
- generate external alarms to warn TransGuide personnel of railroad crossing delays, and
- provide the TransGuide personnel with the ability to easily view the sensor and crossing information.

The four processes are shown in the data flow diagram in Figure 6 on page 31. The software design for dispatch data server messages (award_dsif), dispatch crossing delays (award_tgif), and show detailed status (awdsg) are presented in the following subsections. The software design for monitor trains is presented in Section 5.4.

The award_dsif and award_tgif processes also have a related set of library routines to be used by other processes to interact with the award_dsif and the award_tgif. These libraries will be discussed following the discussions of each process.

5.3.1 Dispatch Data Server Messages (award_dsif)

The award_dsif process provides the single point of interface between the AWARD subsystem and the Data Server. award_dsif is responsible for receiving messages from the other processes in the AWARD subsystem and directing these messages to the Data Server.

5.3.1.1 main

The structure chart for the main routine is shown in Figure 10. The 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 project-level heartbeat process, and responding to requests made by the other processes within the AWARD subsystem. A description of the routines called by the main routine of award_dsif is provided in Table 25.

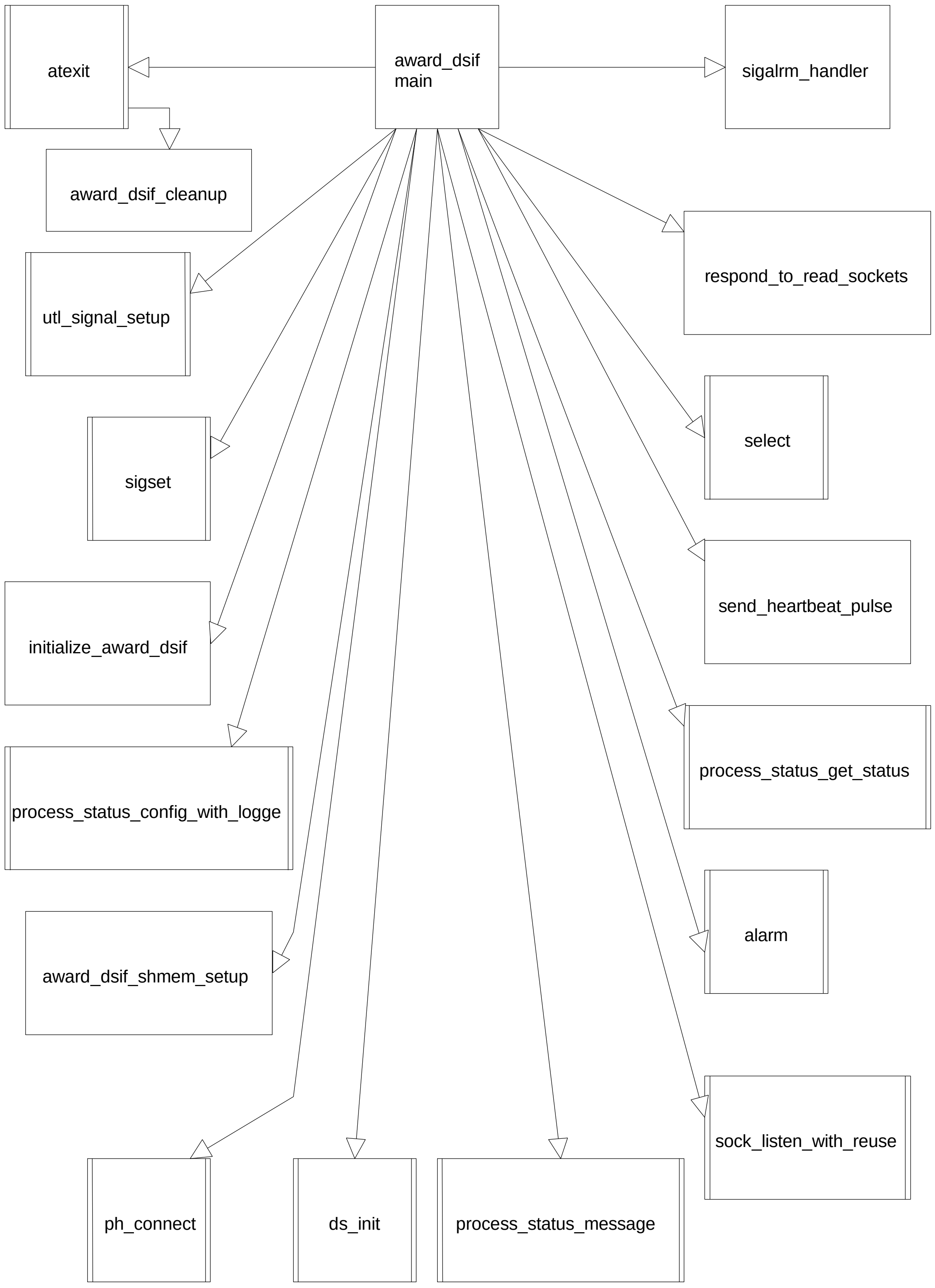


Figure 10. award_dsif main structure chart

Routine	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.
award_dsif main	The award_dsif main routine is responsible for setting up configuration information, opening the socket used for communication, and connecting to the status logger. This routine enters a loop waiting for data server messages and periodically sending heartbeat messages to the subsystem heartbeat process.
award_dsif_cleanup	Called when award_dsif exits. This routine is responsible for performing the housekeeping necessary for a graceful shutdown. This includes sending a last heartbeat, disconnecting from the process-level heartbeat service, disconnecting from the Data Server, and closing any sockets that are open for communicating with the award_dsif process.
award_dsif_shmem_setup	Responsible for setting up the shared memory segment for the AWARD field equipment. There are two shared memory segments for the field equipment. One for the railroad sensors and one for the railroad crossings.
ds_init	MDI Data Server library routine used to initialize the connection to the Data Server.
initialize_award_dsif	The award_dsif configuration file specified on the command line is read to obtain the values of the configurable items of the award_dsif process.
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_logge	process_status_config_with_logger is an MDI Process Status Common routine used to configure the process status handling for the process. This routine is used to set 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.
respond_to_read_sockets	Loops through the list of socket descriptors ready for reading and either accepts connections, if the socket descriptor is for the listen socket, or receives messages containing information to be sent to the Data Server.
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 process-level heartbeat to the Subsystem Heartbeat process.
sigalrm_handler	The signal handler for the SIGALRM signal. This signal is used to indicate when the process-level heartbeat should be sent to the AWARD subsystem heartbeat process. The alarm is reinitialized as part of this routine.
sigset	C Library Function used to modify the disposition of a signal. The signal can be caught, ignored, or returned to the default disposition.
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.
utl_signal_setup	MDI Common Utility Library routine used to set up a default signal handler for all catchable signals.

Table 25. Routines called by award_dsif main

5.3.1.2 award_dsif_cleanup

The award_dsif_cleanup routine is called when the award_dsif process performs a normal termination. This routine performs the necessary housekeeping chores to cause a graceful exit of the award_dsif process. The structure chart for the award_dsif_cleanup routine is shown in Figure 11. A description of the routines called by award_dsif_cleanup is provided in Table 26.

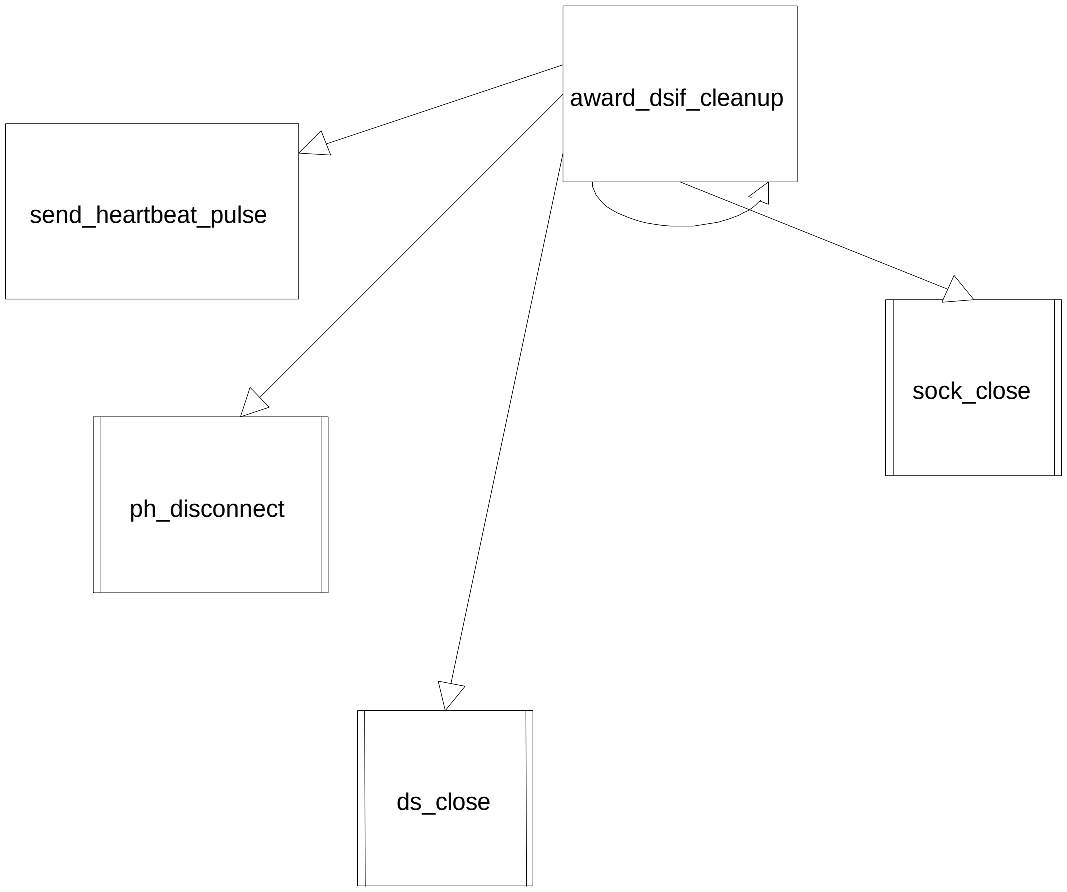


Figure 11. award_dsif_cleanup structure chart

Routine	Description
award_dsif_cleanup	Called when award_dsif exits. This routine is responsible for performing the housekeeping necessary for a graceful shutdown. This includes sending a last heartbeat, disconnecting from the process-level heartbeat service, disconnecting from the Data Server, and closing any sockets that are open for communicating with the award_dsif process.
ds_close	MDI Data Server routine used to close the connection to the Data Server.
ph_disconnect	MDI Process Heartbeat routine used to disconnect from the process-level heartbeat service.
send_heartbeat_pulse	Sends the process-level heartbeat to the Subsystem Heartbeat process.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 26. Routines called by award_dsif_cleanup

5.3.1.3 send_heartbeat_pulse

The send_heartbeat_pulse 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 12. The descriptions of the routines called by send_heartbeat_pulse are contained in Table 27.

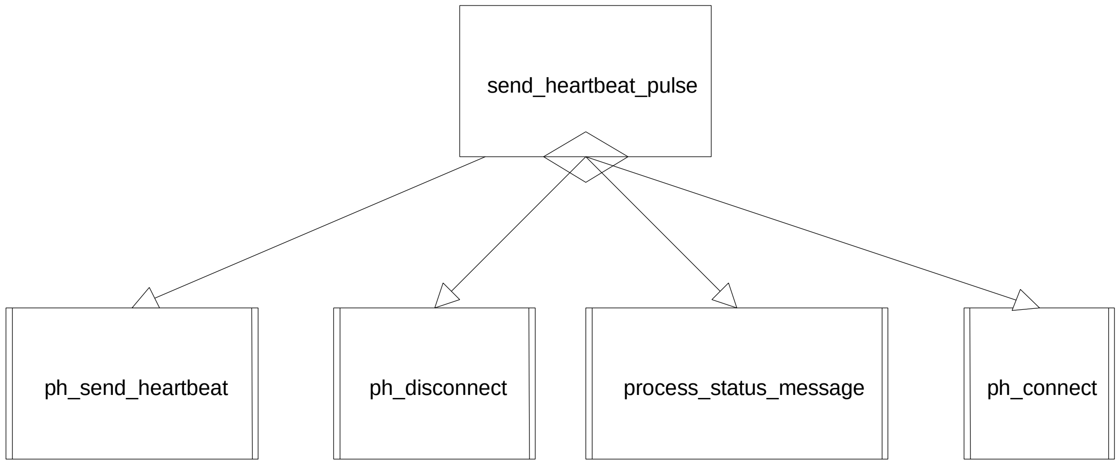


Figure 12. send_heartbeat_pulse structure chart

Routine	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 process-level heartbeat to the Subsystem Heartbeat process.

Table 27. Routines called by send_heartbeat_pulse

5.3.1.4 initialize_award_dsif

The initialize_award_dsif routine is called to read the award_dsif configuration file and set up configuration information for the entire process. The structure chart for initialize_award_dsif is shown in Figure 5. Descriptions of the routines called by initialize_award_dsif are contained inTable 28. Configurable items for the award_dsif process are described inTable 29.

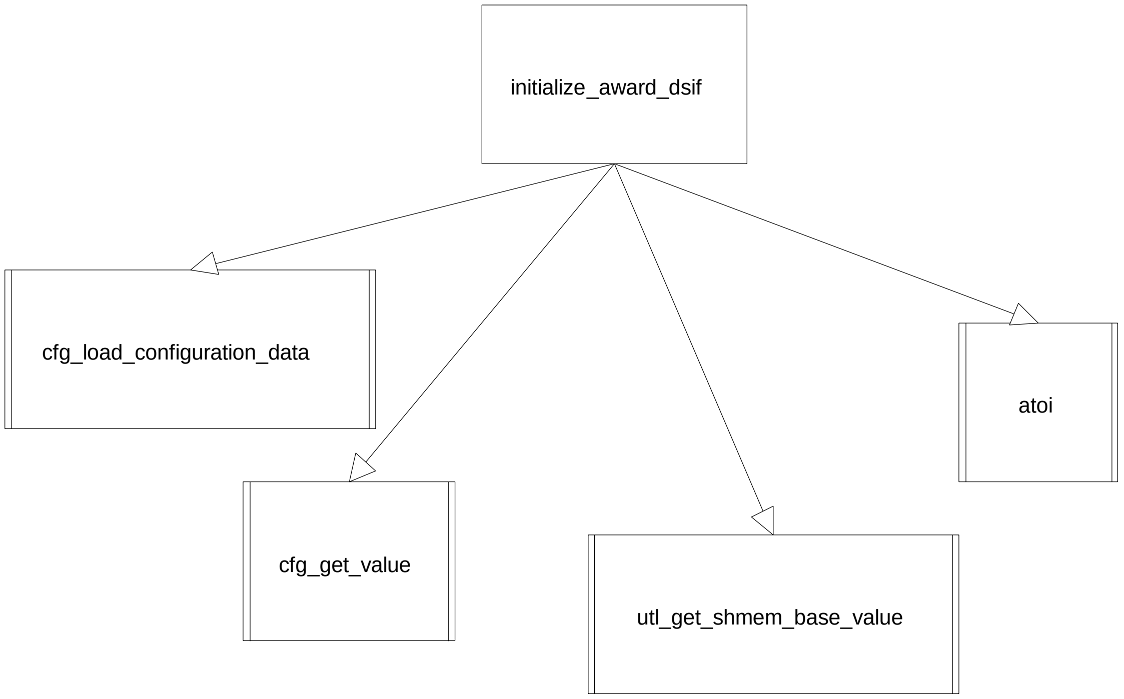


Figure 13. initialize_award_dsif structure chart

Routine	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.
initialize_award_dsif	The award_dsif configuration file specified on the command line is read to obtain the values of the configurable items of the award_dsif process.
utl_get_shmem_base_value	MDI Utility routine used to convert a text string containing a shared memory base value name to the actual base value used by the shared memory routines.

Table 28. Routines called by initialize_award_dsif

Configuration Item	Description	Optional
SERVICE_NAME	The name of the service providied by the award_dsif process.	N
HEARTBEAT_SERVICE_NAME	The name of the service provided by the AWARD project-level heartbeat process.	N
HEARTBEAT_HOST_NAME	The host name where the AWARD project-level heartbeat process resides.	Y
STATUS_LOGGER_SERVICE_NAME	The name of the service provided by the AWARD subsystem status logger process.	N
STATUS_LOGGER_HOST_NAME	The host name where the AWARD subsystem status logger process resides	Y
HEARTBEAT_PULSE	The periodic time value for sending the heartbeat to the AWRAD project-level heartbeat process. This is specified in seconds.	Y
DATASERVER_SERVICE_NAME	The name of the service provided by the data server process.	N
DATASERVER_HOST_NAME	The host name where the data server process resides.	Y
AWARD_SHM_BASE	The name of the constant or an integer value indicating the starting base for the AWARD shared memory segments.	N
SENSOR_SEGMENT_NUMBER	The segment number of the sensor shared memory segment.	N
CROSSING_SEGMENT_NUMBER	The segment number of the crossing shared memory segment.	N
NUM_SHMEM_SEGMENTS	The total number of shared memory segments used by the AWARD subsystem.	N
AWARD_RR_MASTER_CFG	The name of the configuration file containing the information about the sensors and crossings defined for the AWARD subsystem.	N

Table 29. award_dsif configuration items

5.3.1.5 award_dsif_shmem_setup

The award_dsif_shmem_setup routine is responsible for configuring the shared memory manager library routines, setting up the sensor and crossing shared memory segments, and then loading and sorting the sensor and crossing information within the segments. The structure chart for the award_dsif_shmem_setup routine is shown in Figure 14. A description of the routines called by award_dsif_shmem_setup is provided inTable 30.

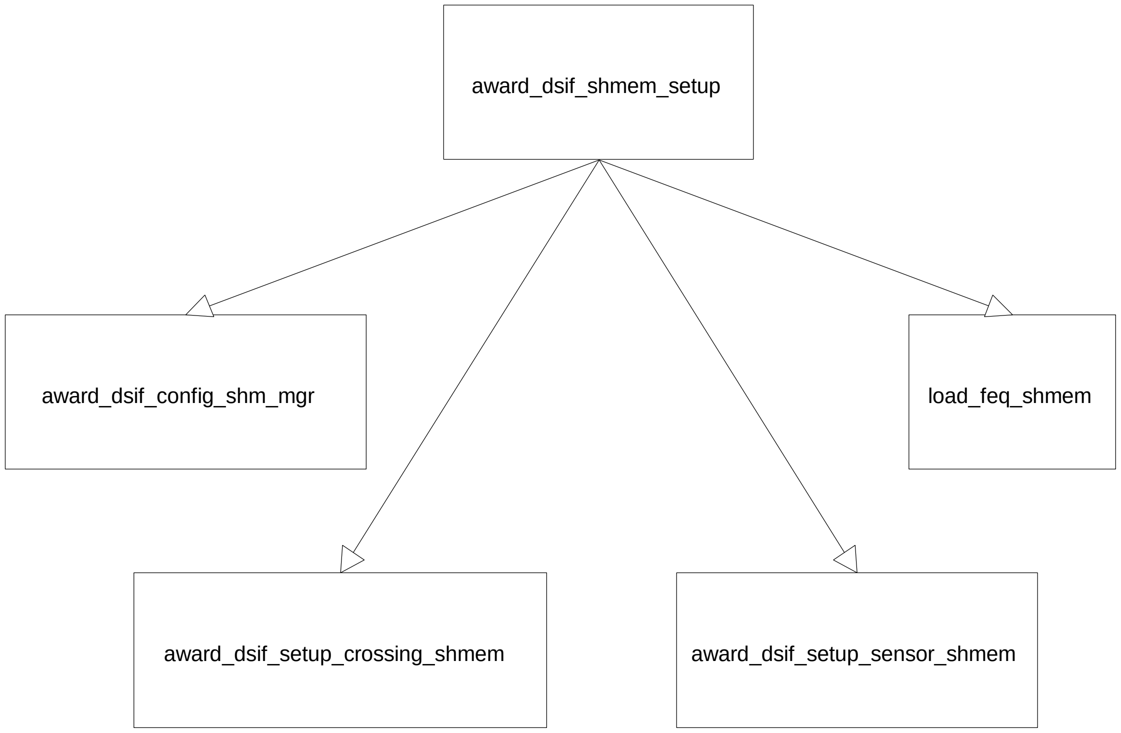

award_dsif_shmem_setup
award_dsif_config_shm_mgr
load_feq_shmem
award_dsif_setup_crossing_shmem
award_dsif_setup_sensor_shmem

Figure 14. award_dsif_shmem_setup structure chart

Routine	Description
award_dsif_config_shm_mgr	Responsible for initializing and configuring the MDI Shared Memory Manager library routines.
award_dsif_setup_crossing_shmem	Responsible for reading the field equipment configuration files to determine the number of crossings being monitored and to initialize the shared memory segment data.
award_dsif_setup_sensor_shmem	Responsible for reading the field equipment configuration files to determine the number of sensors being monitored and to initialize the shared memory segment data.
award_dsif_shmem_setup	Responsible for setting up the shared memory segment for the AWARD field equipment. There are two shared memory segments for the field equipment. One for the railroad sensors and one for the railroad crossings.
load_feq_shmem	Reads the contents of the sensor shared memory segment and the crossing shared memory segment and sorts them in ascending order by address. This allows for easier updates to and retrieval of the information stored within these shared memory segments.

Table 30. Routines called by award_dsif_shmem_setup

5.3.1.6 award_dsif_config_shm_mgr

The award_dsif_config_shm_mgr routine is responsible for initializing the shared memory manager library with the base value of the AWARD shared memory segments and the number of shared memory segments to be maintained. The structure chart for award_dsif_config_shm_mgr is shown in Figure 15. The descriptions of the routines called by award_dsif_config_shm_mgr are contained in Table 31. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

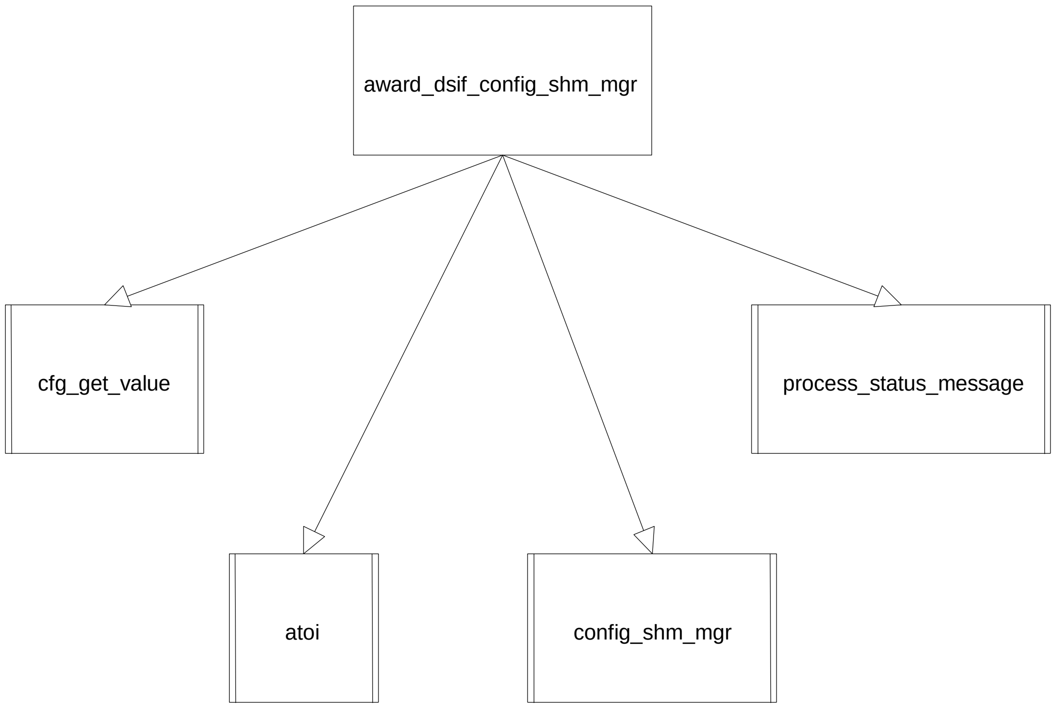


Figure 15. award_dsif_config_shm_mgr structure chart

Routine	Description
atoi	C Library Function to convert an ASCII string to an integer value.
award_dsif_config_shm_mgr	Responsible for initializing and configuring the MDI Shared Memory Manager library routines.
cfg_get_value	MDI Configuration File routine used to return the value of the specified configuration name.
config_shm_mgr	MDI Shared Memory Manager routine used to initialize and configure the shared memory manager library routines for the calling program.
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.

Table 31. Routines called by award_dsif_config_shm_mgr

5.3.1.7 award_dsif_setup_crossing_shmem

The award_dsif_setup_crossing_shmem routine is responsible for creating, attaching, and initializing the shared memory segment associated with the crossing data The crossing configuration file is read in order to determine the number of crossings and their associated ids. The structure chart for award_dsif_setup_crossing_shmem is shown in Figure 16. The descriptions of the routines called by award_dsif_setup_crossing_shmem are contained inTable 32. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

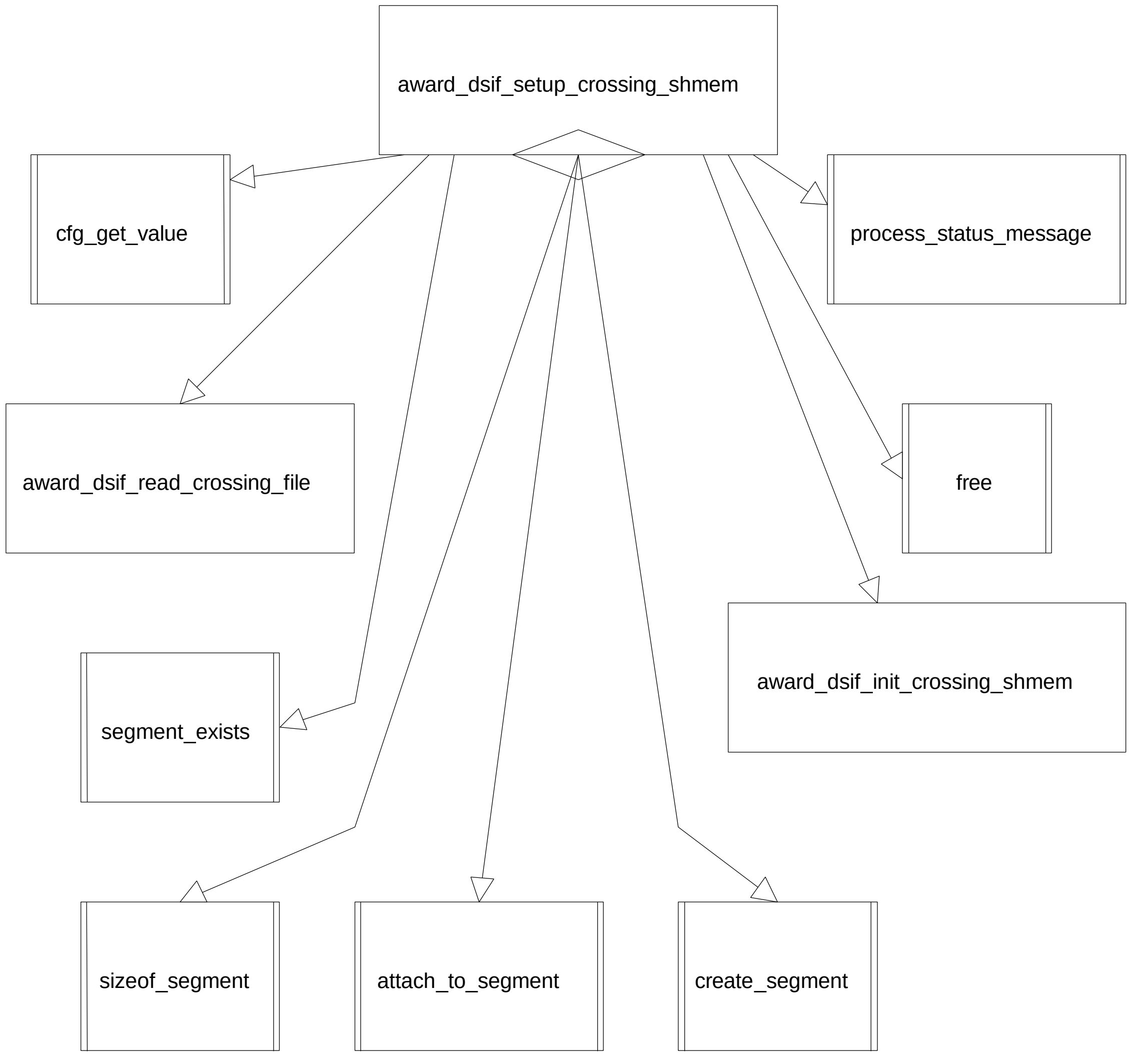


Figure 16. award_dsif_setup_crossing_shmem structure chart

Routine	Description
attach_to_segment	MDI Shared Memory Manager routine used to attach the calling process to the specified shared memory segment.
award_dsif_init_crossing_shmem	Responsible for initializing the crossing shared memory segment based on the number of crossings specified and the list of crossing ids specified.
award_dsif_read_crossing_file	Reads the specified crossing file and builds a comma-delimited list of the names of the crossings currently configured. The memory allocated to the names list must be freed by the calling routine.
award_dsif_setup_crossing_shmem	Responsible for reading the field equipment configuration files to determine the number of crossings being monitored and to initialize the shared memory segment data.
cfg_get_value	MDI Configuration File routine used to return the value of the specified configuration name.
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.
free	C Library Function used to free previously allocated memory and make it available for further allocation.
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.
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.

Table 32. Routines called by award_dsif_setup_crossing_shem

5.3.1.7.1 award_dsif_read_crossing_file

The award_dsif_read_crossing file routine opens the crossing configuration file and parses each line looking for the crossing identifiers. A list of these identifiers is built and the number of crossings found in the file is maintained. This information is used by the calling routine to create the shared memory segments and perform the initialization of these areas. The award_dsif_read_crossing_file routine is made up of numerours C Library Functions. For that reason, no structure chart was produced for this routine.

5.3.1.7.2 award_dsif_init_crossing_shmem

The award_dsif_init_crossing_shmem routine is responsible for clearing the shared memory segment and initializing each of the elements within the crossing shared memory. Each element in the segment corresponds to one crossing configured in the crossing configuration file. The structure chart for award_dsif_init_crossing_shmem is shown in Figure 17. The descriptions of the routines called by award_dsif_init_crossing_shmem are contained in Table 33.

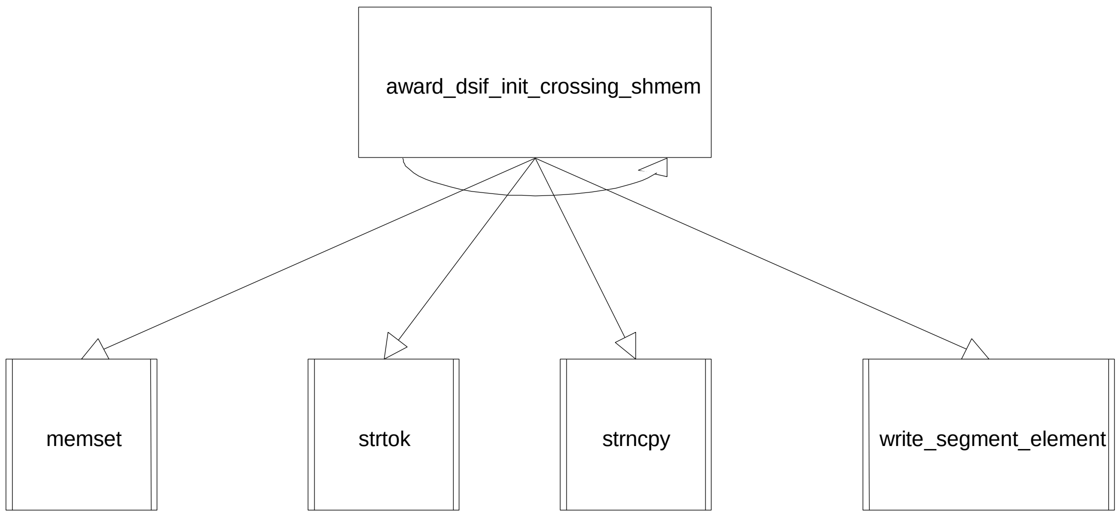


Figure 17. award_dsif_init_crossing_shmem structure chart

Routine	Description
award_dsif_init_crossing_shmem	Responsible for initializing the crossing shared memory segment based on the number of crossings specified and the list of crossing ids specified.
memset	C Library Function used to set an area of memory to a specified value.
strncpy	C Library Function used to copy a specified number of characters from a source string to a destination string.
strtok	C Library Function used to break the specified string into a sequence of tokens.
write_segment_element	MDI Shared Memory Manager function to write information to a specific element in a shared memory segment. In this case the shared memory segment is viewed as an array of elements.

Table 33. Routines called by award_dsif_init_crossing_shem

5.3.1.8 award_dsif_setup_sensor_shmem

The award_dsif_setup_sensor_shmem routine is responsible for creating, attaching, and initializing the shared memory segment associated with the sensor data The sensor configuration file is read in order to determine the number of sensors and their associated ids. The structure chart for award_dsif_setup_sensor_shmem is shown in Figure 18. The descriptions of the routines called by award_dsif_setup_sensor_shmem are contained inTable 34. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

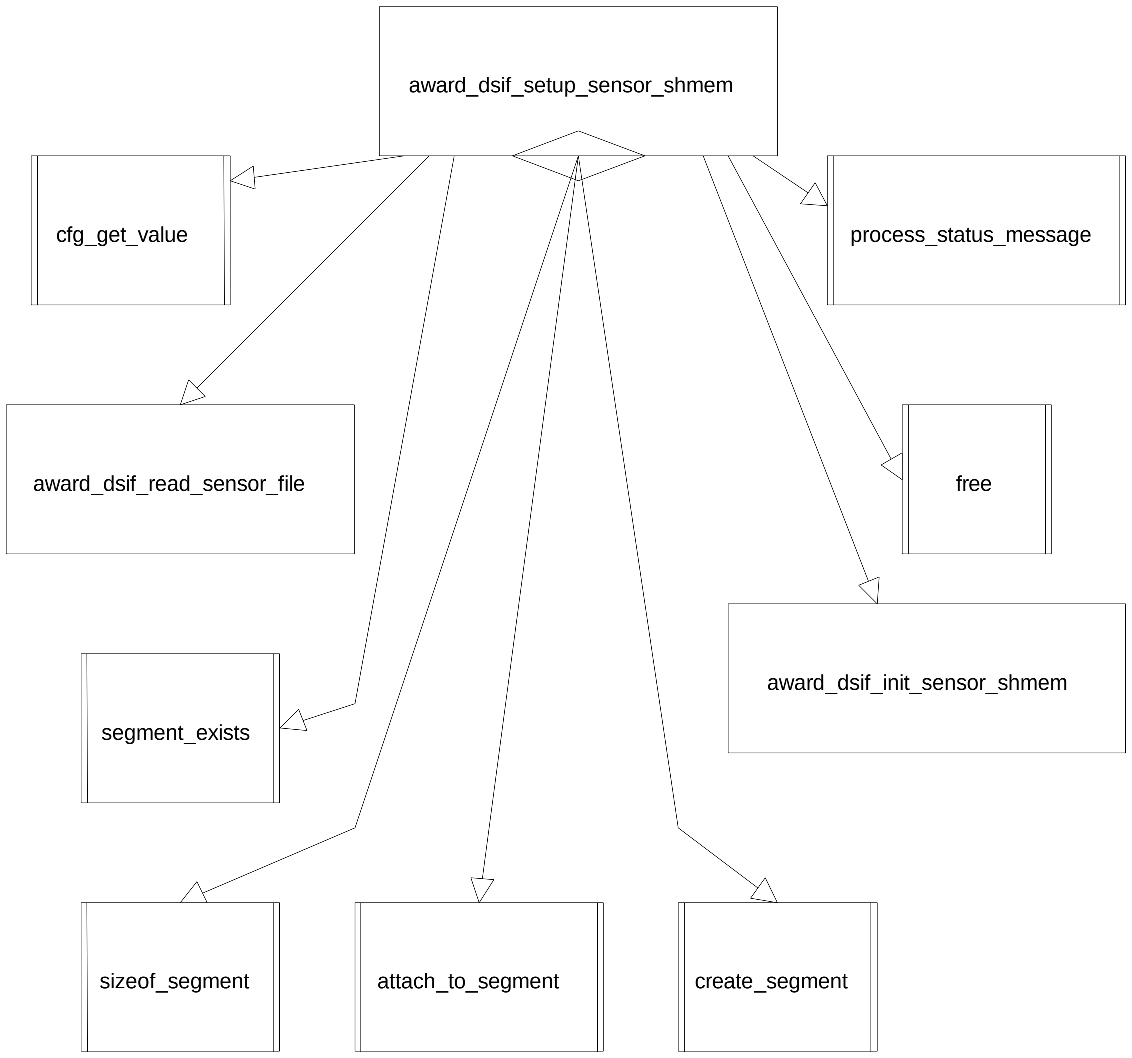


Figure 18 - award_dsif_setup_sensor_shmem structure chart

Routine	Description
attach_to_segment	MDI Shared Memory Manager routine used to attach the calling process to the specified shared memory segment.
award_dsif_init_sensor_shmem	Responsible for initializing the sensor shared memory segment based on the number of sensors specified and the list of sensor ids specified.
award_dsif_read_sensor_file	Reads the specified sensor file and builds a comma-delimited list of the names of the sensors currently configured. The memory allocated to the names list must be freed by the calling routine.
award_dsif_setup_sensor_shmem	Responsible for reading the field equipment configuration files to determine the number of sensors being monitored and to initialize the shared memory segment data.
cfg_get_value	MDI Configuration File routine used to return the value of the specified configuration name.
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.
free	C Library Function used to free previously allocated memory and make it available for further allocation.
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.
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.

Table 34 - Routines called by award_dsif_setup_sensor_shem

5.3.1.8.1 award_dsif_read_sensor_file

The award_dsif_read_sensor_file routine opens the sensor configuration file and parses each line looking for the sensor identifiers. A list of these identifiers is built and the number of sensorss found in the file is maintained. This information is used by the calling routine to create the shared memory segments and perform the initialization of these areas. The award_dsif_read_sensors_file routine is made up of numerous C Library Functions. For that reason, no structure chart was produced for this routine.

5.3.1.8.2 award_dsif_init_sensor_shmem

The award_dsif_init_sensor_shmem routine is responsible for clearing the shared memory segment and initializing each of the elements within the sensor shared memory. Each element in the segment corresponds to one sensor configured in the sensor configuration file. The structure chart for award_dsif_init_sensor_shmem is shown in Figure 19. The descriptions of the routines called by award_dsif_init_sensor_shmem are contained in Table 35.

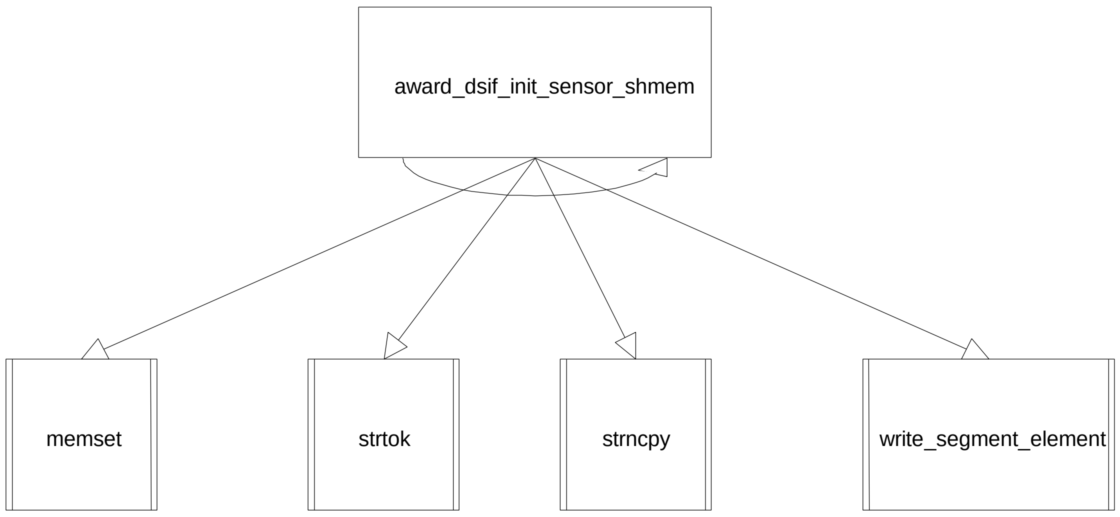


Figure 19. award_dsif_init_sensor_shmem structure chart

Routine	Description
award_dsif_init_sensor_shmem	Responsible for initializing the sensor shared memory segment based on the number of sensors specified and the list of sensor ids specified.
memset	C Library Function used to set an area of memory to a specified value.
strncpy	C Library Function used to copy a specified number of characters from a source string to a destination string.
strtok	C Library Function used to break the specified string into a sequence of tokens.
write_segment_element	MDI Shared Memory Manager function to write information to a specific element in a shared memory segment. In this case the shared memory segment is viewed as an array of elements.

Table 35. Routines called by award_dsif_init_sensor_shmem

5.3.1.9 load_feq_shmem

The load_feq_shmem routine is reads the crossing and sensor shared memory segments into local memory. These lists are then sorted and stored back into the shared memory segments. This is done as an aid in updating the status information for a particular sensor or crossing. The structure chart for load_feq_shmem is shown in Figure 20. The descriptions of the routines called by load_feq_shmem are contained in Table 36. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

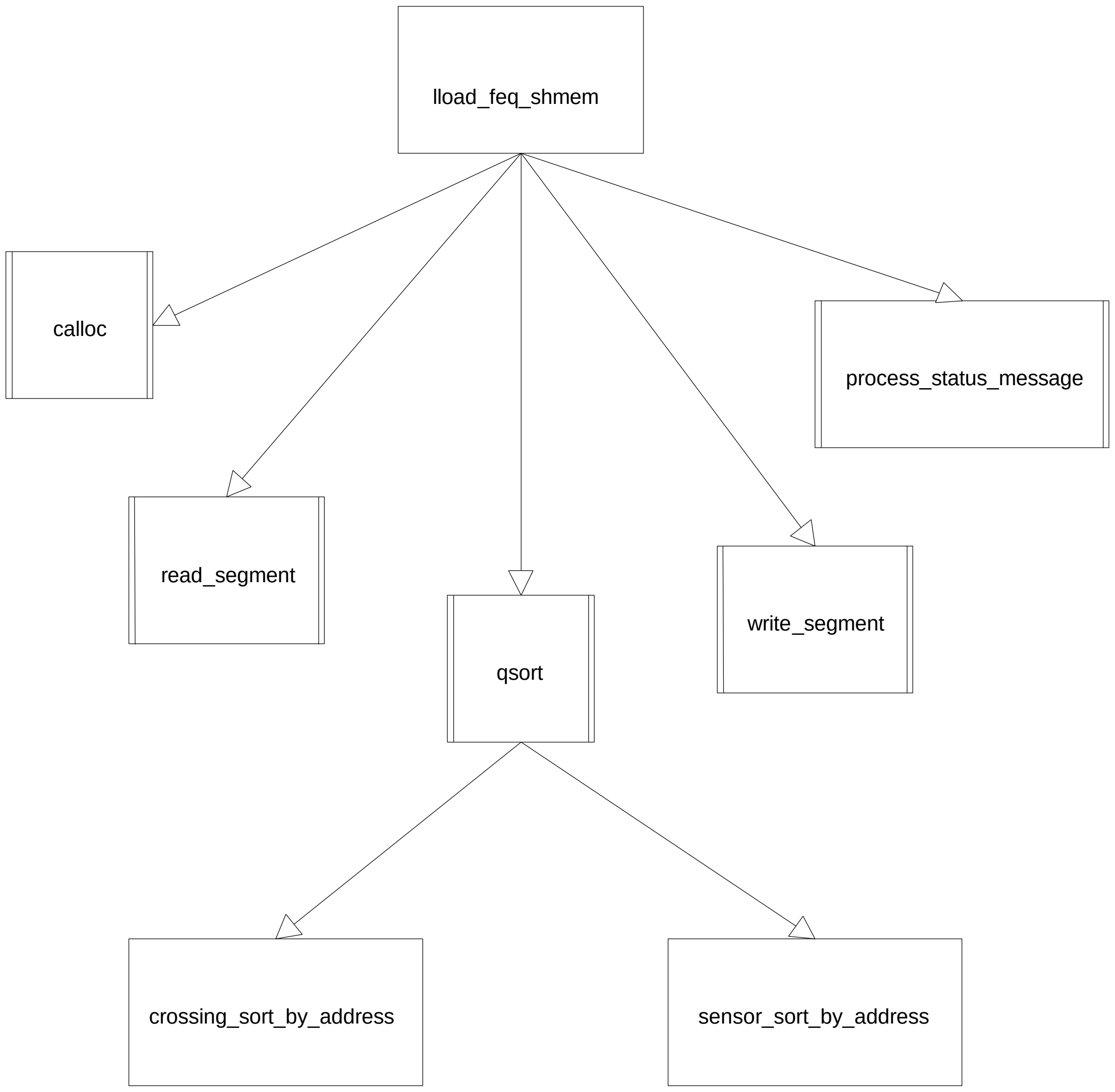


Figure 20. load_feq_shmem structure chart

Routine	Description
calloc	C Library Function to allocate the specified amount of space and fill it with zeros.
crossing_sort_by_address	Comparison routine used in the qsort call to sort the crossing ids in ascending order and used by the bsearch routine to find a match.
load_feq_shmem	Reads the contents of the sensor shared memory segment and the crossing shared memory segment and sorts them in ascending order by address. This allows for easier updates to and retrieval of the information stored within these shared memory segments.
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.
qsort	C Library Function that implements the quick-sort algorithm. The caller supplies the address of the comparison function to be used to sort the data in place.
read_segment	MDI Shared Memory Manager routine to read the contents of the specified shared memory segment. The contents are stored in a memory area allocated by the caller.
sensor_sort_by_address	Comparison routine used in the qsort call to sort the sensor ids in ascending order and used by the bsearch routine to find a match.
write_segment	MDI Shared Memory Manager routine that writes data to the specified shared memory segment.

Table 36. Routines called by award_dsif_setup_crossing_shem

5.3.1.9.1 crossing_sort_by_address

The crossing_sort_by_address routine that returns the result of a string comparison between two crossing ids. This routine is also used during the search for a particular crossing. This routine is made up of only a single call to strcmp. For that reason, no structure chart was produced for this routine.

5.3.1.9.2 sensor_sort_by_address

The crossing_sort_by_address routine that returns the result of a string comparison between two crossing ids. This routine is also used during the search for a particular crossing. This routine is made up of only a single call to strcmp. For that reason, no structure chart was produced for this routine.

5.3.1.10 respond_to_read_sockets

The respond_to_read_sockets routine is heart of the award_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 award_dsif process, a message being sent to the award_dsif process by another process already connected, or it could be an indication of a process that has disconnected from the award_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 award_dsif process the socket connection from the award_dsif process to the disconnected process is closed and removed from the list of active sockets. Errors that occur are logged to the AWARD subsystem status log. The structure chart for the respond_to_read_sockets is shown in Figure 21. A description of the routines called by respond_to_read_sockets is provided in Table 37.

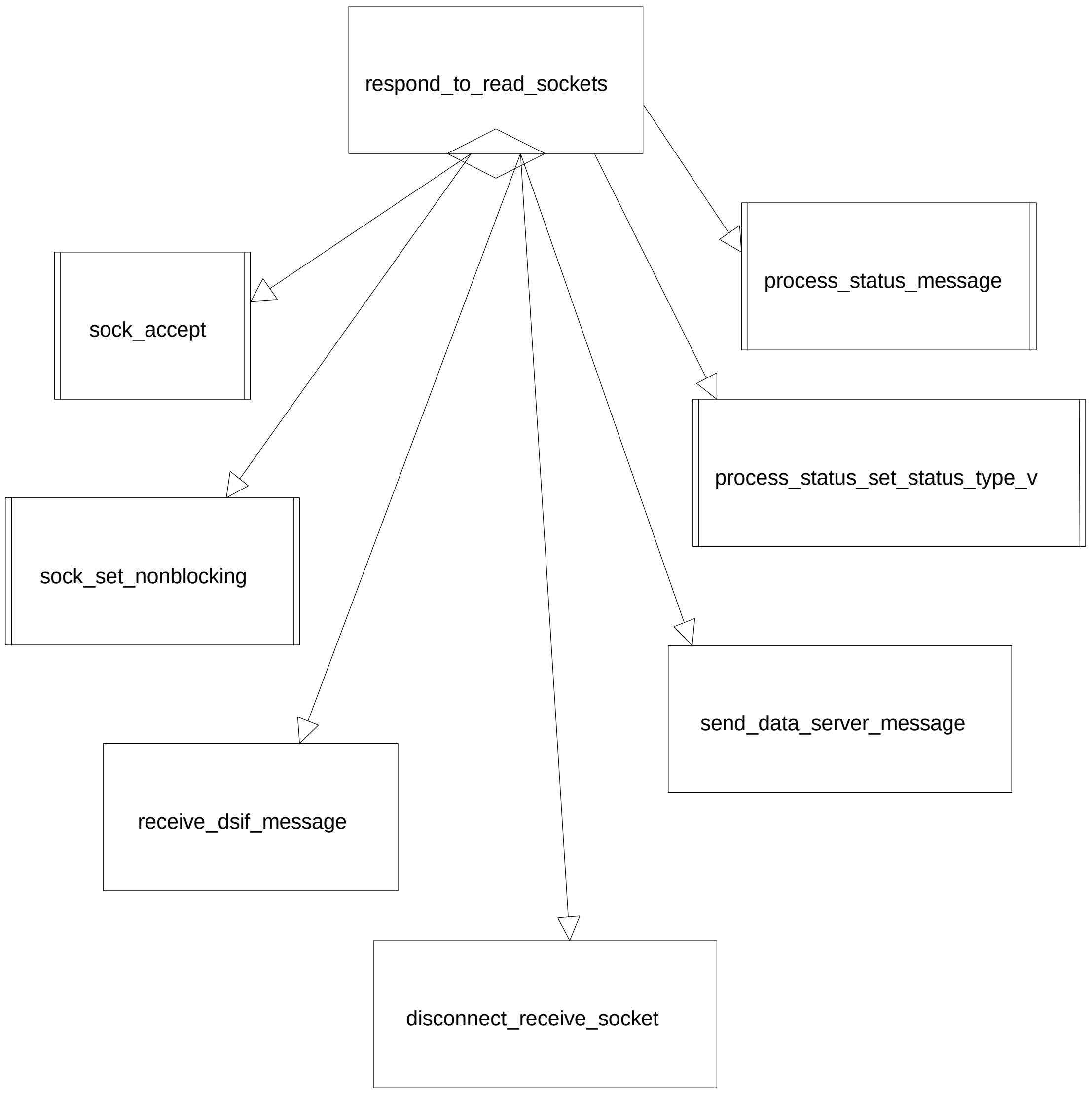


Figure 21. respond_to_read_sockets structure chart

Routine	Description
disconnect_receive_socket	Removes the specified socket descriptor from the specified file descriptor set and shuts down and closes the associated socket.
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_type_v	process_status_set_status_type_value is used to set the value associated with the specified process status status type.
receive_dsif_message	Reads a message from the specified socket. There is no attempt to clear the socket data or try to resynch the message data if any errors occur during reading.
respond_to_read_sockets	Loops through the list of socket descriptors ready for reading and either accepts connections, if the socket descriptor is for the listen socket, or receives messages containing information to be sent to the Data Server.
send_data_server_message	Extracts the contents of the message and sends the contents on to the Data Server. This could be a sensor message, a crossing message, or a heartbeat message.
sock_accept	MDI Socket routine that accepts connections on the specified listen socket.
sock_set_nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.

Table 37. Routines called by respond_to_read_sockets

5.3.1.11 receive_dsif_message

The receive_dsif_message routine reads the message from the active socket and places in the received message buffer.. The structure chart for receive_dsif_message is shown in Figure 22. The descriptions of the routines called by award_dsif_config_shm_mgr are contained in Table 38.

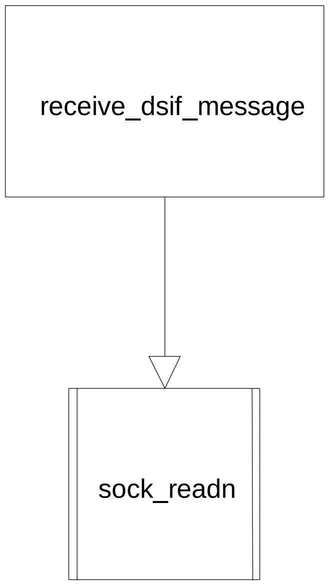


Figure 22. receive_dsif_message structure chart

Routine	Description
receive_dsif_message	Reads a message from the specified socket. There is no attempt to clear the socket data or try to resynch the message data if any errors occur during reading.
sock_readn	MDI Socket routine that reads a specified number of bytes from the specified socket.

Table 38. Routines called by receive_dsif_message

5.3.1.12 disconnect_receive_socket

The disconnect_receive_socket routine shuts down the active socket and removes the socket from the list of sockets the award_dsif process listens to for data. The structure chart for disconnect_receive_socket is shown in Figure 23. The descriptions of the routines called by award_dsif_setup_crossing_shmem are contained in Table 39.

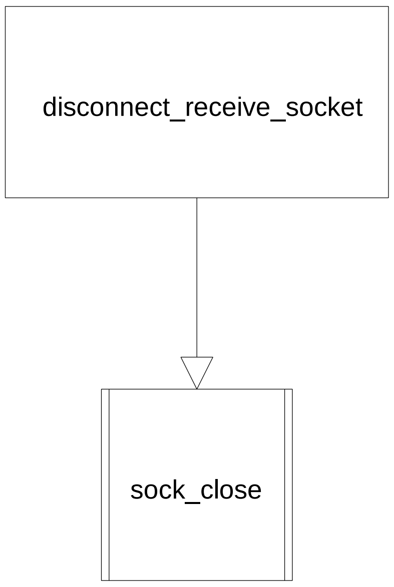


Figure 23. disconnect_receive_socket structure chart

Routine	Description
disconnect_receive_socket	Removes the specified socket descriptor from the specified file descriptor set and shuts down and closes the associated socket.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 39. Routines called by disconnect_receive_socket

5.3.1.13 send_data_server_message

The send_data_server_message routine takes the message that has been read from the active socket and breaks it apart for sending to the data server. The network to host byte-ordering of the data takes place here. The components of the message are used in the different data server library calls depending on the type of message that is received.. If the message is a sensor update then the sensor shared memory is modified and the contents sent to the data server. If the message is a crossing update then the sensor shared memory is modified and the contents sent to the data server. If the message is the heartbeat message then the status in the heartbeat message is passed to the data server. The structure chart for send_data_server_message is shown in Figure 24. The descriptions of the routines called by send_data_server_message are contained inTable 40. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

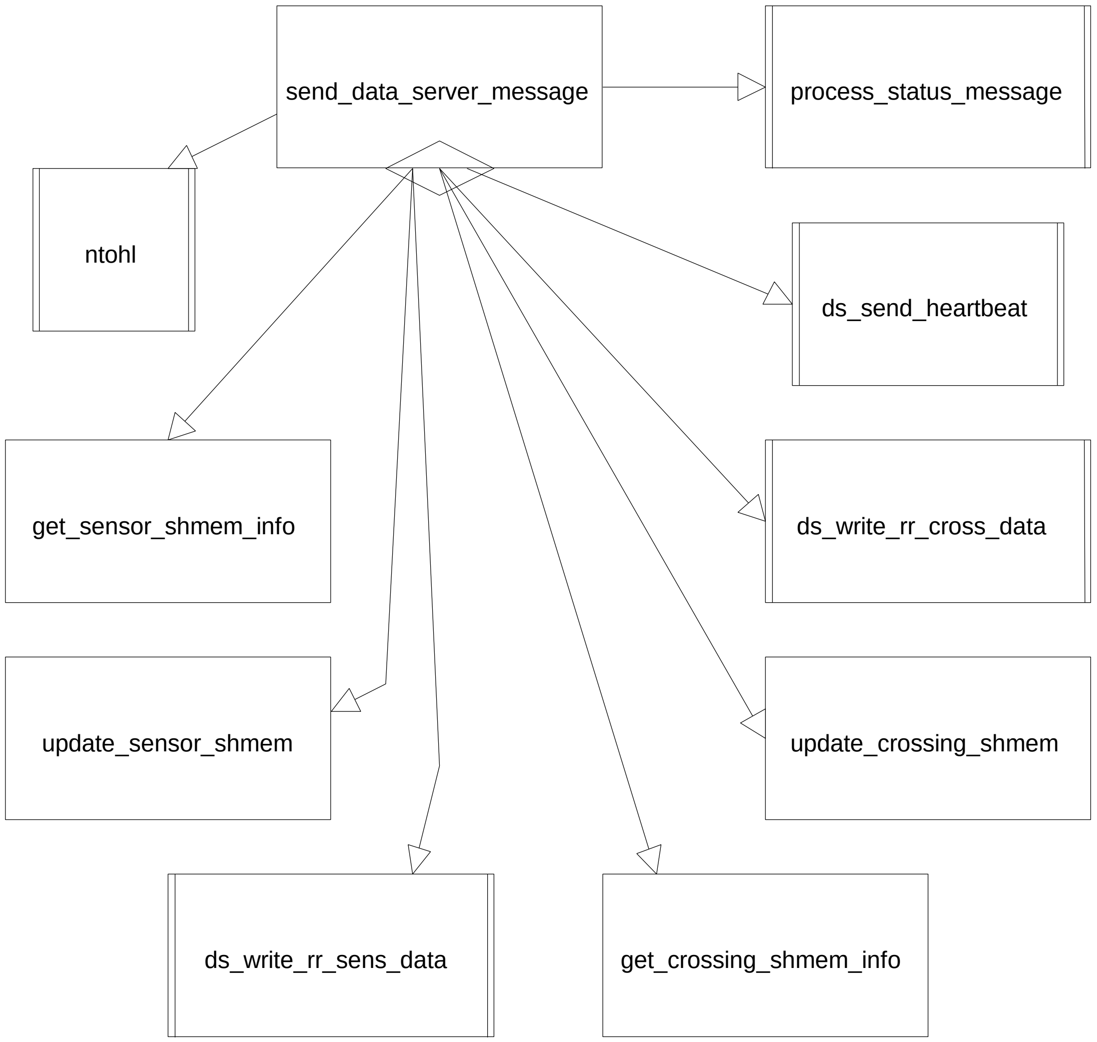


Figure 24. send_data_server_message structure chart

Routine	Description
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.
ds_write_rr_cross_data	MDI Data Server routine used to send the railroad crossing data to the Data Server.
ds_write_rr_sens_data	MDI Data Server routine used to send the railroad sensor data to the Data Server.
get_crossing_shmem_info	Used to obtain the address of the current crossing information and the number of crossings configured.
get_sensor_shmem_info	Used to obtain the address of the current sensor information and the number of sensors configured.
ntohl	Network Function used to convert between network and host byte order.
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_data_server_message	Extracts the contents of the message and sends the contents on to the Data Server. This could be a sensor message, a crossing message, or a heartbeat message.
update_crossing_shmem	Updates the information for the specified crossing in the shared memory segment. The updated information is available to any process attached to the crossing shared memory segment.
update_sensor_shmem	Updates the information for the specified sensor in the shared memory segment. The updated information is available to any process attached to the sensor shared memory segment.

Table 40. Routines called by send_data_server_message

5.3.1.13.1 get_sensor_shmem_info

The get_sensor_shmem_info routine returns the number of sensors currently configured and a pointer to the sensors shared memory segment. This routine is considered an access function and has no function calls. For that reason, no structure chart was produced for this routine.

5.3.1.13.2 update_sensor_shmem

The update_sensor_shmem routine takes the sensor message information and updates the shared memory element for only the sensor specified in the message. Since the shared memory elements are sorted, the bsearch routine is used to locate the sensor of interest The structure chart for update_sensor_shmem is shown in Figure 25. The descriptions of the routines called by update_sensor_shmem are contained inTable 41.

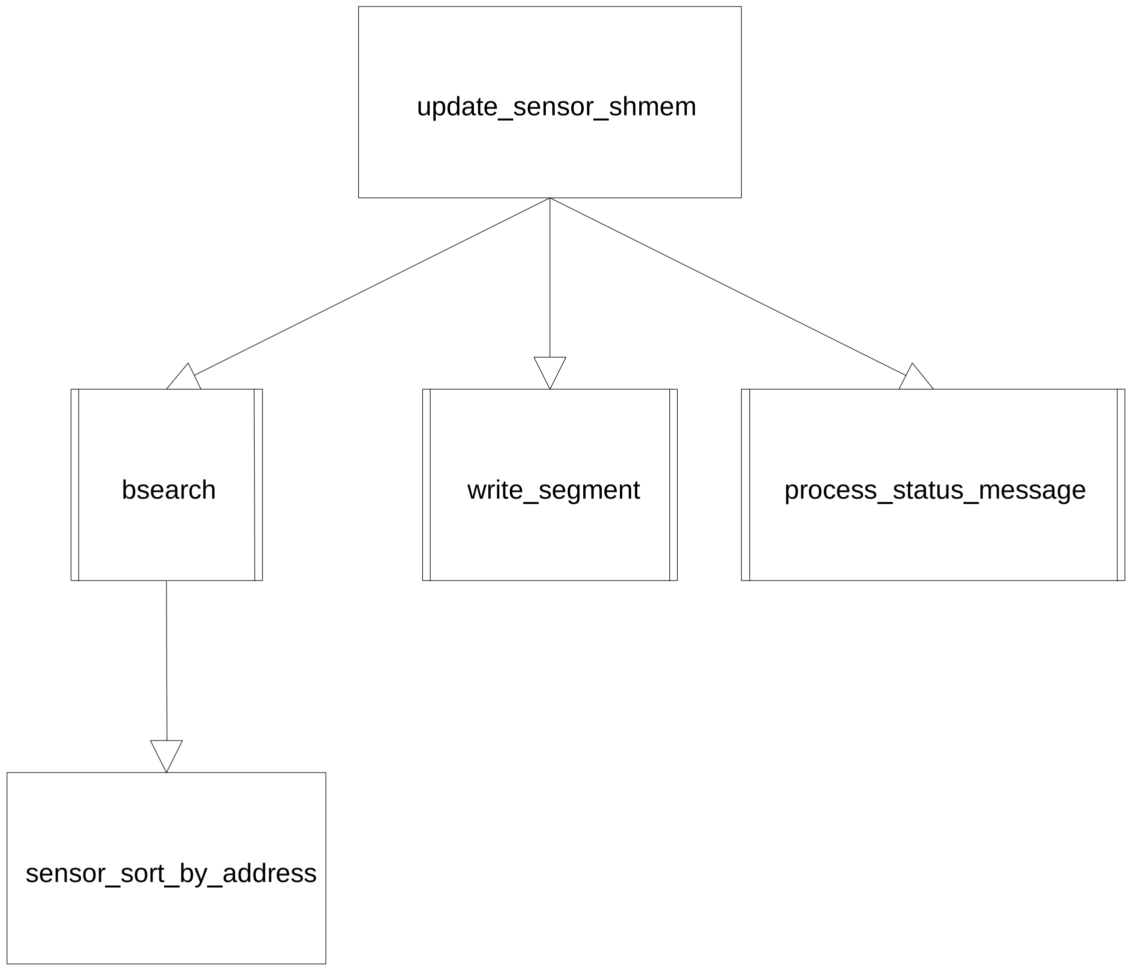


Figure 25. update_sensor_shmem structure chart

Routine	Description
bsearch	C Library Function implementing a binary search algorithm. A function is passed to this routine specifying the comparison routine to be used during the binary search. A pointer to the element found is returned or NULL if no element matching the search criteria is found.
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.
sensor_sort_by_address	Comparison routine used in the qsort call to sort the sensor ids in ascending order and used by the bsearch routine to find a match.
update_sensor_shmem	Updates the information for the specified sensor in the shared memory segment. The updated information is available to any process attached to the sensor shared memory segment.
write_segment	MDI Shared Memory Manager routine that writes data to the specified shared memory segment.

Table 41. Routines called by update_sensor_shmem

5.3.1.13.3 get_crossing_shmem_info

The get_crossing_shmem_info routine returns the number of crossings currently configured and a pointer to the crossings shared memory segment. This routine is considered an access function and has no function calls. For that reason, no structure chart was produced for this routine.

5.3.1.13.4 update_crossing_shmem

The update_crossing_shmem routine takes the crossing message information and updates the shared memory element for only the crossing specified in the message. Since the shared memory elements are sorted, the bsearch routine is used to locate the sensor of interest The structure chart for update_crossing_shmem is shown in Figure 26. The descriptions of the routines called by update_crossing_shmem are contained inTable 42.

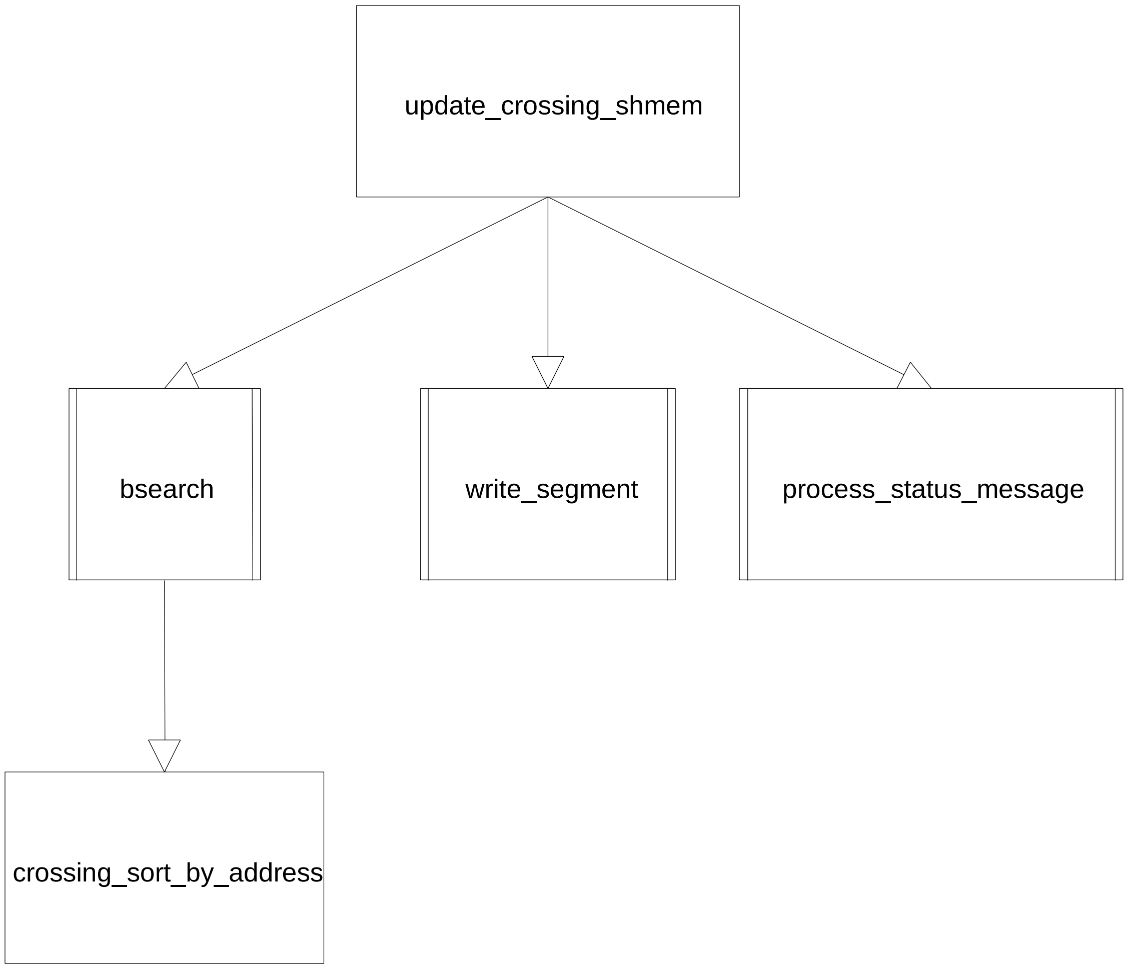


Figure 26. update_crossing_shmem structure chart

Routine	Description
bsearch	C Library Function implementing a binary search algorithm. A function is passed to this routine specifying the comparison routine to be used during the binary search. A pointer to the element found is returned or NULL if no element matching the search criteria is found.
crossing_sort_by_address	Comparison routine used in the qsort call to sort the crossing ids in ascending order and used by the bsearch routine to find a match.
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_crossing_shmem	Updates the information for the specified crossing in the shared memory segment. The updated information is available to any process attached to the crossing shared memory segment.
write_segment	MDI Shared Memory Manager routine that writes data to the specified shared memory segment.

Table 42. Routines called by update_crossing_shmem

5.3.1.14 sigalrm_handler

The sigalrm_handler routine is invoked whenever the award_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. The structure chart for the sigalrm_handler is shown inFigure 27. A description of the routines called by sigalrm_handler is provided in Table 43.

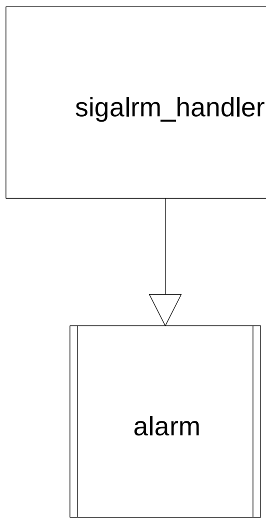


Figure 27. sigalrm_handler structure chart

Routine	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	The signal handler for the SIGALRM signal. This signal is used to indicate when the process-level heartbeat should be sent to the AWARD subsystem heartbeat process. The alarm is reinitialized as part of this routine.

Table 43. Routines called by sigalrm_handler

5.3.2 Dispatch Crossing Delays (award_tgif)

The award_tgif process provides the single point of interface between the AWARD subsystem and the TransGuide ATMS. award_tgif is responsible for receiving messages from the other processes in the AWARD subsystem and directing these messages to the TransGuide ATMS.

5.3.2.1 main

The structure chart for the main routine is shown inFigure 28. The 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, connecting to the heartbeat process and the external alarm handler on the TransGuide ATMS, sending periodic heartbeats to the project-level heartbeat process, and responding to requests made by the other processes within the AWARD subsystem. A description of the routines called by the main routine of award_tgif is provided inTable 44.

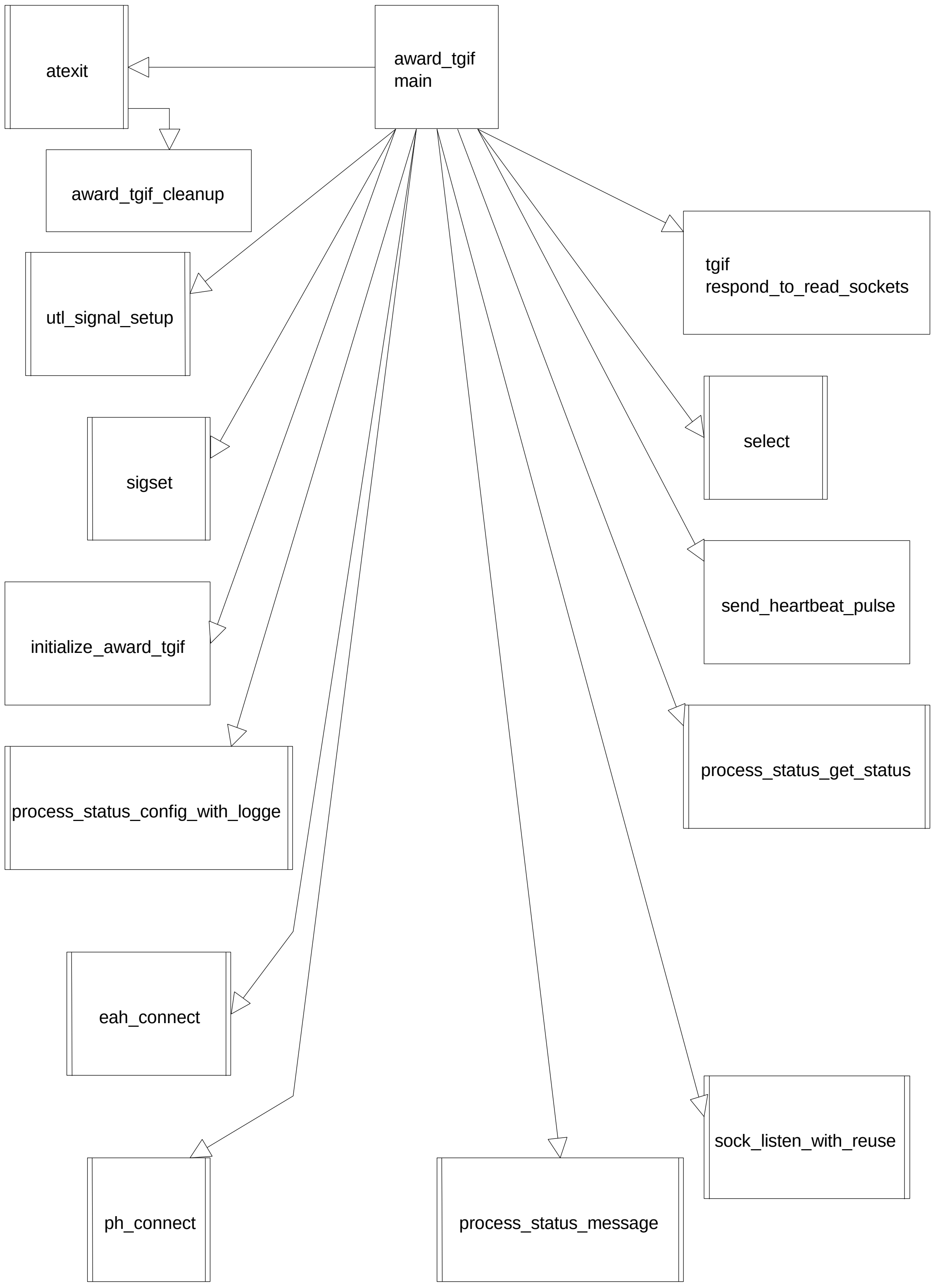


Figure 28. award_tgif main structure chart

Routine	Description
atexit	C Library Function used to register routines to be called on normal termination of a program.
award_tgif main	The award_tgif main routine is responsible for setting up configuration information, opening the socket used for communication, and connecting to the status logger. This routine enters a loop waiting for TransGuide messages and periodically sending heartbeat messages to the subsystem heartbeat process.
award_tgif_cleanup	Called when award_tgif exits. This routine is responsible for performing the housekeeping necessary for a graceful shutdown. This includes sending a last heartbeat, disconnecting from the process-level heartbeat service, disconnecting from the TransGuide external alarm handler, and closing any sockets that are open for communicating with the award_tgif process.
eah_connect	MDI External Alarm Handler routine used to connect to the specified ATMS external alarm handler service. The host name and service name are used to make the connection.
initialize_award_tgif	The award_tgif configuration file specified on the command line is read to obtain the values of the configurable items of the award_tgif process.
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_logge	process_status_config_with_logger is an MDI Process Status Common routine used to configure the process status handling for the process. This routine is used to set 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.
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 process-level heartbeat to the Subsystem Heartbeat 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.
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.
tgif respond_to_read_sockets	Loops through the list of socket descriptors ready for reading and either accepts connections, if the socket descriptor is for the listen socket, or receives messages containing information to be sent to the external alarm handler.
utl_signal_setup	MDI Common Utility Library routine used to set up a default signal handler for all catchable signals.

Table 44. Routines called by award_tgif main

5.3.2.2 award_tgif_cleanup

The award_tgif_cleanup routine is called when the award_tgif process performs a normal termination. This routine performs the necessary housekeeping chores to cause a graceful exit of the award_tgif process. The structure chart for the award_tgif_cleanup routine is shown in Figure 29. A description of the routines called by award_tgif_cleanup is provided in Table 45.

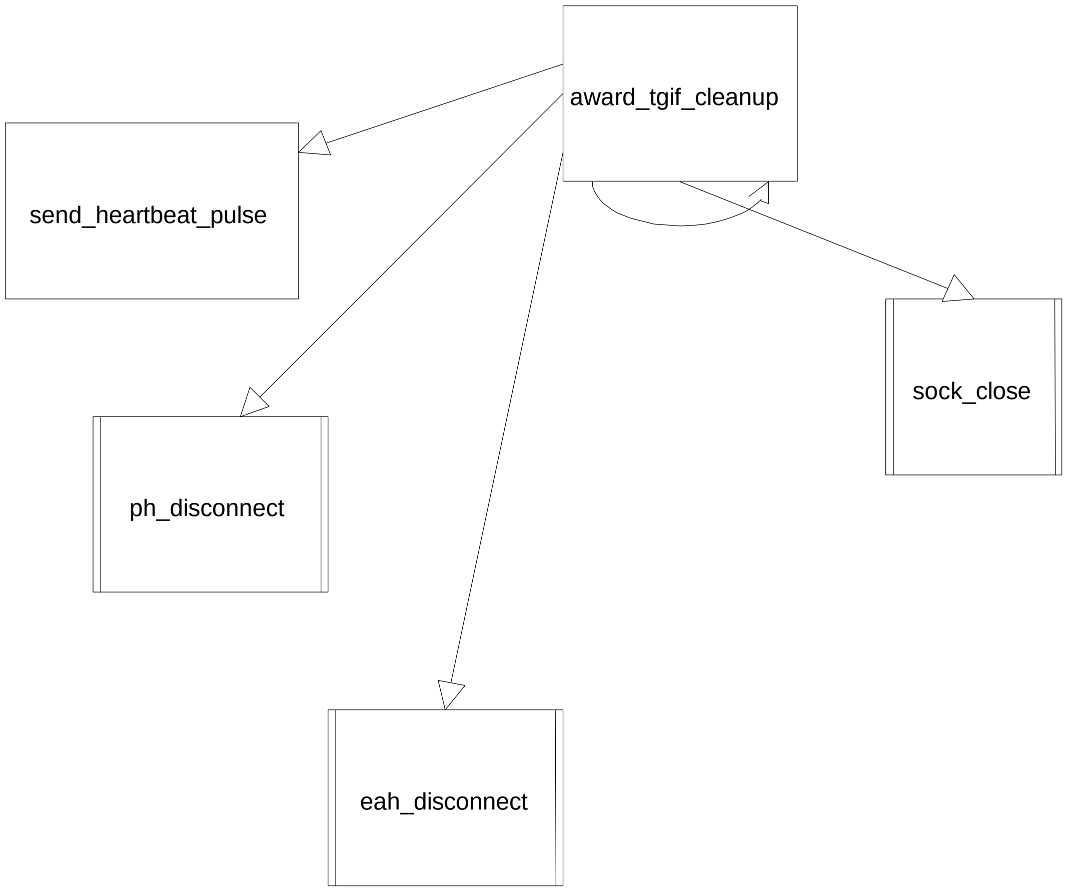


Figure 29. award_tgif_cleanup structure chart

Routine	Description
award_tgif_cleanup	Called when award_tgif exits. This routine is responsible for performing the housekeeping necessary for a graceful shutdown. This includes sending a last heartbeat, disconnecting from the process-level heartbeat service, disconnecting from the TransGuide external alarm handler, and closing any sockets that are open for communicating with the award_tgif process.
eah_disconnect	MDI External Alarm Handler routine used to disconnect from the external alarm handler service.
ph_disconnect	MDI Process Heartbeat routine used to disconnect from the process-level heartbeat service.
send_heartbeat_pulse	Sends the process-level heartbeat to the Subsystem Heartbeat process.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 45. Routines called by award_tgif_cleanup

5.3.2.3 send_heartbeat_pulse

The send_heartbeat_pulse routine is invoked periodically whenever the socket selection is interrupted by a timeout. 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 inFigure 30. The descriptions of the routines called by send_heartbeat_pulse are contained inTable 46.

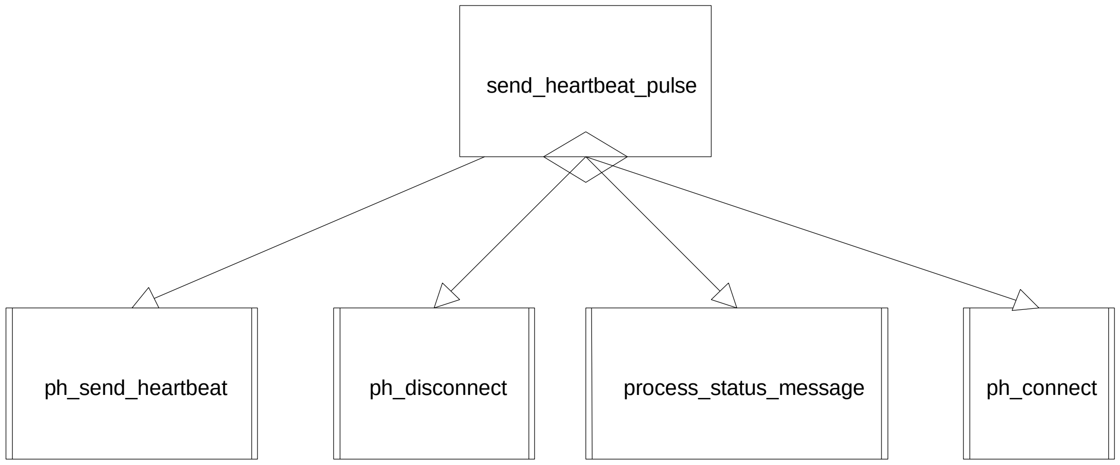


Figure 30. send_heartbeat_pulse structure chart

Routine	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 process-level heartbeat to the Subsystem Heartbeat process.

Table 46. Routines called by send_heartbeat_pulse

5.3.2.4 initialize_award_tgif

The initialize_award_tgif routine is called to read the award_tgif configuration file and set up configuration information for the entire process. The structure chart for initialize_award_tgif is shown in Figure 31. Descriptions of the routines called by initialize_award_tgif are contained inTable 47. Configurable items for the award_tgif process are described inTable 48.

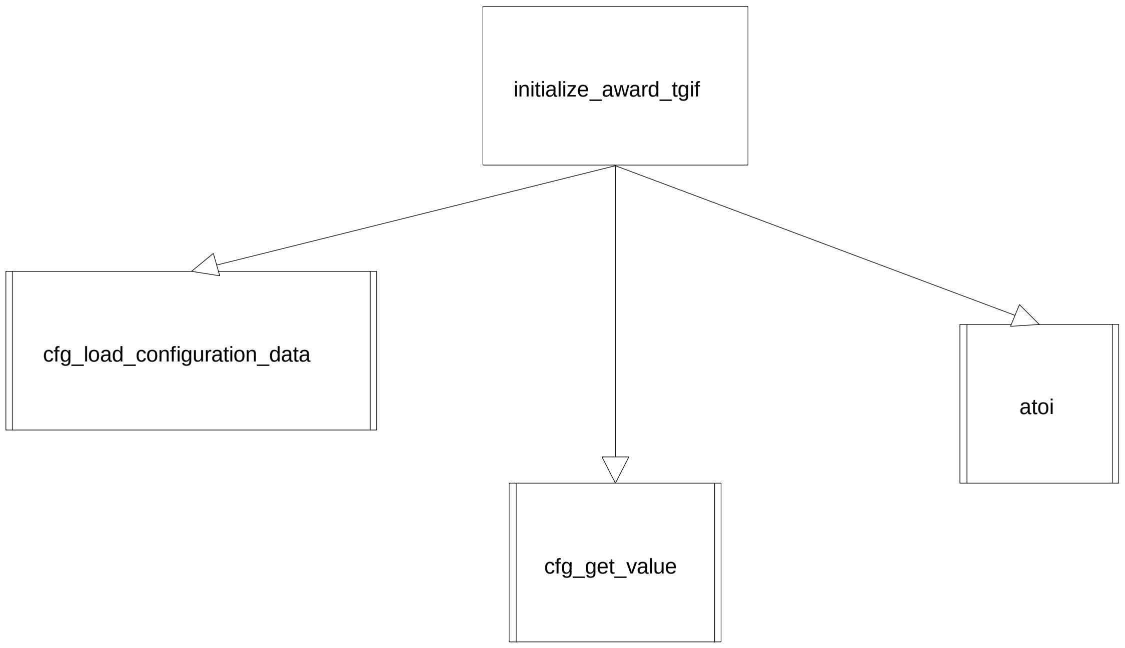


Figure 31. initialize_award_tgif structure chart

Routine	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.
initialize_award_tgif	The award_tgif configuration file specified on the command line is read to obtain the values of the configurable items of the award_tgif process.

Table 47. Routines called by initialize_award_tgif

Configuration Item	Description	Optional
SERVICE_NAME	The name of the service providied by the award_tgif process.	N
HEARTBEAT_SERVICE_NAME	The name of the service provided by the AWARD project-level heartbeat process.	N
HEARTBEAT_HOST_NAME	The host name where the AWARD project-level heartbeat process resides.	Y
STATUS_LOGGER_SERVICE_NAME	The name of the service provided by the AWARD subsystem status logger process.	N
STATUS_LOGGER_HOST_NAME	The host name where the AWARD subsystem status logger process resides	Y
HEARTBEAT_PULSE	The periodic time value for sending the heartbeat to the AWRAD project-level heartbeat process. This is specified in seconds.	Y
EAH_SERVICE_NAME	The name of the service provided by the external alarm handler process.	N
EAH_HOST_NAME	The host name where the external alarm handler process resides.	Y

Table 48. award_tgif configuration items

5.3.2.5 respond_to_read_sockets

The respond_to_read_sockets routine is heart of the award_tgif 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 award_tgif process, a message being sent to the award_tgif process by another process already connected, or it could be an indication of a process that has disconnected from the award_tgif 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 external alarm handler. If a connected process disconnects from the award_tgif process the socket connection from the award_tgif process to the disconnected process is closed and removed from the list of active sockets. Errors that occur are logged to the AWARD subsystem status log. The structure chart for the respond_to_read_sockets is shown in Figure 32. A description of the routines called by respond_to_read_sockets is provided inTable 49.

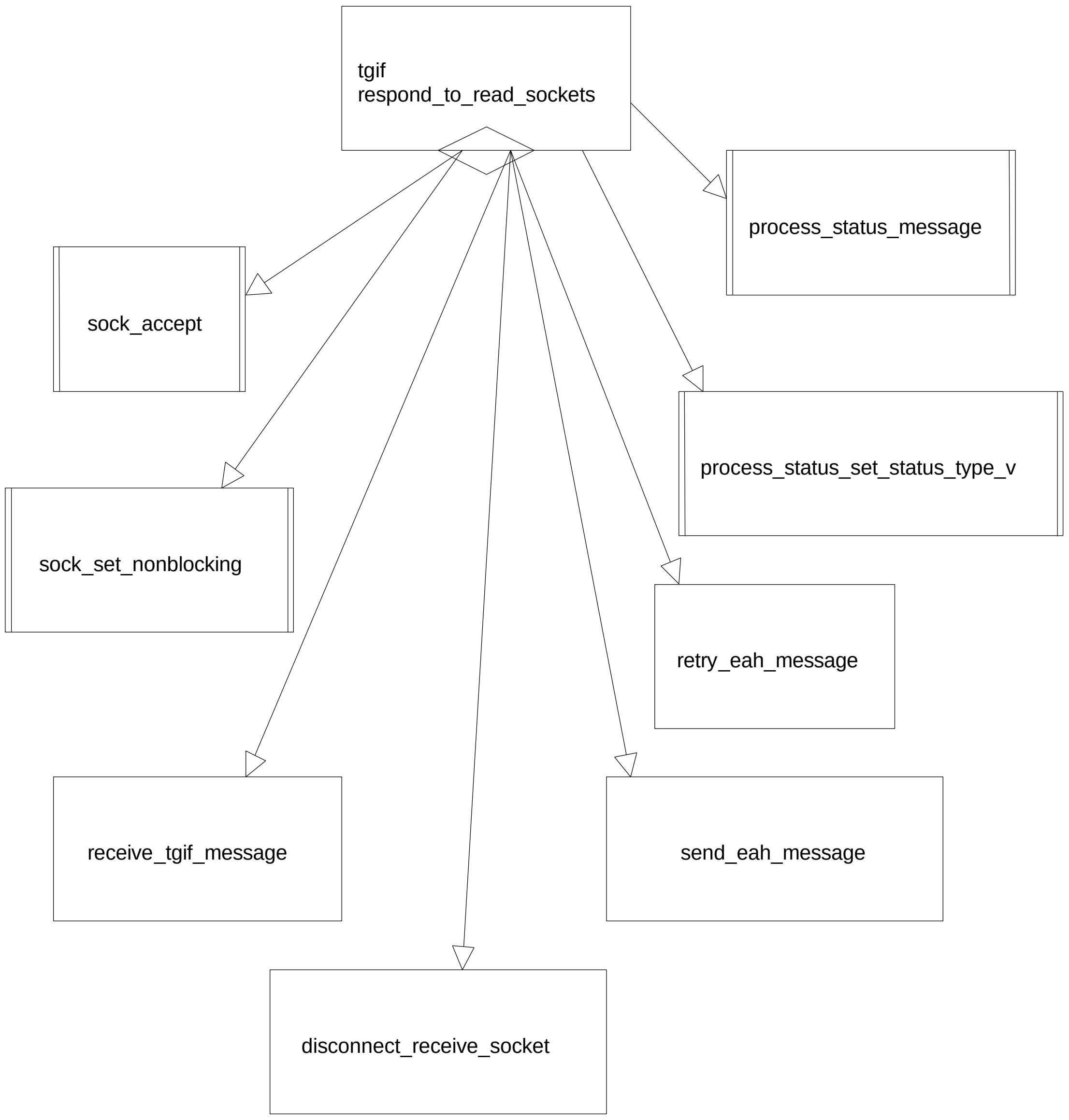


Figure 32. respond_to_read_sockets structure chart

Routine	Description
disconnect_receive_socket	Removes the specified socket descriptor from the specified file descriptor set and shuts down and closes the associated socket.
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_type_v	process_status_set_status_type_value is used to set the value associated with the specified process status status type.
receive_tgif_message	Reads a message from the specified socket. There is no attempt to clear the socket data or try to resynch the message data if any errors occur during reading.
retry_eah_message	Attempts to reestablish communications with the external alarm handler process running on the ATMS master computer. The specified message is sent to the external alarm handler if communications are reestablished.
send_eah_message	Extracts the contents of the message and sends the contents on to the External Alarm Handler. The message contains the data needed to create the crossing delay alarm.
sock_accept	MDI Socket routine that accepts connections on the specified listen socket.
sock_set_nonblocking	MDI Socket routine that sets the specified socket to be a non-blocking socket.
tgif respond_to_read_sockets	Loops through the list of socket descriptors ready for reading and either accepts connections, if the socket descriptor is for the listen socket, or receives messages containing information to be sent to the external alarm handler.

Table 49. Routines called by respond_to_read_sockets

5.3.2.6 receive_tgif_message

The receive_tgif_message routine reads the message from the active socket and places in the received message buffer. The structure chart for receive_tgif_message is shown in Figure 33. The descriptions of the routines called by receive_tgif_message are contained in Table 50.

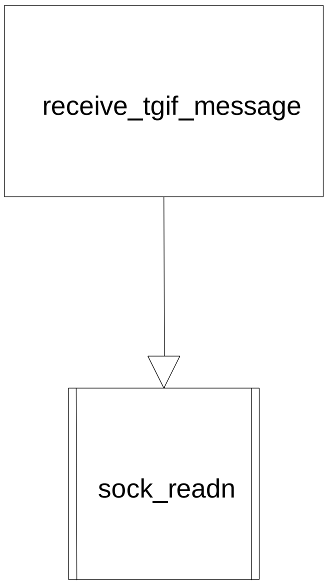


Figure 33. receive_tgif_message structure chart

Routine	Description
receive_tgif_message	Reads a message from the specified socket. There is no attempt to clear the socket data or try to resynch the message data if any errors occur during reading.
sock_readn	MDI Socket routine that reads a specified number of bytes from the specified socket.

5.3.2.7 disconnect_receive_socket

The disconnect_receive_socket routine shuts down the active socket and removes the socket from the list of sockets the award_tgif process listens to for data. The structure chart for disconnect_receive_socket is shown in Figure 34. The descriptions of the routines called by disconnect_receive_socket are contained in Table 51.

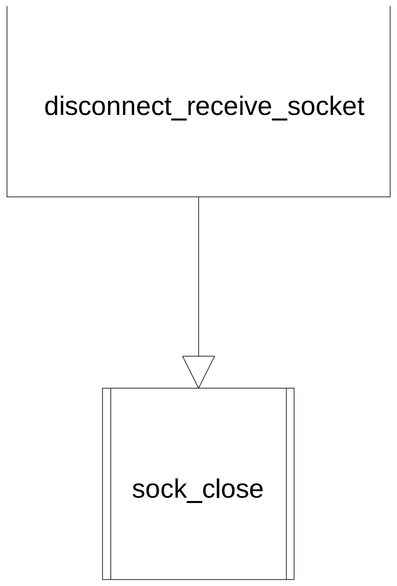


Figure 34. disconnect_receive_socket structure chart

Routine	Description
disconnect_receive_socket	Removes the specified socket descriptor from the specified file descriptor set and shuts down and closes the associated socket.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 51. Routines called by disconnect_receive_socket

5.3.2.8 send_eah_message

The send_eah_message routine takes the message that has been read from the active socket and breaks it apart for sending to the external alarm handler. The network to host byte-ordering of the data takes place here. The components of the message are used in the external alarm handler library call The structure chart for send_eah_message is shown in Figure 35. The descriptions of the routines called by send_eah_message are contained in Table 52. Any errors that occur during this routine are logged to the AWARD status log using the process_status_message routine.

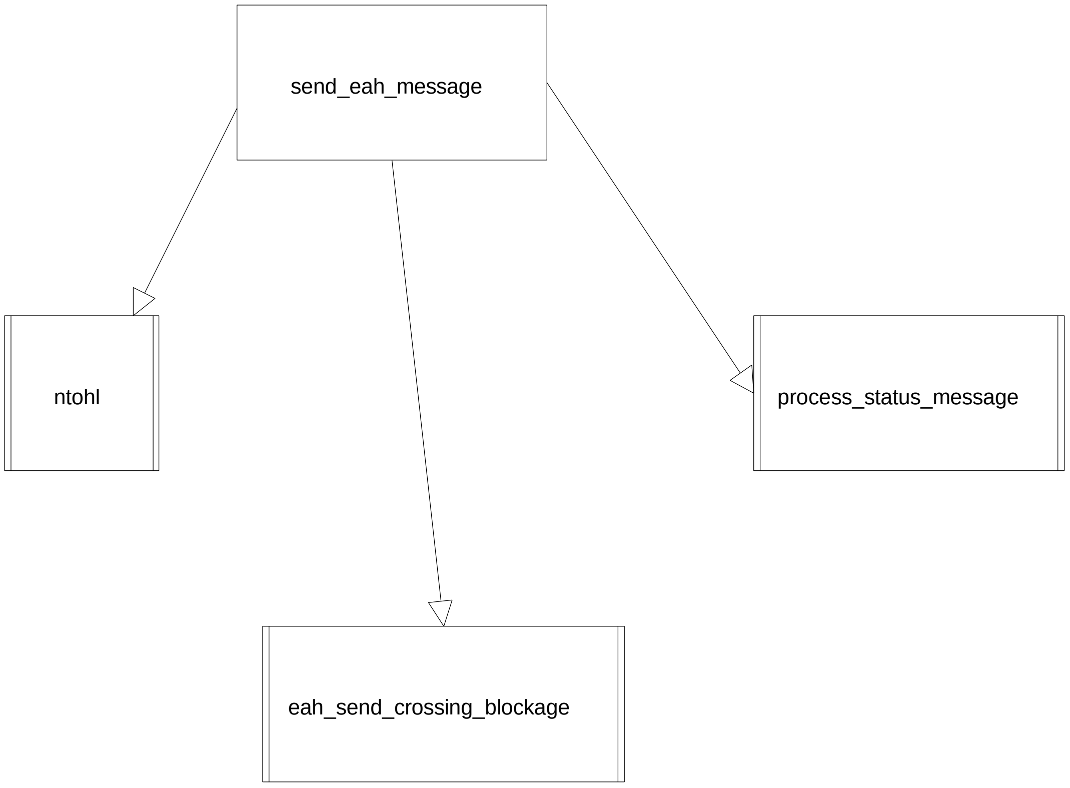


Figure 35. send_eah_message structure chart

Routine	Description
eah_send_crossing_blockage	Packages the specified crossing delay data into the crossing delay alarm message and sends it to the External Alarm Handler process.
ntohl	Network Function used to convert between network and host byte order.
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_eah_message	Extracts the contents of the message and sends the contents on to the External Alarm Handler. The message contains the data needed to create the crossing delay alarm.

Table 52. Routines called by send_eah_message

5.3.2.9 retry_eah_message

The retry_eah_message routine is called in the event an error occurs during the first send to the external alarm handler. This routine disconnects from the external alarm handler in order to reset the connection information, reconnects to the external alarm handler, and attempts to send the message again. The structure chart for retry_eah_message is shown in Figure 36. The descriptions of the routines called by retry_eah_message are contained in Table 53.

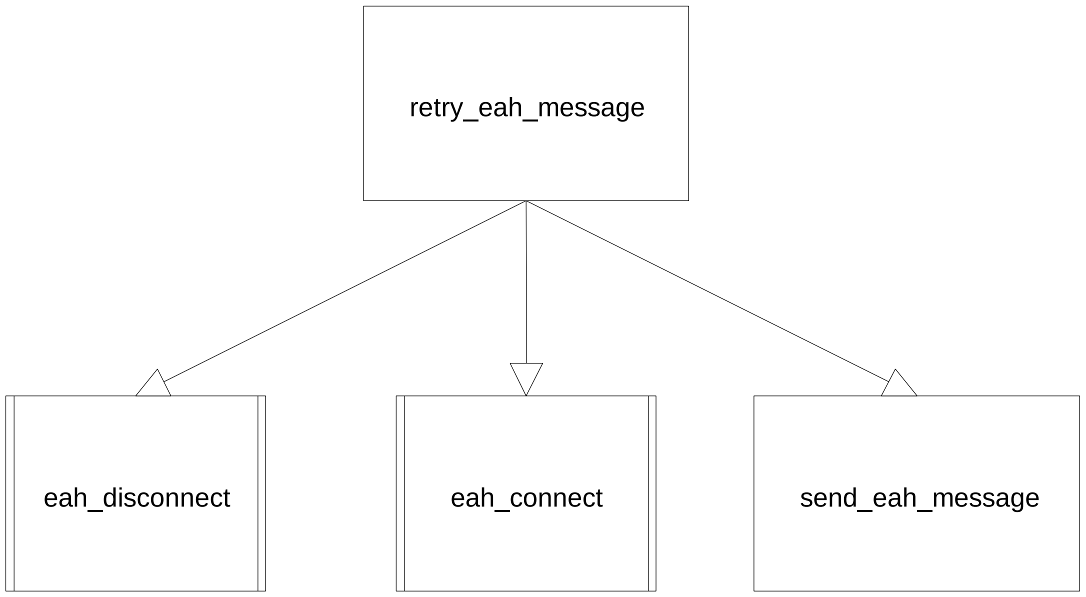


Figure 36. retry_eah_message structure chart

Routine	Description
eah_connect	MDI External Alarm Handler routine used to connect to the specified ATMS external alarm handler service. The host name and service name are used to make the connection.
eah_disconnect	MDI External Alarm Handler routine used to disconnect from the external alarm handler service.
retry_eah_message	Attempts to reestablish communications with the external alarm handler process running on the ATMS master computer. The specified message is sent to the external alarm handler if communications are reestablished.
send_eah_message	Extracts the contents of the message and sends the contents on to the External Alarm Handler. The message contains the data needed to create the crossing delay alarm.

Table 53. Routines called by retry_eah_message

5.3.3 Show Detailed Status (awdsg)

The AWARD detailed status GUI (awdsg) process provides the ability to visually inspect the current status and values associated with each of the sensors and crossings configured within AWARD. awdsg runs as a separate process from the rest of the AWARD subsystem. When awdsg runs it attempts to attach itself to the sensor and crossing shared memory segments. Then on a periodic basis it reads the information from these shared memory segments and displays on the workstation for the user.

5.3.3.1 TeleUSE_main

The TeleUSE model requires a TeleUSE provided main routine to initially gain control of the process. This main routine does numerous things that are not modeled here with the exception of invoking the application main routine and then firing the INITIALLY rules that exist in the different D modules. The structure chart for the TeleUSE_main routine is shown in Figure 37. This is a theoretical represenation of the actual source code since this routine was not developed by SwRI. A description of the routines called by TeleUSE_main is provided in Table 54.

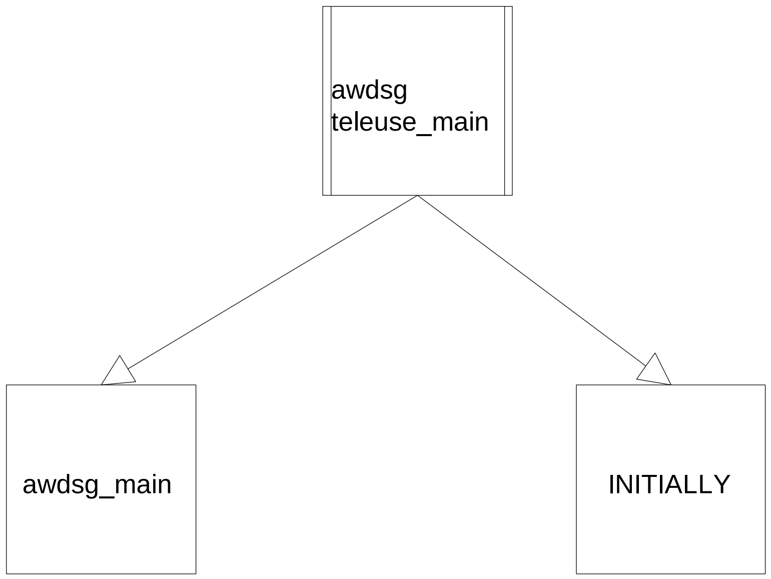


Figure 37. awdsg teleuse_main structure chart

Routine	Description
awdsg teleuse_main	This is the main routine of the AWARD Detailed Status GUI. This routine is supplied by the TeleUSE UIMS tool and is used as the entry point into the process. This routine is responsible for setting up any TeleUSE specific environment and then invoking the application main routine followed by the INITIALLY events in the associated D modules.
awdsg_main	This is the main routine of the AWARD Detailed Status GUI. This routine is responsible for loading the configuration information, configuring the shared memory manager library, and attaching to the field equipment shared memory segments.
INITIALLY	This D event is the first event invoked by the TeleUSE runtime environment. This event creates the top-level shell to contain the detailed status information, sets the update rate for the GUI, and then starts the update process by triggering the periodic_update event. Any errors during this event will cause tu_exit to be called to start a graceful shutdown of the process.

Table 54. Routines called by awdsg teleuse_main

5.3.3.2 awdsg_main

The awdsg_main routine is called prior to any GUI setup being done. This routine is used to initialize the application prior to performing any user interface functions. This routine loads the configuration data, configures the shared memory manager, and attaches to the shared memory segment.. The structure chart for the awdsg_main routine is shown inFigure 38. A description of the routines called by awdsg_main is provided in Table 55. The only configuration item for this program is the UPDATE_RATE. This item is optional, but if specified it indicates how often the details will be updated.

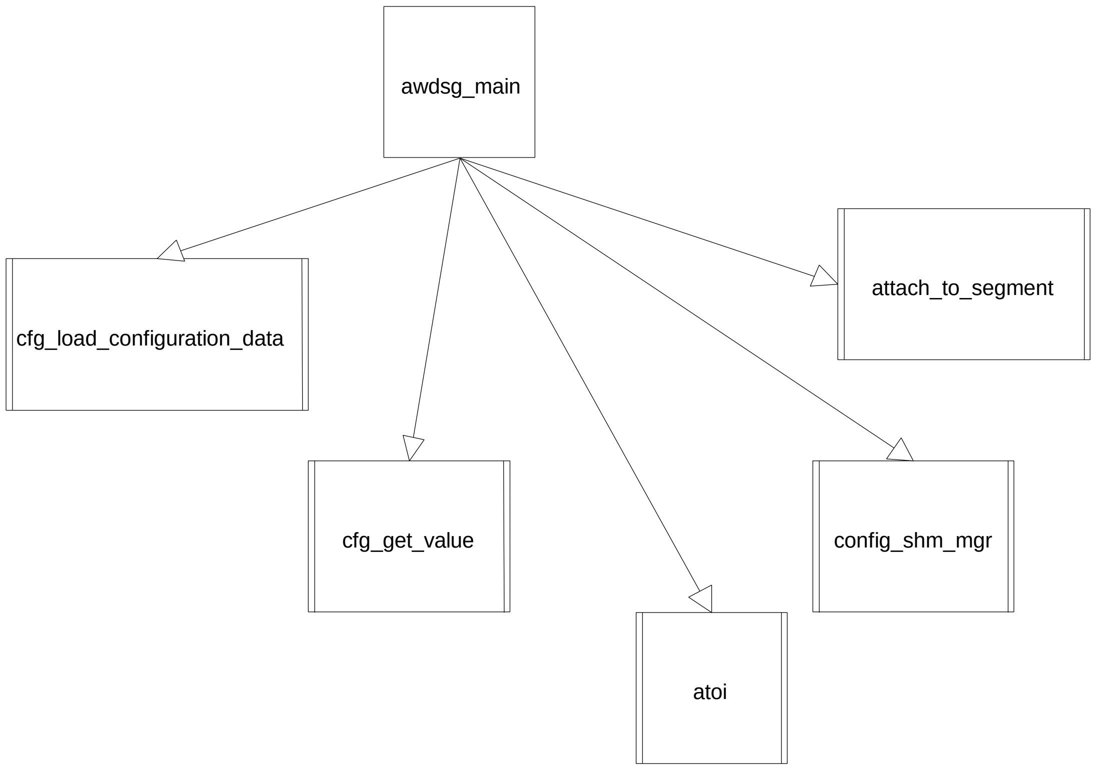


Figure 38. awdsg_main structure chart

Routine	Description
atoi	C Library Function to convert an ASCII string to an integer value.
attach_to_segment	MDI Shared Memory Manager routine used to attach the calling process to the specified shared memory segment.
awdsg_main	This is the main routine of the AWARD Detailed Status GUI. This routine is responsible for loading the configuration information, configuring the shared memory manager library, and attaching to the field equipment shared memory segments.
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.
config_shm_mgr	MDI Shared Memory Manager routine used to initialize and configure the shared memory manager library routines for the calling program.

Table 55. Routines called by awdsg_main

5.3.3.3 INITIALLY

INITIALLY is the first rule that gets triggered in any D module. For the detailed status GUI the INITIALLY rule creates the top level shell widget, obtains the current update rate defined for the application, and starts the periodic updates. The structure chart for INITIALLY in Figure 39. Descriptions of the routines called by INITIALLY are contained inTable 56.

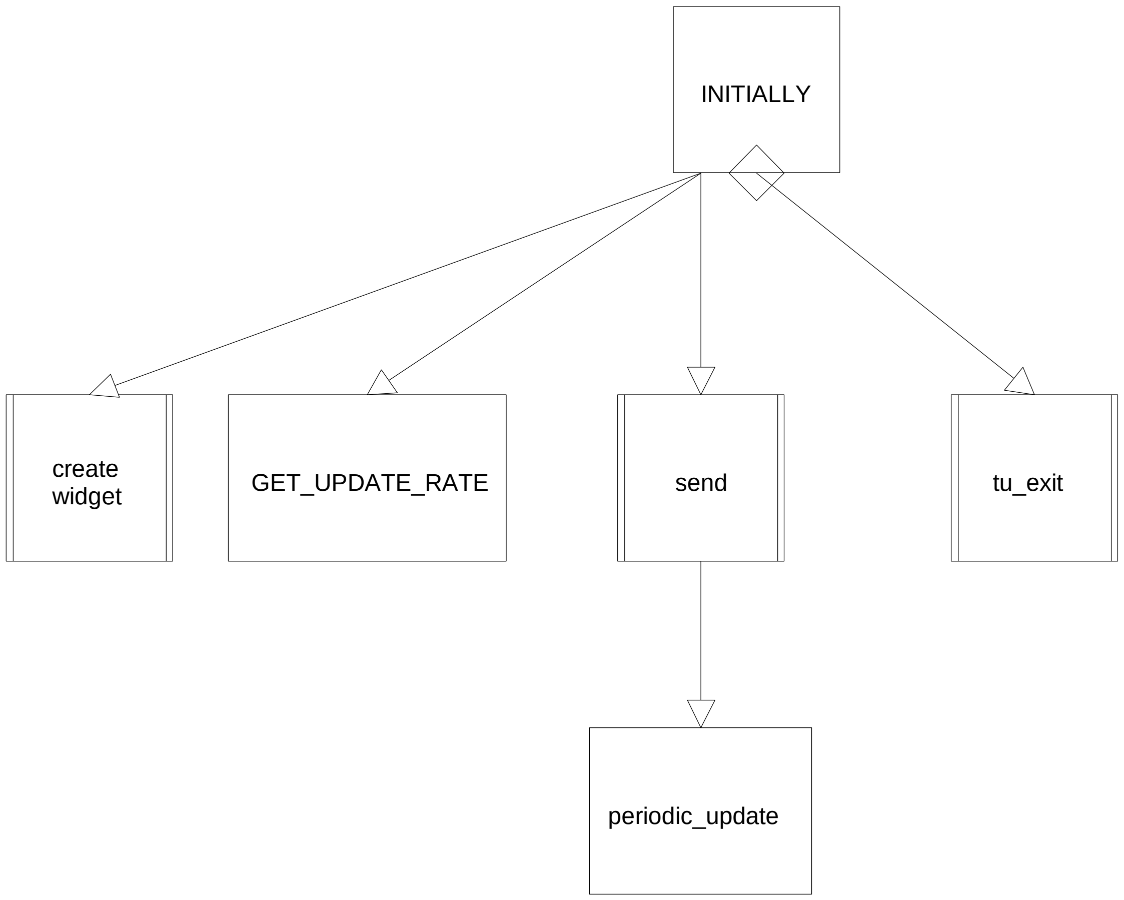


Figure 39. INITIALLY structure chart

Routine	Description
create widget	create widget is used to create a widget of a particular TeleUSE template allowing for the specification of a widget name and a parent for the widget.
GET_UPDATE_RATE	A bridge layer routine used to obtain the update rate value from the application layer.
INITIALLY	This D event is the first event invoked by the TeleUSE runtime environment. This event creates the top-level shell to contain the detailed status information, sets the update rate for the GUI, and then starts the update process by triggering the periodic_update event. Any errors during this event will cause tu_exit to be called to start a graceful shutdown of the process.
periodic_update	An GUI layer event used to perform the steps necesary to update the details of the GUI on a periodic basis.
send	A TeleUSE statement used to trigger events immediately or queue events for later dispatch.
tu_exit	A TeleUSE library routine used to exit the application.

Table 56. Routines called by INITIALLY

5.3.3.3.1 GET_UPDATE_RATE

The GET_UPDATE_RATE routine is a bridge layer routine that calls the application layer routine to obtain the configured update rate The structure chart for GET_UPDATE_RATE is shown inFigure 40. The descriptions of the routines called by GET_UPDATE_RATE are contained inTable 57.

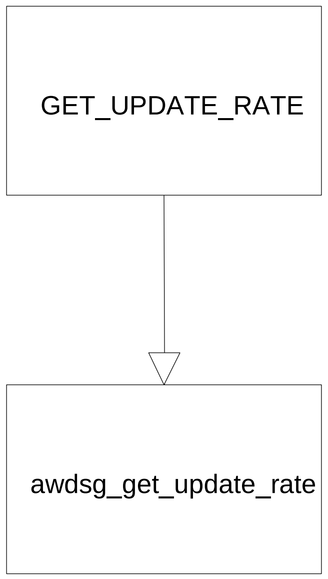


Figure 40. GET_UPDATE_RATE structure chart

Routine	Description
awdsg_get_update_rate	The application layer routine responsible for returning the configured update rate for the detailed status GUI.
GET_UPDATE_RATE	A bridge layer routine used to obtain the update rate value from the application layer.

Table 57. Routines called by GET_UPDATE_RATE

5.3.3.3.1.1 periodic_update

The periodic_update event is invoked to control the updates to the detailed status GUI and to cause the next triggering of periodic_update to occurr.. The structure chart for periodic_update is shown in Figure 41. The descriptions of the routines and events called by periodic_update are contained in Table 58.

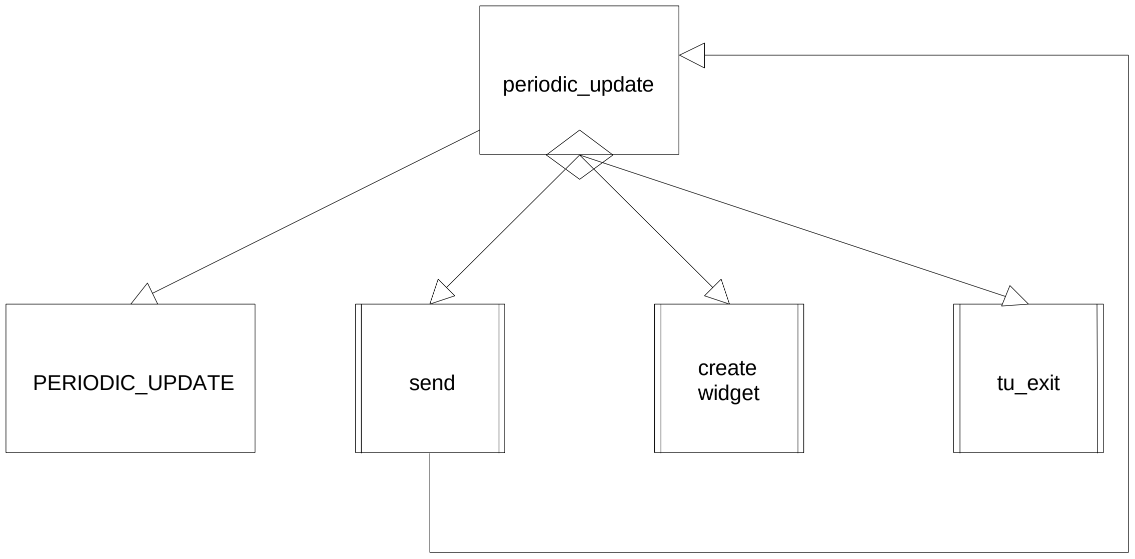


Figure 41. periodic_update structure chart

Routine	Description
create widget	create widget is used to create a widget of a particular TeleUSE template allowing for the specification of a widget name and a parent for the widget.
PERIODIC_UPDATE	The bridge layer routine that invokes the application layer routine responsible for handling the periodic update requests.
periodic_update	An GUI layer event used to perform the steps necesary to update the details of the GUI on a periodic basis.
tu_exit	A TeleUSE library routine used to exit the application.

Table 58. Routines called by periodic_update

5.3.3.3.1.1.1 PERIODIC_UPDATE

The PERIODIC_UPDATE routine is the bridge layer routine responsible for invoking the application layer routine which provides the update capability. The structure chart for the PERIODIC_UPDATE is shown in Figure 42. A description of the routines called by PERIODIC_UPDATE is provided in Table 59.

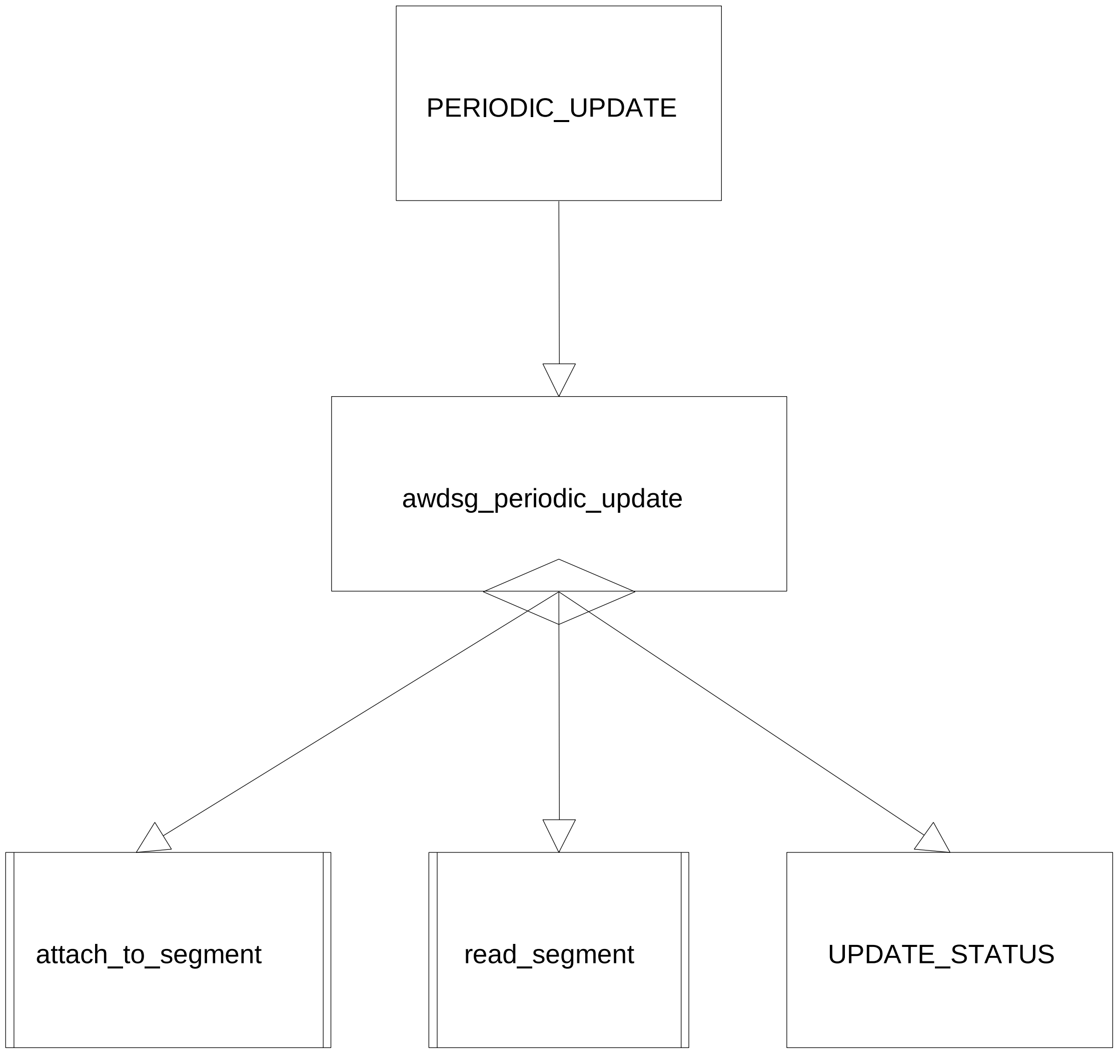


Figure 42. PERIODIC_UPDATE structure chart

Routine	Description
attach_to_segment	MDI Shared Memory Manager routine used to attach the calling process to the specified shared memory segment.
awdsg_periodic_update	The application layer routine responsible for handling the periodic update requests. This routine attaches to the sensor and crossing shared memory segments if not attached, reads the data from these segments, and, using the bridge layer, causes the contents of the GUI to be updated based on the contents of the shared memory segments.
PERIODIC_UPDATE	The bridge layer routine that invokes the application layer routine responsible for handling the periodic update requests.
read_segment	MDI Shared Memory Manager routine to read the contents of the specified shared memory segment. The contents are stored in a memory area allocated by the caller.
UPDATE_STATUS	This is the bridge layer routine that receives the current status information for a particular sensor or crossing and then generates the event to update the status information within the detailed status GUI.

Table 59. Routines called by PERIODIC_UPDATE

5.3.3.3.1.1.1.1 UPDATE_STATUS

The UPDATE_STATUS takes the detailed status information and creates the necessary event structures in order to inform the GUI layer to update the appropriate GUI components. The structure chart for UPDATE_STATUS is shown in Figure 43. The descriptions of the routines called by UPDATE_STATUS are contained in Table 60.

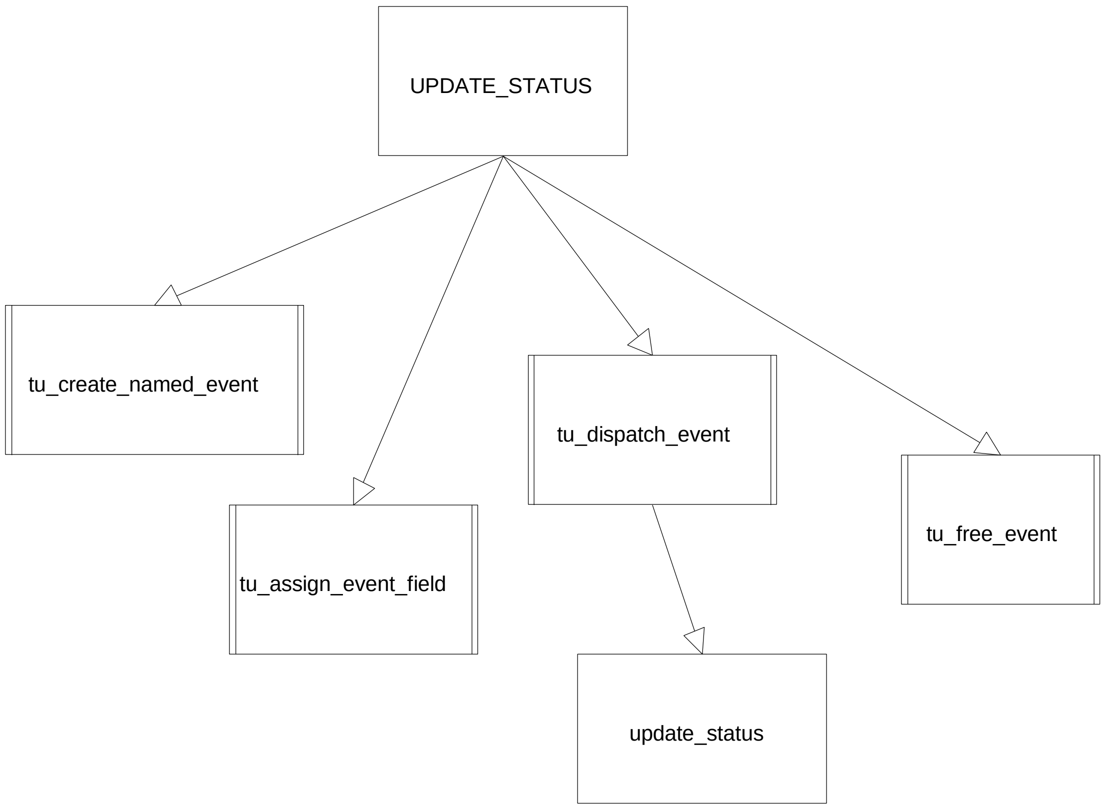


Figure 43. UPDATE_STATUS structure chart

5.3.4 award_dsif Library Routines

Routine	Description
send	A TeleUSE statement used to trigger events immediately or queue events for later dispatch.
tu_assign_event_field	TeleUSE Library Function to associate the contents of a C variable with the contents of an event attribute.
tu_create_named_event	TeleUSE Library Function used to create the data structure necessary to interface the C code with the D event code.
tu_dispatch_event	TeleUSE Library Function used to dispatch the created event. This causes the event to be executed.
tu_free_event	TeleUSE Library Function used to free up any memory that was allocated to the event data structure using tu_create_named_event.
update_indicator	This GUI layer event is used to make the actual modifications to the appropriate GUI components. The GUI component is specified as one of the calling attributes along with the current status information to be displayed in the GUI.
update_status	The D event that receives the status information and updates the appropriate GUI components.
UPDATE_STATUS	This is the bridge layer routine that receives the current status information for a particular sensor or crossing and then generates the event to update the status information within the detailed status GUI.

Table 60. Routines called by UPDATE_STATUS

The award_dsif library contains four routines that are used to interact with the award_dsif process. The routines needed to send a heartbeat to the award_dsif are described in*MDI Common Code Software Design Document*. The four routines defined by this library are:

- award_dsif_connect
- award_dsif_send_sensor_data
- award_dsif_send_crossing_delay
- award_dsif_disconnect

These routines are discussed in more detail in the following subsections.

5.3.4.1 award_dsif_connect

The structure chart for the award_dsif_connect routine is shown in Figure 44. 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.. A description of the routines called by award_dsif_connect is provided in Table 61.

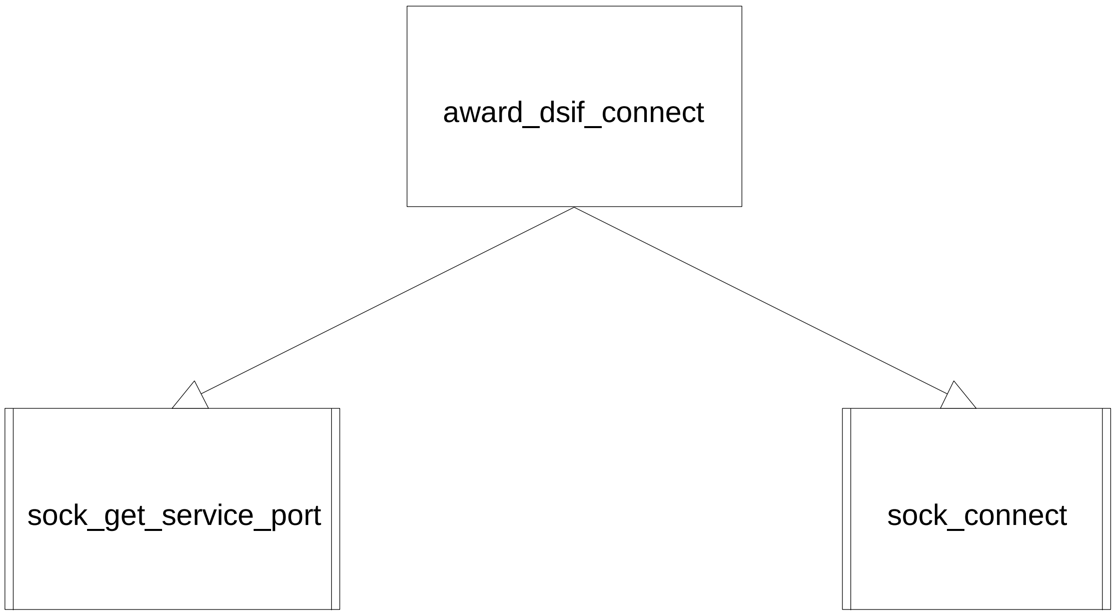


Figure 44. award_dsif_connect structure chart

Routine	Description
award_dsif_connect	Used to connect to the AWARD Data Server Interface process. The caller specifies the host name and service name associated with the AWARD Data Server Interface.
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.

Table 61. Routines called by award_dsif_connect

5.3.4.2 award_dsif_send_sensor_data

The structure chart for award_dsif_send_sensor_data is shown in Figure 45. This routine is responsible for filling in the message structure with the specified data parameters. A call to select is used to make sure the award_dsif connection is still valid. All values in the message are converted to network byte format using the htonl call, when required. A description of the routines called by award_dsif_send_sensor_data is provided inTable 62.

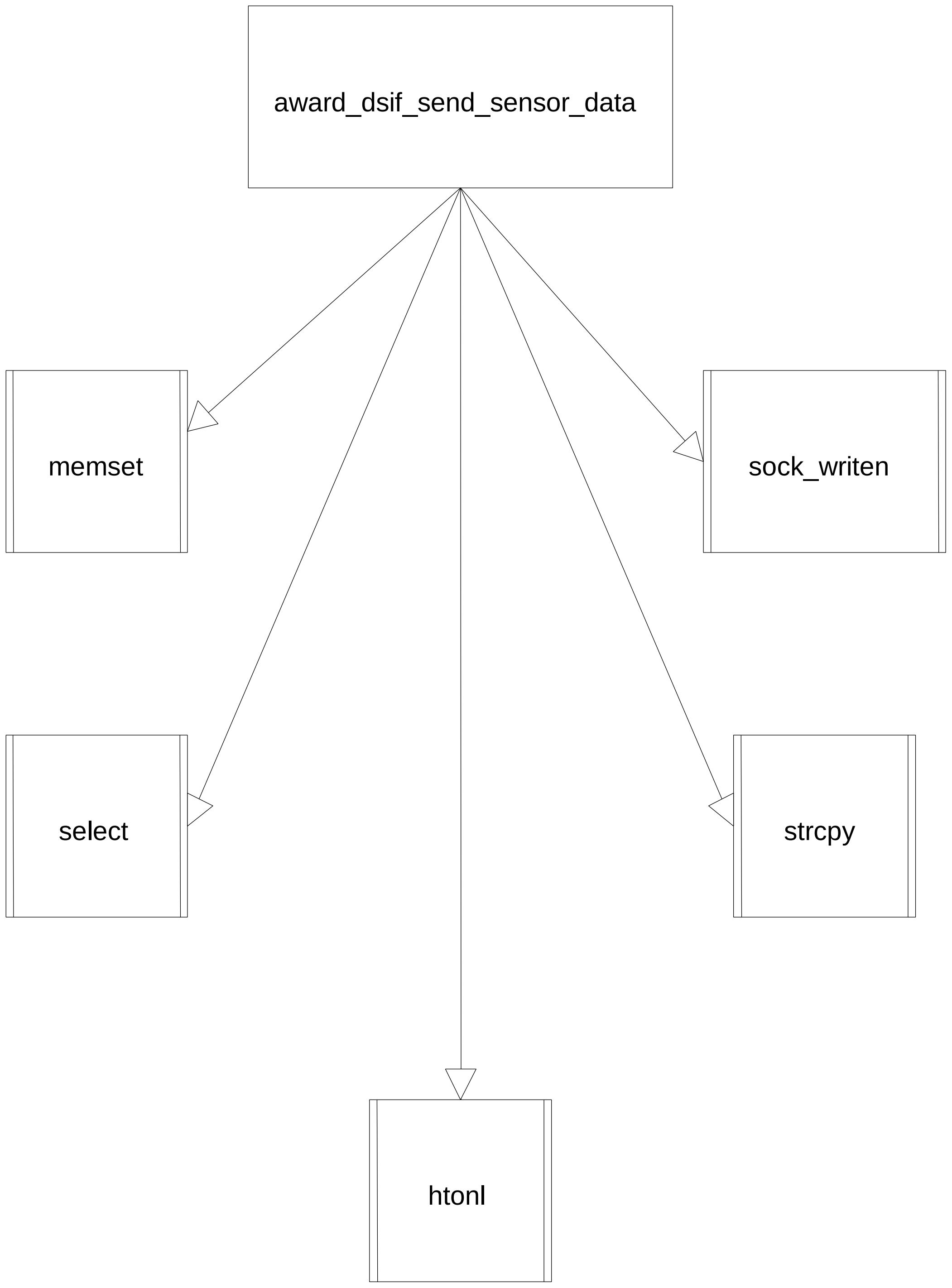


Figure 45. award_dsif_send_sensor_data structure chart

Routine	Description
award_dsif_send_sensor_data	Packages the specified sensor data into the appropriate message and sends it to the AWARD Data Server Interface process.
htonl	Network function used to convert from host to network byte formats.
memset	C Library Function used to set an area of memory to a specified value.
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.
sock_writen	MDI Socket routine used to write a specified number of bytes to a specified socket.
strcpy	C Library Function used to copy a source string to a destination string.

Table 62. Routines called by award_dsif_send_sensor_data

5.3.4.3 award_dsif_send_crossing_delay

The structure chart for award_dsif_send_crossing_delay is shown in Figure 46. This routine is responsible for filling in the message structure with the specified data parameters. A call to select is used to make sure the award_dsif connection is still valid. All values in the message are converted to network byte format using the htonl call, when required. A description of the routines called by award_dsif_send_crossing_delay is provided in Table 63.

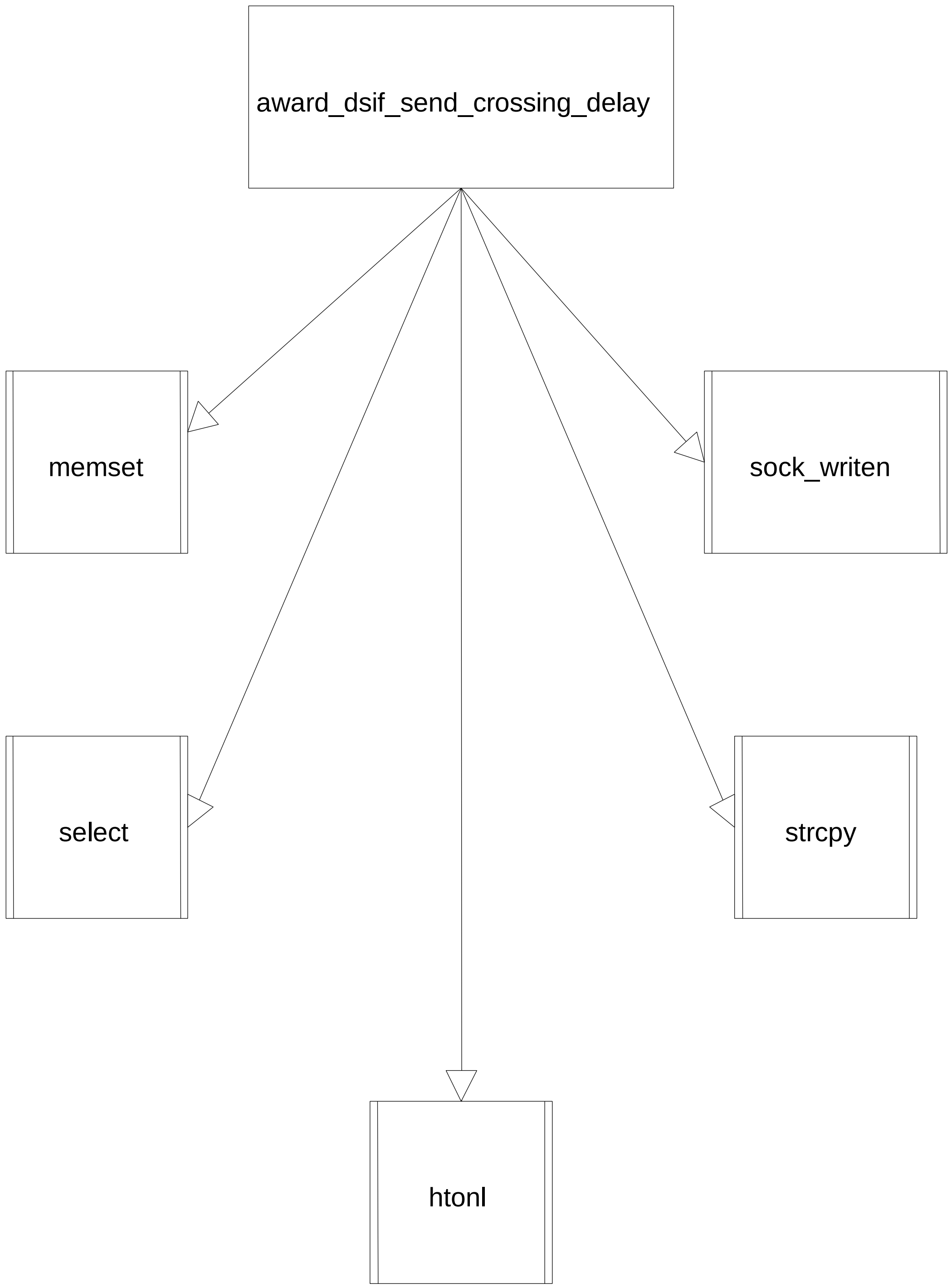


Figure 46. award_dsif_send_crossing_delay structure chart

Routine	Description
award_dsif_send_crossing_delay	Packages the specified crossing data into the appropriate message and sends it to the AWARD Data Server Interface process.
htonl	Network function used to convert from host to network byte formats.
memset	C Library Function used to set an area of memory to a specified value.
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.
sock_writen	MDI Socket routine used to write a specified number of bytes to a specified socket.
strcpy	C Library Function used to copy a source string to a destination string.

Table 63. Routines called by award_dsif_send_crossing_delay

5.3.4.4 award_dsif_disconnect

The structure chart for award_dsif_disconnect is shown in Figure 47. This routine is used to disconnect the calling process from the award_dsif process. Once this routine is called no more messages can be sent to the award_dsif process unless the award_dsif_connect routine is called. A description of the routines called by award_dsif_disconnect is provided in Table 64.

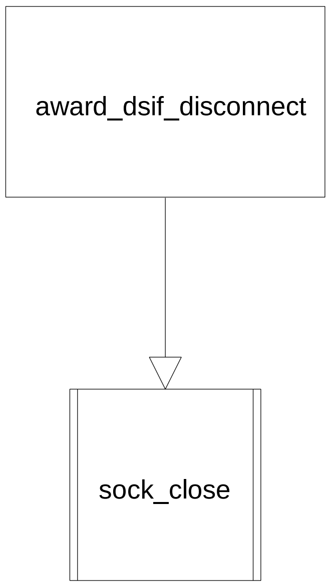


Figure 47. award_dsif_disconnect structure chart

Routine	Description
award_dsif_disconnect	Disconnects the calling process from the AWARD Data Server Interface process. The socket connection is closed and reset to indicate a connection to the AWARD Data Server Interface process does not exist.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 64. Routines called by award_dsif_disconnect

5.3.5 award_tgif Library Routines

The award_tgif library contains three routines that are used to interact with the award_tgif process. The three routines defined by this library are:

- award_tgif_connect
- award_tgif_send_crossing_blockage
- award_tgif_disconnect

These routines are discussed in more detail in the following subsections.

5.3.5.1 award_tgif_connect

The structure chart for the award_tgif_connect routine is shown in Figure 48. This routine is responsible for establishing the communications path to the award_tgif process 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. A description of the routines called by award_tgif_connect is provided in Table 65.

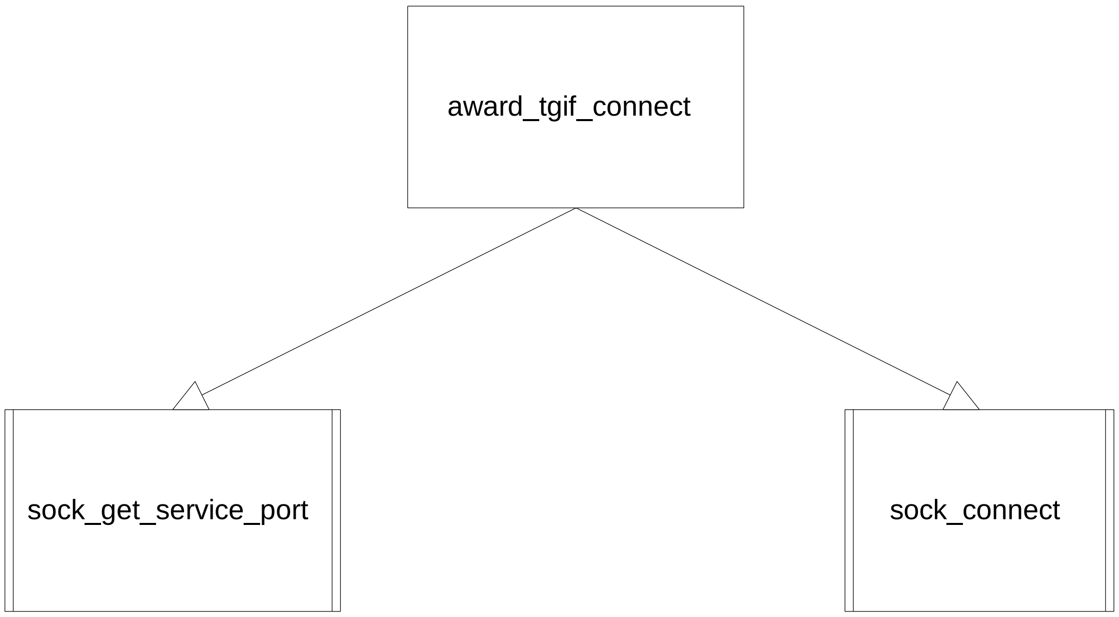


Figure 48. award_tgif_connect structure chart

Routine	Description
award_tgif_connect	Used to connect to the AWARD TransGuide Interface process. The caller specifies the host name and service name associated with the AWARD TransGuide Interface.
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.

Table 65. Routines called by award_tgif_connect

5.3.5.2 award_tgif_send_crossing_blockage

The structure chart for award_tgif_send_crossing_blockage is shown in Figure 49. This routine is responsible for filling in the message structure with the specified data parameters. A call to select is used to make sure the award_tgif connection is still valid. All values in the message are converted to network byte format using the htonl call, when required. A description of the routines called by award_tgif_send_crossing_blockage is provided in Table 66.

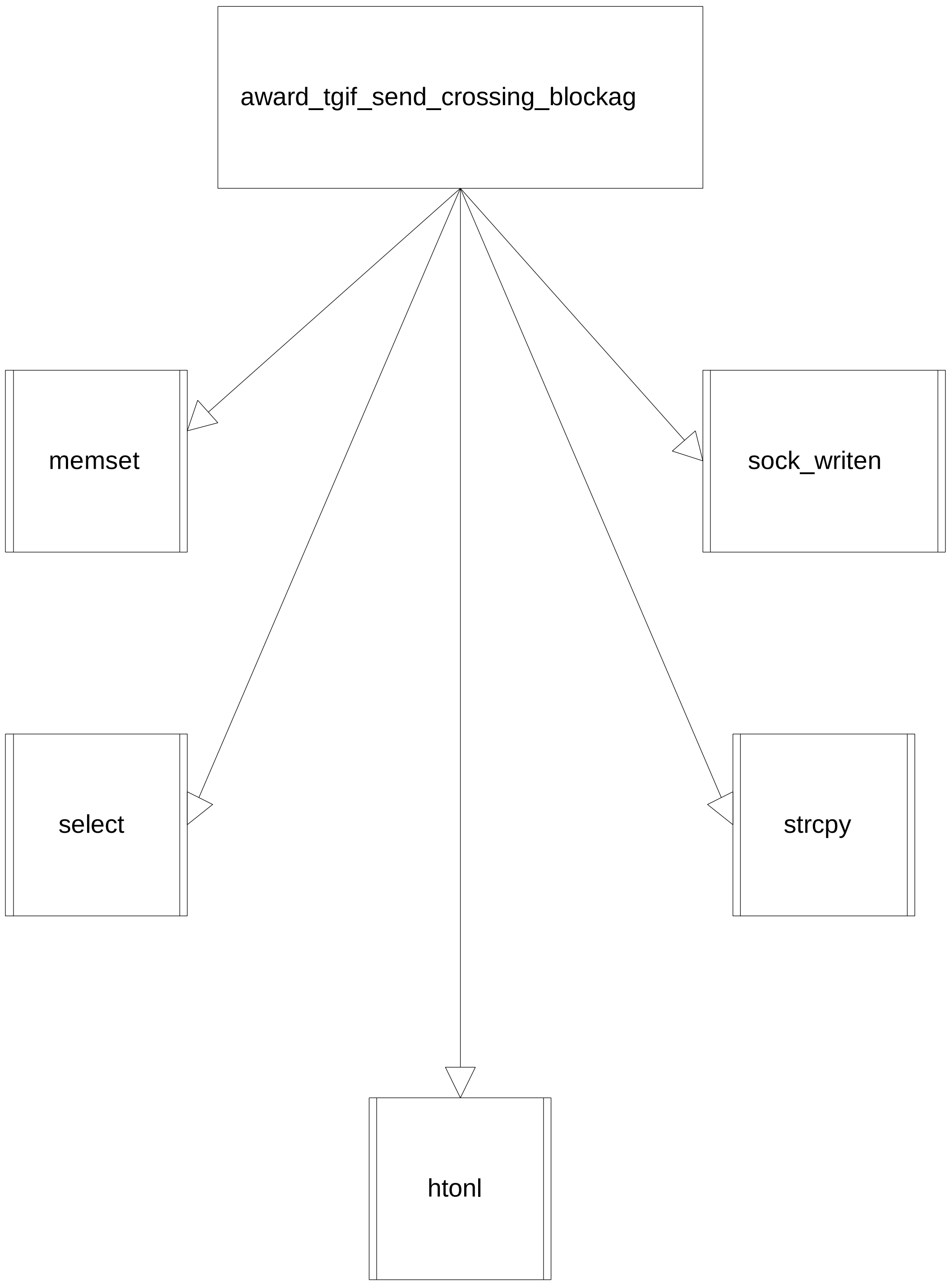


Figure 49. award_tgif_send_crossing_blockage structure chart

Routine	Description
award_tgif_send_crossing_blockag	Packages the specified crossing delay data into the crossing blockage message and sends it to the AWARD TransGuide Interface process.
htonl	Network function used to convert from host to network byte formats.
memset	C Library Function used to set an area of memory to a specified value.
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.
sock_writen	MDI Socket routine used to write a specified number of bytes to a specified socket.
strcpy	C Library Function used to copy a source string to a destination string.

Table 66. Routines called by award_tgif_send_crossing_blockage

5.3.5.3 award_tgif_disconnect

The structure chart for award_tgif_disconnect is shown in Figure 50. This routine is used to disconnect the calling process from the award_tgif process. Once this routine is called no more messages can be sent to the award_tgif process unless the award_tgif_connect routine is called. A description of the routines called by award_tgif_disconnect is provided in Table 67.

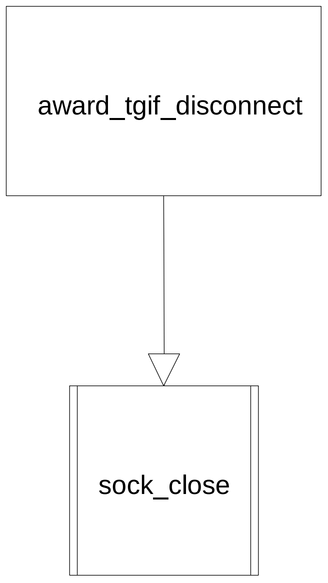


Figure 50. award_tgif_disconnect structure chart

Routine	Description
award_tgif_disconnect	Disconnects the calling process from the AWARD TransGuide Interface process. The socket connection is closed and reset to indicate a connection to the AWARD TransGuide Interface process does not exist.
sock_close	MDI Socket routine used to close the specified socket connection.

Table 67. Routines called by award_tgif_disconnect

5.4 Railroad Subsystem Software Architecture

The Railroad Operational Software (ROS) subsystem transmits the railroad sensor data and the railroad delay data to the MDI Data Server thus allowing access to this information by other programs within the MDI. The railroad sensor data is transmitted to the MDI Data Server via socket communications and consists of the following information for each physical sensor:

- Sensor ID - A unique identifier assigned to each railroad sensor.
- Communication Status - The current status of the communications between the FE Communication Subsystem and the physical sensors.
- Measurement - the speed value obtained from the sensor during the last polling cycle
- Acceleration/Deceleration - the rate of acceleration or deceleration detected during the last polling cycle.
- Last Update Time - A timestamp indicating the last time the sensor was polled.

The railroad delay data is transmitted to the MDI Data Server using socket communications and consists of the following information for each railroad crossing associated with physical railroad sensors:

- Crossing ID - A unique identifier assigned to each railroad crossing.
- Estimated ETA of the front of the train
- Estimated ETA of the back of the train
- Length of the train
- Estimated duration of the railroad crossing blockage

The ROS subsystem transmits the railroad delay information to the TransGuide ATMS to allow TransGuide operations personnel to properly handle each railroad crossing delay incident. The railroad delay data is transmitted to the TransGuide ATMS as a RR Incident Event using socket communications. The contents of the RR Incident Event are:

- Crossing ID - A unique identifier assigned to each railroad crossing
- The type of ATMS alarm (Major, Minor, or Normal)
- Estimated ETA of the front of the train
- Estimated ETA of the back of the train
- Length of the train
- Estimated duration of the railroad crossing blockage

5.4.1 RR Configuration Information

Configuration information, located in text files, enumerates sensors, virtual sensors, crossings, and connections between the virtual sensors and crossings. Specifically, each actual sensor becomes two virtual sensors; this approach was taken to reduce complexity of the configuration files. In this arrangement, it is simple to get a downstream virtual sensor associated with an upstream element; it is also somewhat verbose in that many distances between sensors, etc. will be entered twice. The connection's list requires distances to be both positive and negative items, where sign is dependent on sensor pointing direction, presenting an even greater challenge at entering the data correctly.

One additional configuration file is defined and because of the relationship to the determination of acceleration, the file definition is located in that section, below. Each of the definitions in the boxes lists all of the information that will appear on each line of data in the configuration file.

Information on Each Line of the Master Configuration File	Actual sensors configuration file name Virtual sensors configuration file name Crossings configuration file name Downstream connections configuration file name AWARD ATMS events configuration file name Acceleration modification configuration file name Name of the DataServer interface process host machine Name of the DataServer service Name of the Heartbeat interface process host machine Name of the Heartbeat service Name of the ATMS interface process host machine Name of the ATMS service Name of the status logger interface process host machine Name of the status logger service Use the acceleration in calculations flag (yes or no) System poll cycle time (minimum)
Information on Each Line of the Actual Sensors List	Sensor ID (a1..aN) Telephone number SUN port id to use Angle of the sensor versus the tracks Maximum velocity for this site Sensor is operational (flag) Alpha-numeric text descriptor (ATMS identifier) Comment
Information on Each Line of the Virtual Sensors List (There must be 2 virtual sensors for each actual sensor - one associated with each direction of train movement)	Virtual sensor ID (s1..sM) Actual sensor associated (a1..aN) Alpha-numeric text descriptor Direction with respect to the sensor (i.e. says whether positive or negative velocities from the actual sensor belong with this virtual sensor) Comment (like "at Dreamland heading towards downtown")
Information on Each Line of the Crossings List	Crossing ID (c1..cP) Alpha-numeric text descriptor Nominal time to train crossing sensing distance (0 if no crossing signals) Comment
Information on Each Line of the AWARD ATMS Events List	Crossing ID (c1..cP) Alpha-numeric text descriptor of the name of the crossing Alpha-numeric text descriptor of the event ID Time the event is to start Time the event is to end Comment
Information on Each Line of the Connections List (Just virtual sensors to downstream crossings and other downstream virtual sensors)	Virtual sensor (s1..sM) Downstream item (s1..sM or c1..cP) Distance (must be positive in miles) Comment

Information on Each Line of the Acceleration Modification List (Associated with virtual sensors)	Virtual sensor (s1..sM) Day of week (1..7) Start and End time of affect Interpretable code defining affect on acceleration Comment

Table 68. Railroad Configuration Information

5.4.2 RR Software Design Details

To explain the diagram (and software design) below, the RR Software Object Relationship Diagram, in some more detail, the diagram reveals that the actual sensor object (SensorIF), actually maintains a list of samples where each has the sensor reading and a timestamp (ValTime). These values are processed, resulting in an interpretation of the activity on the tracks. This information is used to update the virtual sensor (Vsensor) objects, which creates or updates a train summary (TrainSumry) object. A train summary object (which is associated with a particular virtual sensor) may also be the same train that another sensor is detecting or has detected in the recent past. Therefore, train summaries are merged to form a single unified interpretation of a train (Train) object, including length, speed, acceleration, etc. There are two important pieces of information the ROS needs to know about a train: when is it going to block a crossing and when should it be detected by some downstream sensor? Answering these two questions requires knowledge of where downstream sensors and crossings are located (Connection Descriptions) and coming up with an estimate of how long before the train reaches them (Connection ETA). When a train is certain to block an intersection, some description of the blockage is communicated to the Area Wide Database (and therefore to In-Vehicle Navigation, Kiosks, and the TransGuide Map) as well as to the TransGuide ATMS.

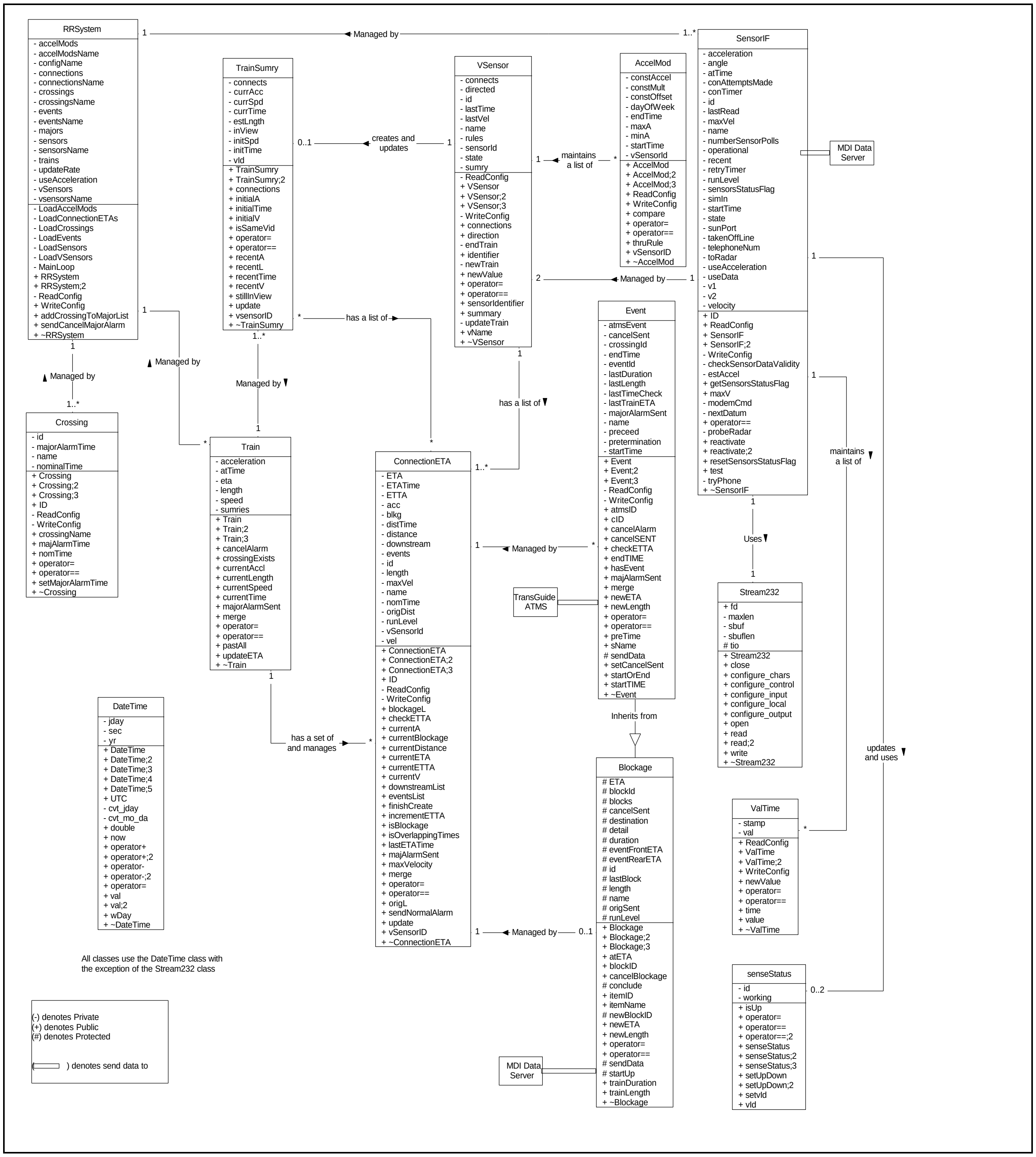


Figure 51. RR Software Object Relationship Diagram

As the diagram above illustrates, the ROS is a collection of objects that holds and manipulates data or manages and manipulates the objects (that hold data) to inform the TransGuide ATMS and the Area Wide Database component of MDI of upcoming crossing blockages. The classes are described in more detail below.

5.4.2.1 Railroad System (RRSystem) Class

This class is the topmost of the AWARD system. The class, upon creation loads configuration information about train sensors and begins the processing loop. The class also maintains the current list of trains that are of interest. The class cannot be copied because the Stream232 class (which is part of SensorIF) cannot be copied. The data maintained by (within) the class includes:

- name of master configuration file
- log handling pointers/references
- a list of sensor interfaces (SensorIF class)
- a list of crossings (Crossing class)
- a list of trains (Train class)

The RRSystem class contains the following attributes:

PRIVATE:

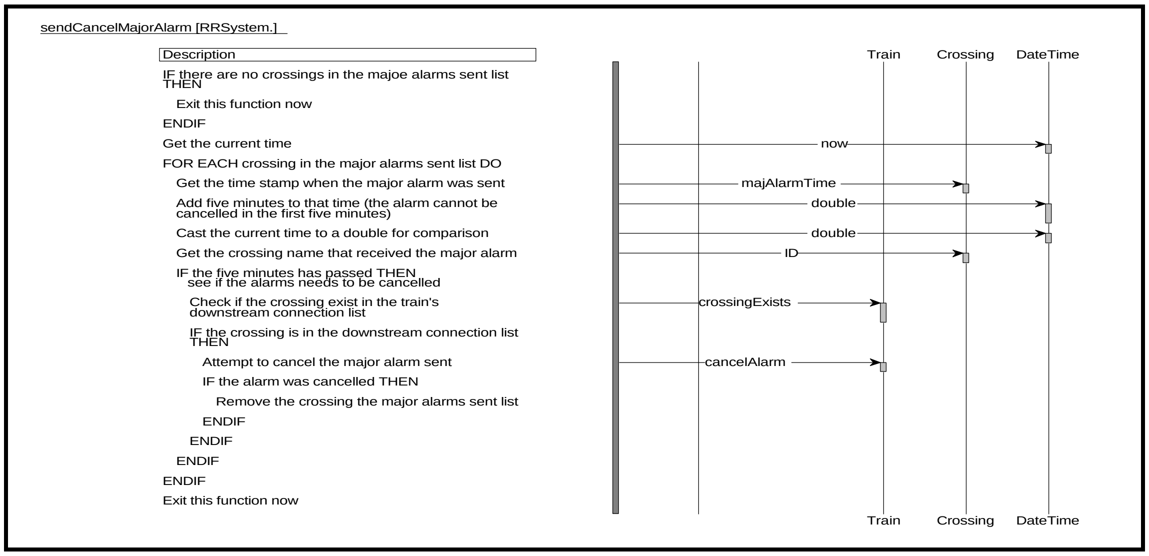
accelMods	List of all acceleration modifications.
accelModsName	Acceleration modification configuration file name.
configName	Master configuration file name.
connections	List of all downstream connections.
connectionsName	Downstream connections configuration file name.
crossings	List of all crossings.
crossingsName	Crossings configuration file name.
events	List of all ATMS events.
eventsName	ATMS events configuration file name.
majors	List of all major alarms sent to ATMS that have NOT been cancelled.
sensors	List of all actual sensors.
sensorsName	Actual sensors configuration file name.
trains	List of all active trains.
updateRate	Duration between normal polling of the sensors.

useAcceleration	Flag indicating to use calculated acceleration in the estimates for sending alarms. This flag is set via the configuration file.
vSensors	List of all virtual sensors
vsensorsName	Virtual sensors configuration file name.

The RRSystem class contains the following operations:

PUBLIC:

`addCrossingToMajorList`	Adds a crossing to the major alarm's sent to ATMS list.
`RRSystem`	Default constructor. Note: the default constructor generates a runtime error if it is used without an initializer.
`RRSystem;2`	Constructor that takes a configuration file name as and the run level indicator as initializers.
`sendCancelMajorAlarm`	Sends a cancel event for a previously sent major alarm. The sequence diagram for this member function is given in Figure 52.



```
IF there are no crossings in the majoe alarms sent list
THEN
  Exit this function now
ENDIF
Get the current time
FOR EACH crossing in the major alarms sent list DO
  Get the time stamp when the major alarm was sent
  Add five minutes to that time (the alarm cannot be
  cancelled in the first five minutes)
  Cast the current time to a double for comparison
  Get the crossing name that received the major alarm
  IF the five minutes has passed THEN
    see if the alarms needs to be cancelled
    Check if the crossing exist in the train's
    downstream connection list
    IF the crossing is in the downstream connection list
    THEN
      Attempt to cancel the major alarm sent
      IF the alarm was cancelled THEN
        Remove the crossing the major alarms sent list
      ENDIF
    ENDIF
  ENDIF
ENDIF
Exit this function now
```

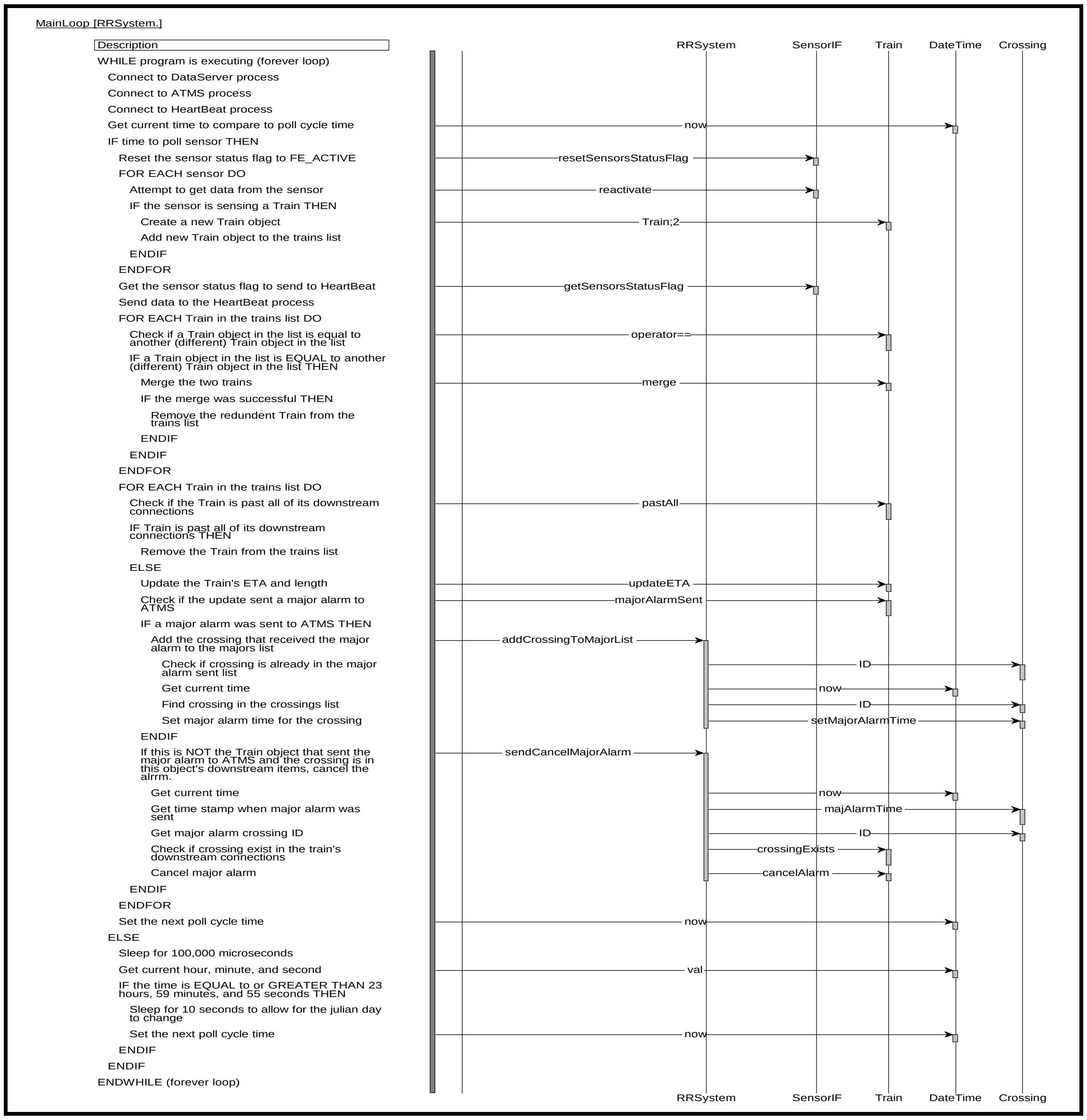
Figure 52. RRSystem::sendCancelMajorAlarm Sequence Diagram

`WriteConfig`	Write actual or example configuration data to a file.
`~RRSystem`	Destructor.

PRIVATE:

`LoadAccelMods`	Create the acceleration modifications list and initialize it from its configuration file.
`LoadConnectionETAs`	Create the downstream connections list and initialize it from its configuration file.
`LoadCrossings`	Create the crossing list and initialize it from its configuration file.
`LoadEvents`	Create the events list and initialize it from its configuration file.
`LoadSensors`	Create the actual sensors list and initialize it from its configuration file.
`LoadVSensors`	Create the virtual sensors list and initialize it from its configuration file.

`MainLoop`	Executes the main processing loop. It polls the sensors, updates existing trains, etc. The sequence diagram for this member function is given in Figure 53.


MainLoop [RRSystem.]

```
Description
WHILE program is executing (forever loop)
  Connect to DataServer process
  Connect to ATMS process
  Connect to HeartBeat process
  Get current time to compare to poll cycle time
  IF time to poll sensor THEN
    Reset the sensor status flag to FE_ACTIVE
    FOR EACH sensor DO
      Attempt to get data from the sensor
      IF the sensor is sensing a Train THEN
        Create a new Train object
        Add new Train object to the trains list
      ENDIF
    ENDFOR
    Get the sensor status flag to send to HeartBeat
    Send data to the HeartBeat process
    FOR EACH Train in the trains list DO
      Check if a Train object in the list is equal to
      another (different) Train object in the list
      IF a Train object in the list is EQUAL to another
      (different) Train object in the list THEN
        Merge the two trains
        IF the merge was successful THEN
          Remove the redundent Train from the
          trains list
        ENDIF
      ENDIF
    ENDFOR
    FOR EACH Train in the trains list DO
      Check if the Train is past all of its downstream
      connections
      IF Train is past all of its downstream
      connections THEN
        Remove the Train from the trains list
      ELSE
        Update the Train's ETA and length
        Check if the update sent a major alarm to
        ATMS
        IF a major alarm was sent to ATMS THEN
          Add the crossing that received the major
          alarm to the majors list
            Check if crossing is already in the major
            alarm sent list
            Get current time
            Find crossing in the crossings list
            Set major alarm time for the crossing
        ENDIF
        If this is NOT the Train object that sent the
        major alarm to ATMS and the crossing is in
        this object's downstream items, cancel the
        alrm.
          Get current time
          Get time stamp when major alarm was
          sent
          Get major alarm crossing ID
          Check if crossing exist in the train's
          downstream connections
          Cancel major alarm
      ENDIF
    ENDFOR
    Set the next poll cycle time
  ELSE
    Sleep for 100,000 microseconds
    Get current hour, minute, and second
    IF the time is EQUAL to or GREATER THAN 23
    hours, 59 minutes, and 55 seconds THEN
      Sleep for 10 seconds to allow for the julian day
      to change
      Set the next poll cycle time
    ENDIF
  ENDIF
ENDWHILE (forever loop)
```

Figure 53. RRSystem::MainLoop Sequence Diagram

`ReadConfig`	Reads the configuration information.

5.4.2.2 Sensor Interface (SensorIF) Class

This class is responsible for interfacing and controlling the Stream232 class that communicates to the remote radar units. The class implements a finite state machine (depicted in the figure below). When the class is created, it must be provided configuration information. Then most later interactions are simply to tell it to reactivate (leave some internal wait state and run) which causes it to attempt to get data from the remote sensor, determine if it is valid, and send it to the appropriate virtual sensor. This class cannot be copied because the Stream232 object, which the class contains, cannot be copied. When the configuration data for the class is loaded, a flag is set which indicates that the remote sensor is operational. If it is not, the class does not initialize and cannot be used.

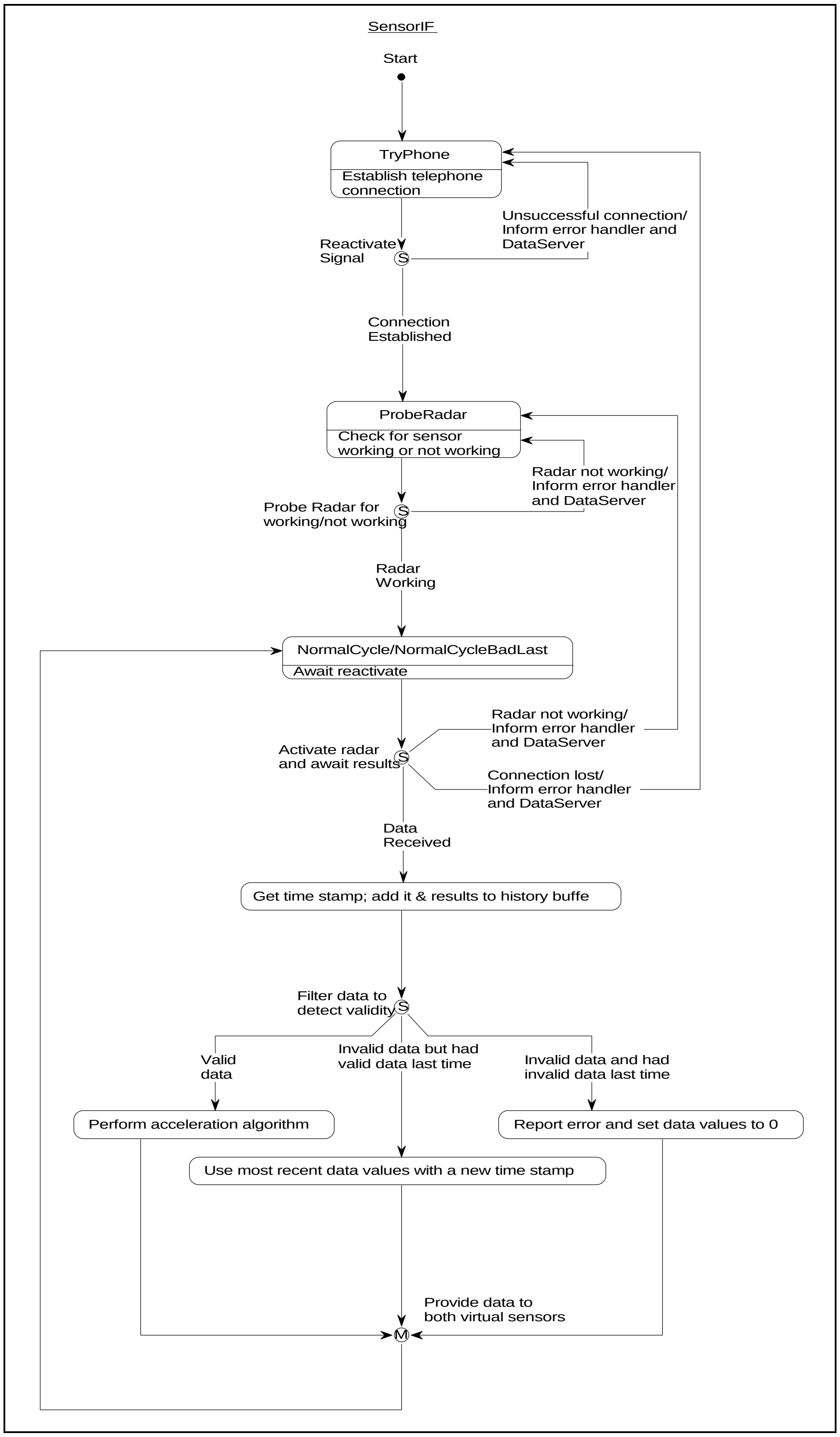

SensorIF
Start
TryPhone
Establish telephone connection
Reactivate Signal
S
Unsuccessful connection/ Inform error handler and DataServer
Connection Established
ProbeRadar
Check for sensor working or not working
Probe Radar for working/not working
S
Radar not working/ Inform error handler and DataServer
Radar Working
NormalCycle/NormalCycleBadLast
Await reactivate
Activate radar and await results
S
Radar not working/ Inform error handler and DataServer
Connection lost/ Inform error handler and DataServer
Data Received
Get time stamp; add it & results to history buffe
Filter data to detect validity
S
Valid data
Invalid data but had valid data last time
Invalid data and had invalid data last time
Perform acceleration algorithm
Report error and set data values to 0
Use most recent data values with a new time stamp
Provide data to both virtual sensors
M

Figure 54. Sensor Interface (SensorIF) Class State Diagram

The SensorIF class contains the following attributes:

PRIVATE:

acceleration	The current estimate of acceleration.
angle	The angle of the sensor to the track in degrees (the sign of the angle is ignored).
atTime	The timestamp associated with the current values.
conAttemptsMade	The number of consecutive connection attempts made to a sensor.
conTimer	Time when to attempt a reconnection to a sensor that was removed from service due to a loss of communication.
id	The actual sensor identifier, a1..aN.
lastRead	Last entry read from file. Note: This field is required for simulation operations.
maxVel	The maximum velocity this sensor should see.
name	The ATMS name of the sensor.
numberSensorPolls	The number of sensor polls to keep the useData flag set to TRUE because the data valid bit from the sensor was set to TRUE.
operational	Configuration parameter indicating the unit is known to be up or down.
recent	A list of recent recorded values and timestamps.
retryTimer	If TRUE, attempt reconnection, after the proper time has elapsed, to a sensor that was removed from service due to a loss of communication.
runLevel	The program's execution level. 0 is normal operations, -3 is simulation mode.
sensorsStatusFlag	Global status flag used to update the MDI heartbeat process. This global variable is only visible to SensorIF objects. If one sensors fails, the flag is used to inform ATMS of a possible warning.

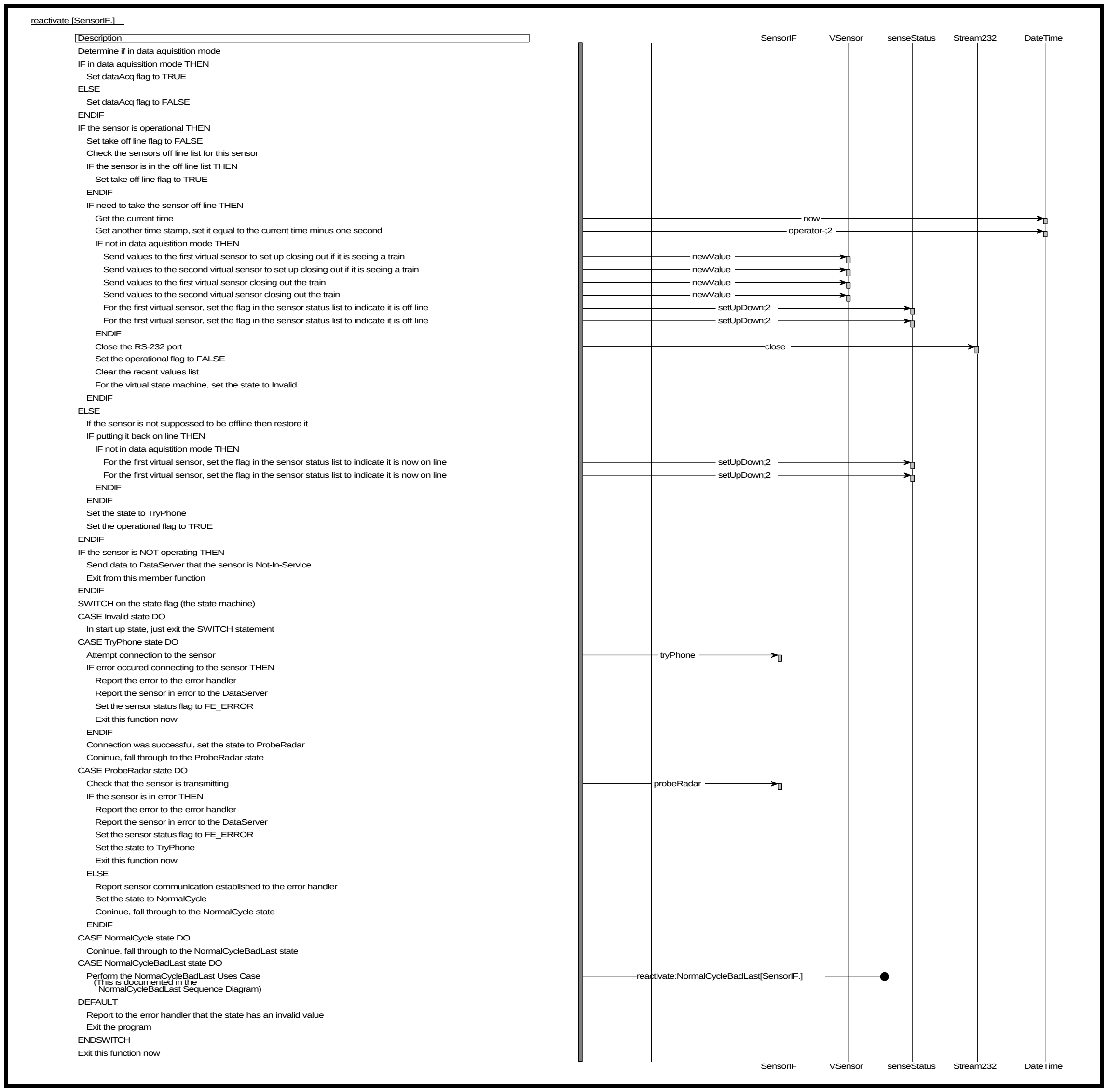
simIn	The simulation file. Note: This field is required for simulation operations.
startTime	The beginning simulation time. Note: This field is required for simulation operations.
state	The current state with respect to the finite state machine.
sunPort	SUN port id to use to communicate with the radar.
takenOffLine	If TRUE, the sensor has been removed from service due to a lost communication and unable to reconnect.
telephoneNum	The telephone number of the radar unit.
toRadar	The Stream232 interface to the Sensor.
useAcceleration	Flag indicating to use calculated acceleration in the estimates for sending alarms.
useData	Flag indicating to use the data from the sensor because the current or previous data valid bit (from the sensor) was set to TRUE.
v1	Virtual sensor #1.
v2	Virtual sensor #2.
velocity	The current (radar) velocity.

The SensorIF class contains the following operations:

PUBLIC:

`ID`	Returns the actual sensor id.
`ReadConfig`	Reads original configuration information.
`SensorIF`	Default constructor. Note: the default constructor generates a runtime error (on purpose) so it should never be used.
`SensorIF;2`	Constructor that loads in configuration data from a supplied data file.
`getSensorsStatusFlag`	Returns the global sensor status flag.
`maxV`	Returns the maximum velocity the sensor should see.

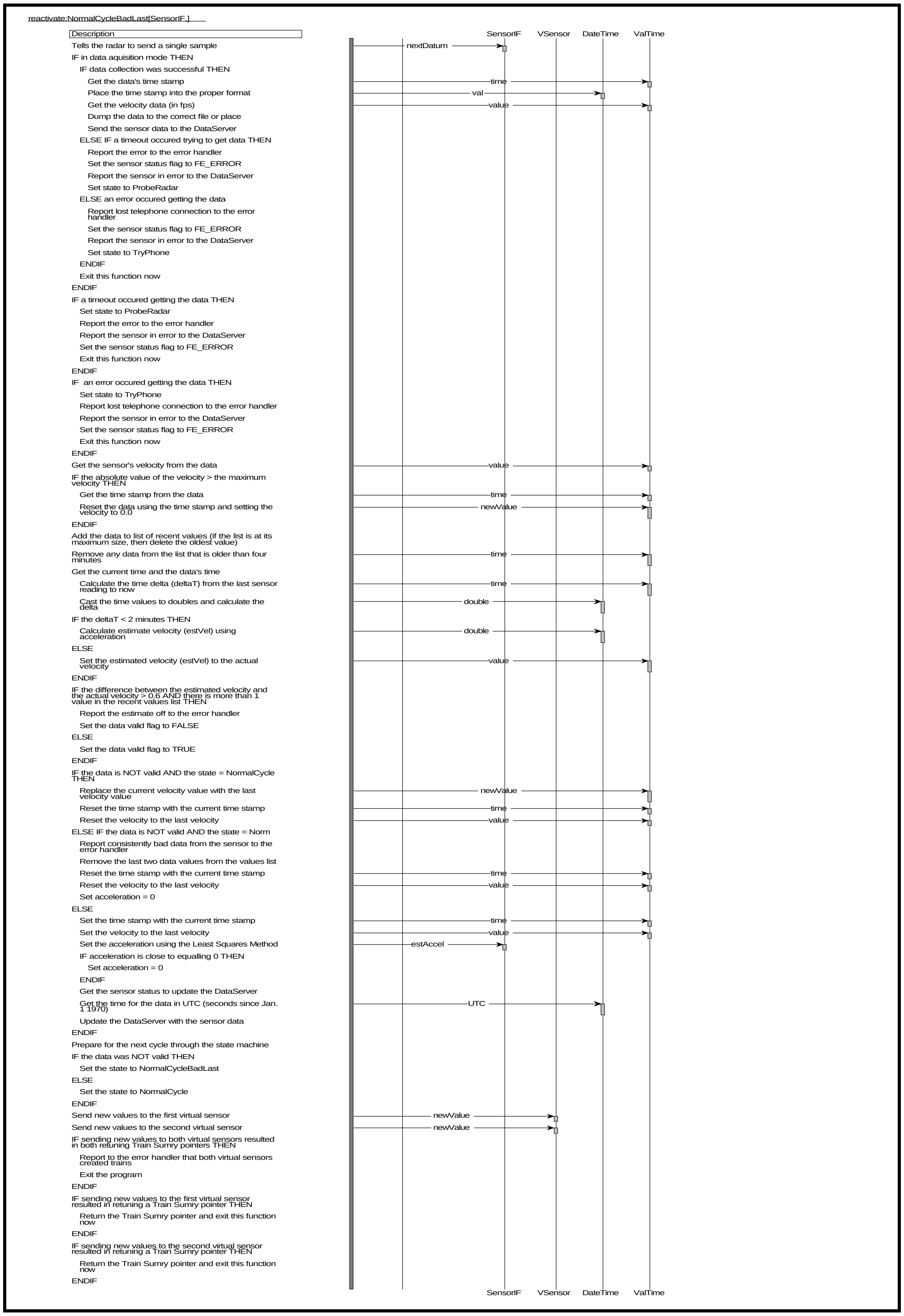
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`reactivate`	Get data from the radar unit and process it.
`reactivate;2`	Get data from the radar unit and process it for data acquisition. The sequence diagrams for this member function are given in Figure 55 and Figure 56.



reactivate [SensorIF.]

```
Description
Determine if in data aquistition mode
IF in data aquissition mode THEN
  Set dataAcq flag to TRUE
ELSE
  Set dataAcq flag to FALSE
ENDIF
IF the sensor is operational THEN
  Set take off line flag to FALSE
  Check the sensors off line list for this sensor
  IF the sensor is in the off line list THEN
    Set take off line flag to TRUE
  ENDIF
  IF need to take the sensor off line THEN
    Get the current time
    Get another time stamp, set it equal to the current time minus one second
    IF not in data aquistition mode THEN
      Send values to the first virtual sensor to set up closing out if it is seeing a train
      Send values to the second virtual sensor to set up closing out if it is seeing a train
      Send values to the first virtual sensor closing out the train
      Send values to the second virtual sensor closing out the train
      For the first virtual sensor, set the flag in the sensor status list to indicate it is off line
      For the first virtual sensor, set the flag in the sensor status list to indicate it is off line
    ENDIF
    Close the RS-232 port
    Set the operational flag to FALSE
    Clear the recent values list
    For the virtual state machine, set the state to Invalid
  ENDIF
ELSE
  If the sensor is not suppossed to be offline then restore it
  IF putting it back on line THEN
    IF not in data aquistition mode THEN
      For the first virtual sensor, set the flag in the sensor status list to indicate it is now on line
      For the first virtual sensor, set the flag in the sensor status list to indicate it is now on line
    ENDIF
  ENDIF
  Set the state to TryPhone
  Set the operational flag to TRUE
ENDIF
IF the sensor is NOT operating THEN
  Send data to DataServer that the sensor is Not-In-Service
  Exit from this member function
ENDIF
SWITCH on the state flag (the state machine)
CASE Invalid state DO
  In start up state, just exit the SWITCH statement
CASE TryPhone state DO
  Attempt connection to the sensor
  IF error occured connecting to the sensor THEN
    Report the error to the error handler
    Report the sensor in error to the DataServer
    Set the sensor status flag to FE_ERROR
    Exit this function now
  ENDIF
  Connection was successful, set the state to ProbeRadar
  Coninue, fall through to the ProbeRadar state
CASE ProbeRadar state DO
  Check that the sensor is transmitting
  IF the sensor is in error THEN
    Report the error to the error handler
    Report the sensor in error to the DataServer
    Set the sensor status flag to FE_ERROR
    Set the state to TryPhone
    Exit this function now
  ELSE
    Report sensor communication established to the error handler
    Set the state to NormalCycle
    Coninue, fall through to the NormalCycle state
  ENDIF
CASE NormalCycle state DO
  Coninue, fall through to the NormalCycleBadLast state
CASE NormalCycleBadLast state DO
  Perform the NormaCycleBadLast Uses Case
    (This is documented in the
     NormalCycleBadLast Sequence Diagram)
DEFAULT
  Report to the error handler that the state has an invalid value
  Exit the program
ENDSWITCH
Exit this function now
```

Figure 55. SensorIF::reactivate Sequence Diagram



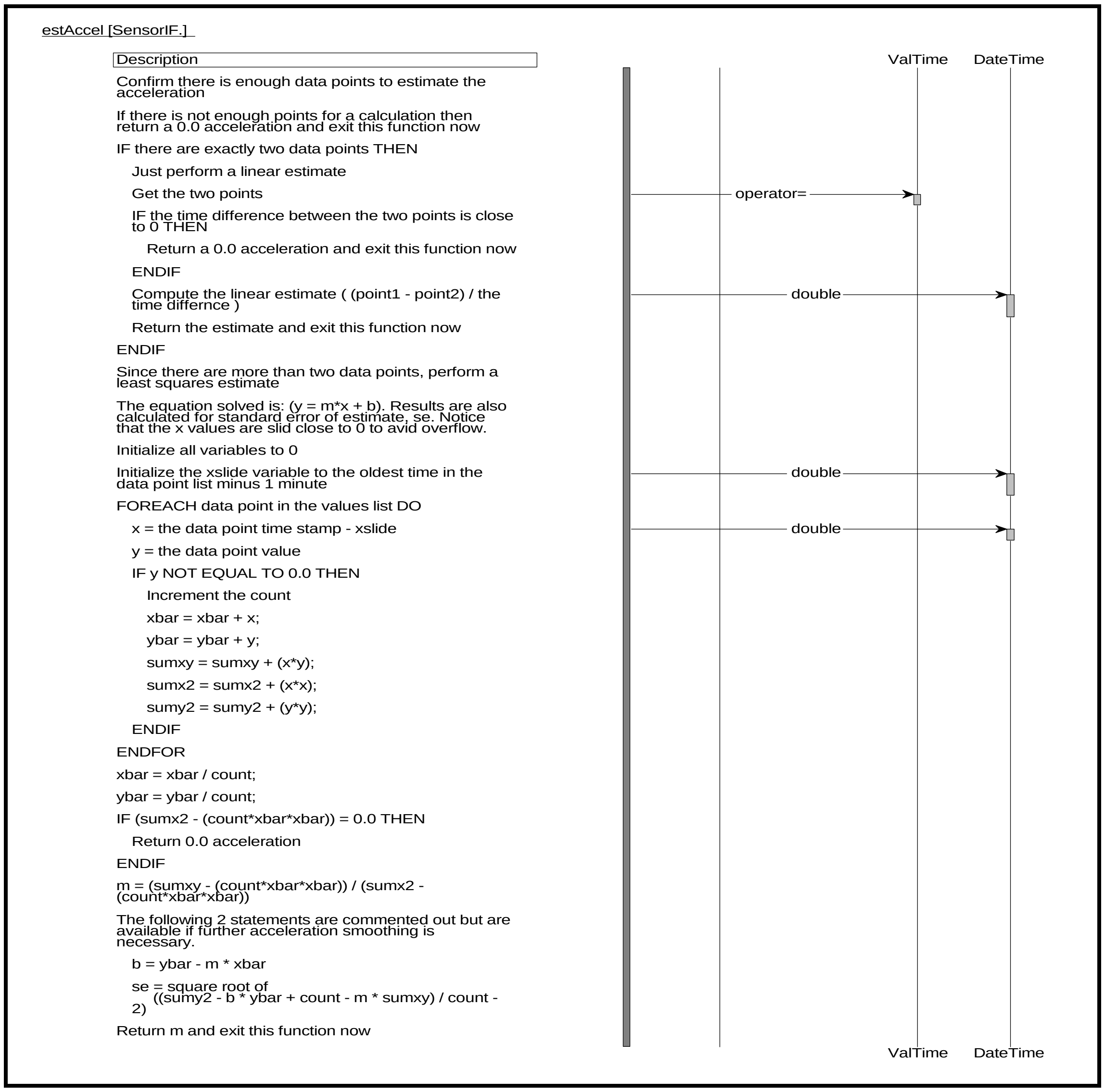
```
Description
Tells the radar to send a single sample
IF in data aquisition mode THEN
  IF data collection was successful THEN
    Get the data's time stamp
    Place the time stamp into the proper format
    Get the velocity data (in fps)
    Dump the data to the correct file or place
    Send the sensor data to the DataServer
  ELSE IF a timeout occured trying to get data THEN
    Report the error to the error handler
    Set the sensor status flag to FE_ERROR
    Report the sensor in error to the DataServer
    Set state to ProbeRadar
  ELSE an error occured getting the data
    Report lost telephone connection to the error
    handler
    Set the sensor status flag to FE_ERROR
    Report the sensor in error to the DataServer
    Set state to TryPhone
  ENDIF
  Exit this function now
ENDIF
IF a timeout occured getting the data THEN
  Set state to ProbeRadar
  Report the error to the error handler
  Report the sensor in error to the DataServer
  Set the sensor status flag to FE_ERROR
  Exit this function now
ENDIF
IF  an error occured getting the data THEN
  Set state to TryPhone
  Report lost telephone connection to the error handler
  Report the sensor in error to the DataServer
  Set the sensor status flag to FE_ERROR
  Exit this function now
ENDIF
Get the sensor's velocity from the data
IF the absolute value of the velocity > the maximum
velocity THEN
  Get the time stamp from the data
  Reset the data using the time stamp and setting the
  velocity to 0.0
ENDIF
Add the data to list of recent values (if the list is at its
maximum size, then delete the oldest value)
Remove any data from the list that is older than four
minutes
Get the current time and the data's time
  Calculate the time delta (deltaT) from the last sensor
  reading to now
  Cast the time values to doubles and calculate the
  delta
IF the deltaT < 2 minutes THEN
  Calculate estimate velocity (estVel) using
  acceleration
ELSE
  Set the estimated velocity (estVel) to the actual
  velocity
ENDIF
IF the difference between the estimated velocity and
the actual velocity > 0.6 AND there is more than 1
value in the recent values list THEN
  Report the estimate off to the error handler
  Set the data valid flag to FALSE
ELSE
  Set the data valid flag to TRUE
ENDIF
IF the data is NOT valid AND the state = NormalCycle
THEN
  Replace the current velocity value with the last
  velocity value
  Reset the time stamp with the current time stamp
  Reset the velocity to the last velocity
ELSE IF the data is NOT valid AND the state = Norm
  Report consistently bad data from the sensor to the
  error handler
  Remove the last two data values from the values list
  Reset the time stamp with the current time stamp
  Reset the velocity to the last velocity
  Set acceleration = 0
ELSE
  Set the time stamp with the current time stamp
  Set the velocity to the last velocity
  Set the acceleration using the Least Squares Method
  IF acceleration is close to equalling 0 THEN
    Set acceleration = 0
  ENDIF
  Get the sensor status to update the DataServer
  Get the time for the data in UTC (seconds since Jan.
  1 1970)
  Update the DataServer with the sensor data
ENDIF
Prepare for the next cycle through the state machine
IF the data was NOT valid THEN
  Set the state to NormalCycleBadLast
ELSE
  Set the state to NormalCycle
ENDIF
Send new values to the first virtual sensor
Send new values to the second virtual sensor
IF sending new values to both virtual sensors resulted
in both retuning Train Sumry pointers THEN
  Report to the error handler that both virtual sensors
  created trains
  Exit the program
ENDIF
IF sending new values to the first virtual sensor
resulted in retuning a Train Sumry pointer THEN
  Return the Train Sumry pointer and exit this function
  now
ENDIF
IF sending new values to the second virtual sensor
resulted in retuning a Train Sumry pointer THEN
  Return the Train Sumry pointer and exit this function
  now
ENDIF
```

Figure 56. SensorIF::reactivate:NormalCycleBadLast (Use Case) Sequence Diagram

`resetSensorsStatusFlag`	Resets the global sensor status flag.
`~SensorIF`	Destructor.

PRIVATE:

`checkSensorDataValidity`	Polls the sensor for multiple data sets to check for the data valid bits. Returns TRUE if any of the data sets has a TRUE setting of the data valid bit.
`estAccel`	Estimates acceleration from the current and previous velocities. The sequence diagram for this member function is given in Figure 57.



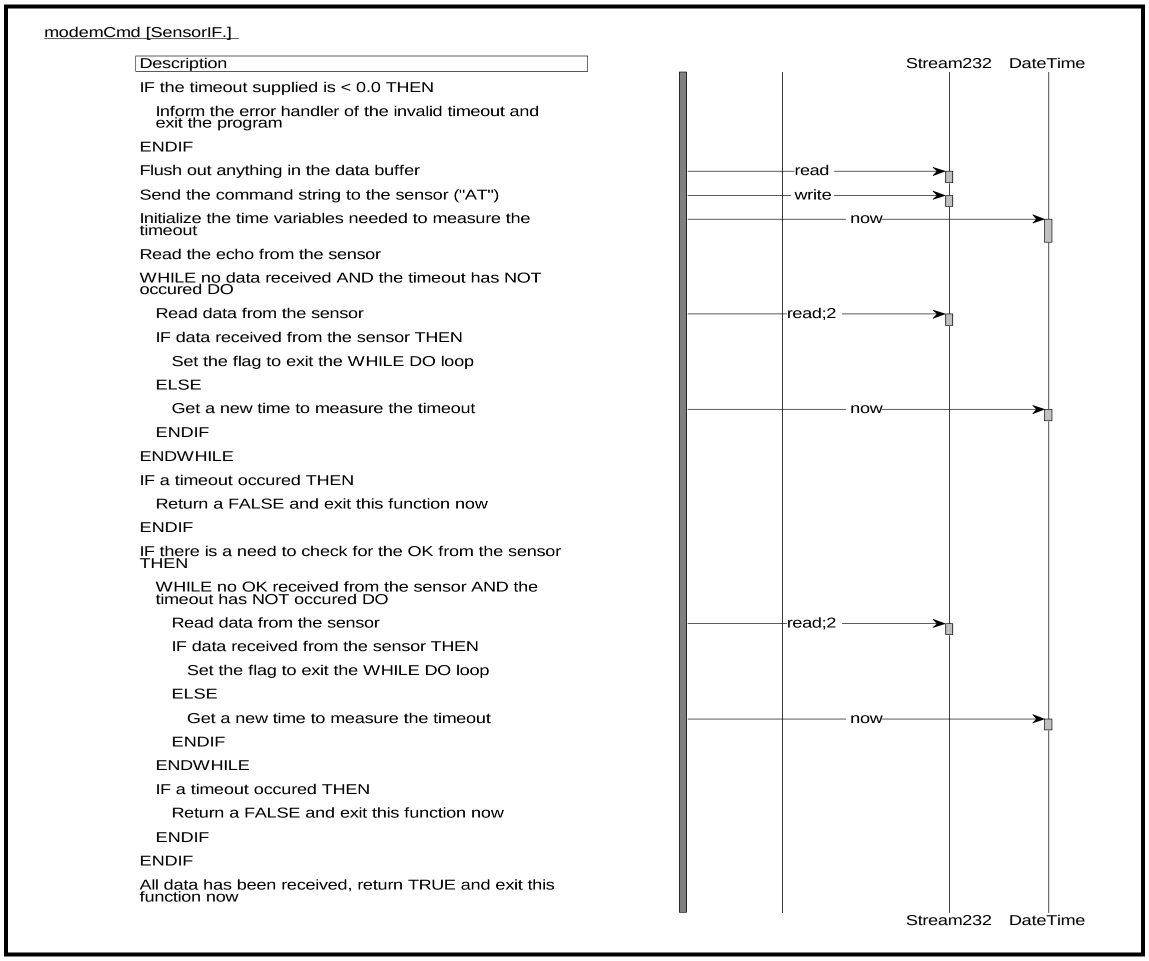
```
Confirm there is enough data points to estimate the acceleration
If there is not enough points for a calculation then return a 0.0 acceleration and exit this function now
IF there are exactly two data points THEN
   Just perform a linear estimate
   Get the two points
   IF the time difference between the two points is close to 0 THEN
      Return a 0.0 acceleration and exit this function now
   ENDIF
   Compute the linear estimate ( (point1 - point2) / the time differnce )
   Return the estimate and exit this function now
ENDIF
Since there are more than two data points, perform a least squares estimate
The equation solved is: (y = m*x + b). Results are also calculated for standard error of estimate, se. Notice that the x values are slid close to 0 to avid overflow.
Initialize all variables to 0
Initialize the xslide variable to the oldest time in the data point list minus 1 minute
FOREACH data point in the values list DO
   x = the data point time stamp - xslide
   y = the data point value
   IF y NOT EQUAL TO 0.0 THEN
      Increment the count
      xbar = xbar + x;
      ybar = ybar + y;
      sumxy = sumxy + (x*y);
      sumx2 = sumx2 + (x*x);
      sumy2 = sumy2 + (y*y);
   ENDIF
ENDFOR
xbar = xbar / count;
ybar = ybar / count;
IF (sumx2 - (count*xbar*xbar)) = 0.0 THEN
   Return 0.0 acceleration
ENDIF
m = (sumxy - (count*xbar*xbar)) / (sumx2 - (count*xbar*xbar))
The following 2 statements are commented out but are available if further acceleration smoothing is necessary.
   b = ybar - m * xbar
   se = square root of
      ((sumy2 - b * ybar + count - m * sumxy) / count - 2)
Return m and exit this function now
```

Figure 57. SensorIF::estAccel Sequence Diagram

`WriteConfig`	Write actual or example configuration data to a file.

`modemCmd`

Sends the "AT" command to the modem.
The sequence diagram for this member function is given in Figure 58.

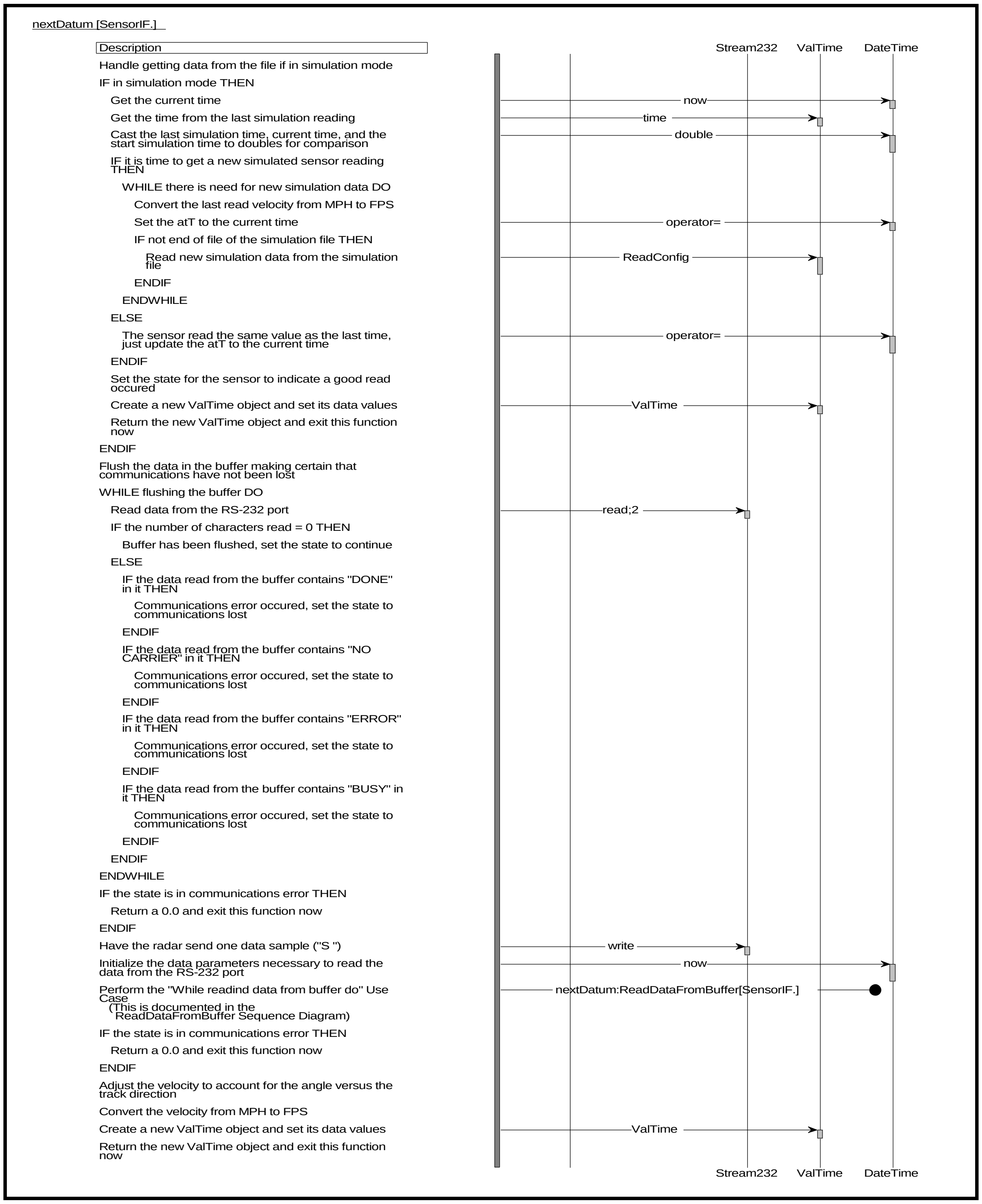


```
IF the timeout supplied is < 0.0 THEN
  Inform the error handler of the invalid timeout and exit the program
ENDIF
Flush out anything in the data buffer
Send the command string to the sensor ("AT")
Initialize the time variables needed to measure the timeout
Read the echo from the sensor
WHILE no data received AND the timeout has NOT occured DO
  Read data from the sensor
  IF data received from the sensor THEN
    Set the flag to exit the WHILE DO loop
  ELSE
    Get a new time to measure the timeout
  ENDIF
ENDWHILE
IF a timeout occured THEN
  Return a FALSE and exit this function now
ENDIF
IF there is a need to check for the OK from the sensor THEN
  WHILE no OK received from the sensor AND the timeout has NOT occured DO
    Read data from the sensor
    IF data received from the sensor THEN
      Set the flag to exit the WHILE DO loop
    ELSE
      Get a new time to measure the timeout
    ENDIF
  ENDWHILE
  IF a timeout occured THEN
    Return a FALSE and exit this function now
  ENDIF
ENDIF
All data has been received, return TRUE and exit this function now
```

Figure 58. SensorIF::modemCmd Sequence Diagram

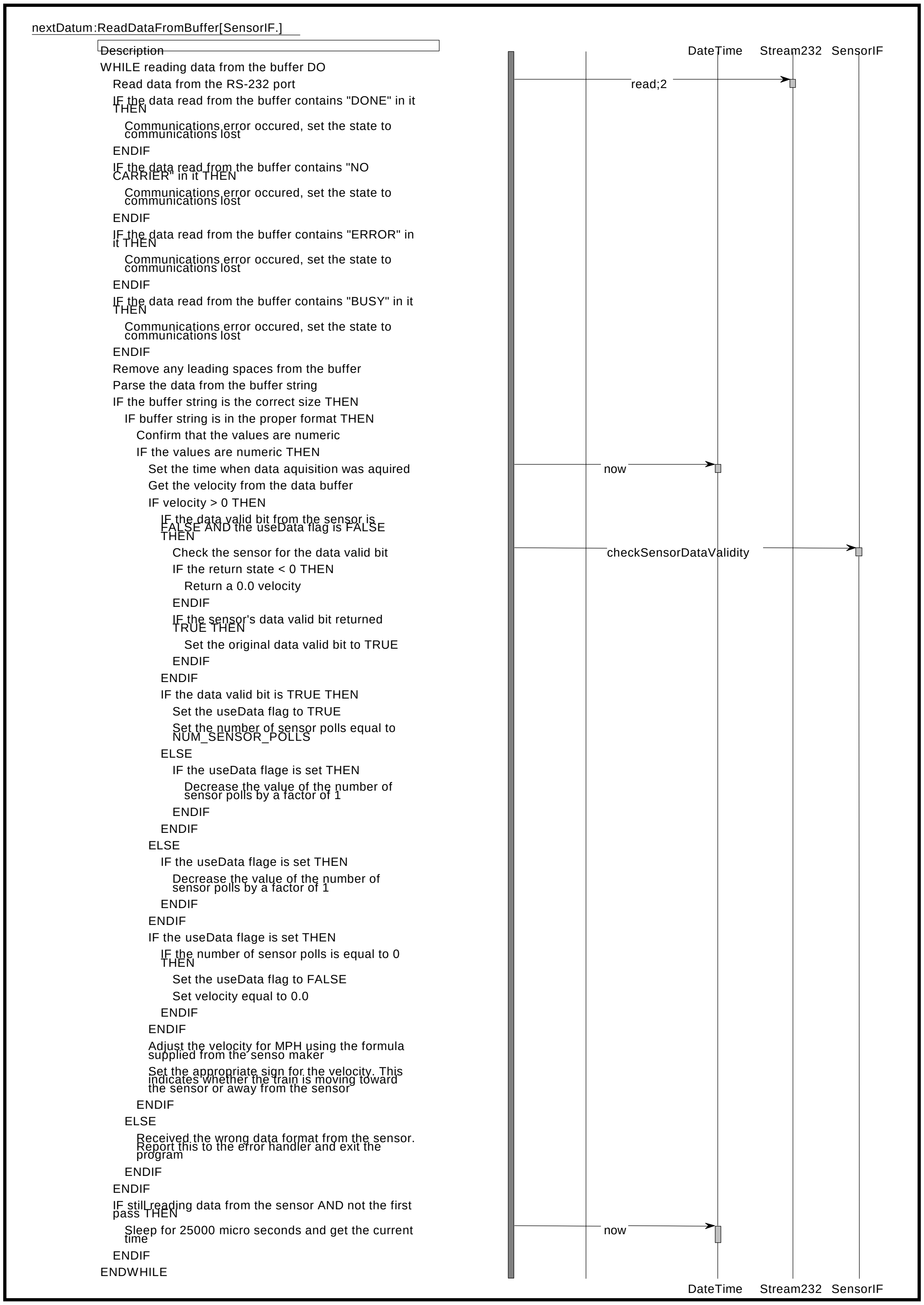
`NextDatum`

Tells the radar to send a single sample.
The sequence diagrams for this member function are given in Figure 59 and Figure 60.



```
Handle getting data from the file if in simulation mode
IF in simulation mode THEN
   Get the current time
   Get the time from the last simulation reading
   Cast the last simulation time, current time, and the
   start simulation time to doubles for comparison
   IF it is time to get a new simulated sensor reading
   THEN
      WHILE there is need for new simulation data DO
         Convert the last read velocity from MPH to FPS
         Set the atT to the current time
         IF not end of file of the simulation file THEN
            Read new simulation data from the simulation
            file
         ENDIF
      ENDWHILE
   ELSE
      The sensor read the same value as the last time,
      just update the atT to the current time
   ENDIF
   Set the state for the sensor to indicate a good read
   occured
   Create a new ValTime object and set its data values
   Return the new ValTime object and exit this function
   now
ENDIF
Flush the data in the buffer making certain that
communications have not been lost
WHILE flushing the buffer DO
   Read data from the RS-232 port
   IF the number of characters read = 0 THEN
      Buffer has been flushed, set the state to continue
   ELSE
      IF the data read from the buffer contains "DONE"
      in it THEN
         Communications error occured, set the state to
         communications lost
      ENDIF
      IF the data read from the buffer contains "NO
      CARRIER" in it THEN
         Communications error occured, set the state to
         communications lost
      ENDIF
      IF the data read from the buffer contains "ERROR"
      in it THEN
         Communications error occured, set the state to
         communications lost
      ENDIF
      IF the data read from the buffer contains "BUSY" in
      it THEN
         Communications error occured, set the state to
         communications lost
      ENDIF
   ENDIF
ENDWHILE
IF the state is in communications error THEN
   Return a 0.0 and exit this function now
ENDIF
Have the radar send one data sample ("S ")
Initialize the data parameters necessary to read the
data from the RS-232 port
Perform the "While readind data from buffer do" Use
Case
  (This is documented in the
   ReadDataFromBuffer Sequence Diagram)
IF the state is in communications error THEN
   Return a 0.0 and exit this function now
ENDIF
Adjust the velocity to account for the angle versus the
track direction
Convert the velocity from MPH to FPS
Create a new ValTime object and set its data values
Return the new ValTime object and exit this function
now
```

Figure 59. SensorIF::nextDatum Sequence Diagram



```
WHILE reading data from the buffer DO
   Read data from the RS-232 port
   IF the data read from the buffer contains "DONE" in it THEN
      Communications error occured, set the state to communications lost
   ENDIF
   IF the data read from the buffer contains "NO CARRIER" in it THEN
      Communications error occured, set the state to communications lost
   ENDIF
   IF the data read from the buffer contains "ERROR" in it THEN
      Communications error occured, set the state to communications lost
   ENDIF
   IF the data read from the buffer contains "BUSY" in it THEN
      Communications error occured, set the state to communications lost
   ENDIF
   Remove any leading spaces from the buffer
   Parse the data from the buffer string
   IF the buffer string is the correct size THEN
      IF buffer string is in the proper format THEN
         Confirm that the values are numeric
         IF the values are numeric THEN
            Set the time when data aquisition was aquired
            Get the velocity from the data buffer
            IF velocity > 0 THEN
               IF the data valid bit from the sensor is FALSE AND the useData flag is FALSE THEN
                  Check the sensor for the data valid bit
                  IF the return state < 0 THEN
                     Return a 0.0 velocity
                  ENDIF
                  IF the sensor's data valid bit returned TRUE THEN
                     Set the original data valid bit to TRUE
                  ENDIF
               ENDIF
               IF the data valid bit is TRUE THEN
                  Set the useData flag to TRUE
                  Set the number of sensor polls equal to NUM_SENSOR_POLLS
               ELSE
                  IF the useData flage is set THEN
                     Decrease the value of the number of sensor polls by a factor of 1
                  ENDIF
               ENDIF
            ELSE
               IF the useData flage is set THEN
                  Decrease the value of the number of sensor polls by a factor of 1
               ENDIF
            ENDIF
            IF the useData flage is set THEN
               IF the number of sensor polls is equal to 0 THEN
                  Set the useData flag to FALSE
                  Set velocity equal to 0.0
               ENDIF
            ENDIF
            Adjust the velocity for MPH using the formula supplied from the senso maker
            Set the appropriate sign for the velocity. This indicates whether the train is moving toward the sensor or away from the sensor
         ENDIF
      ELSE
         Received the wrong data format from the sensor. Report this to the error handler and exit the program
      ENDIF
   ENDIF
   IF still reading data from the sensor AND not the first pass THEN
      Sleep for 25000 micro seconds and get the current time
   ENDIF
ENDWHILE
```

Figure 60. SensorIF::nextDatum:ReadDataFromBuffer (Use Case) Sequence Diagram

`probeRadar`

Attempts to determine if the radar unit is working.
The sequence diagram for this member function is given in Figure 61.

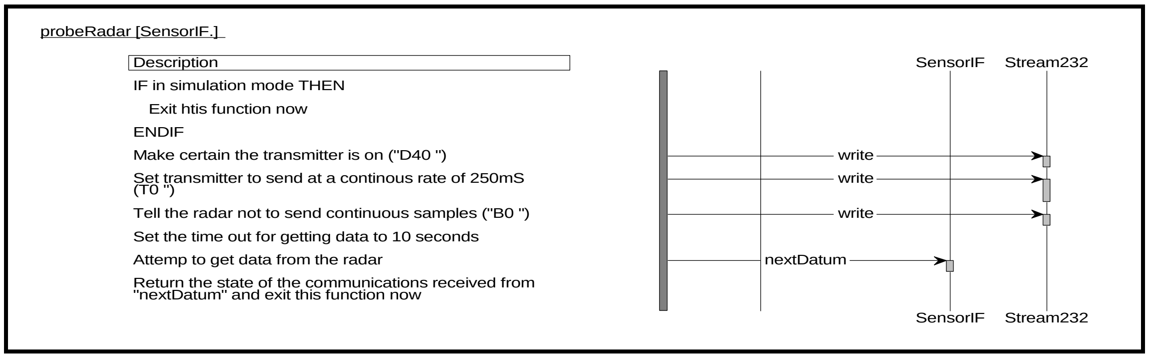
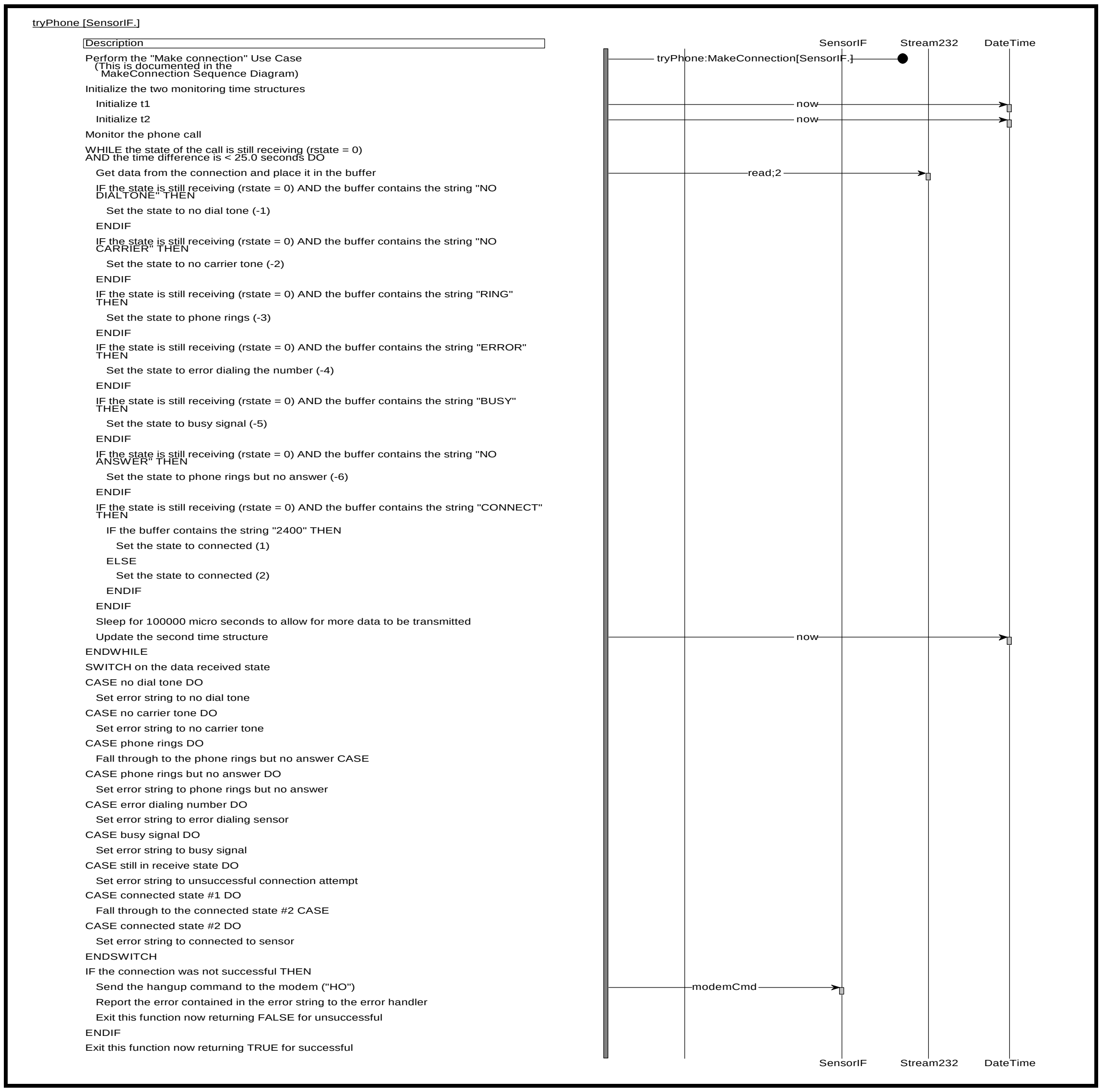

probeRadar [SensorIF.]
Description
IF in simulation mode THEN
Exit htis function now
ENDIF
Make certain the transmitter is on ("D40 ")
Set transmitter to send at a continous rate of 250mS (T0 ")
Tell the radar not to send continuous samples ("B0 ")
Set the time out for getting data to 10 seconds
Attemp to get data from the radar
Return the state of the communications received from "nextDatum" and exit this function now
SensorIF
Stream232
write
write
write
nextDatum
SensorIF
Stream232

Figure 61. SensorIF::probeRadar Sequence Diagram

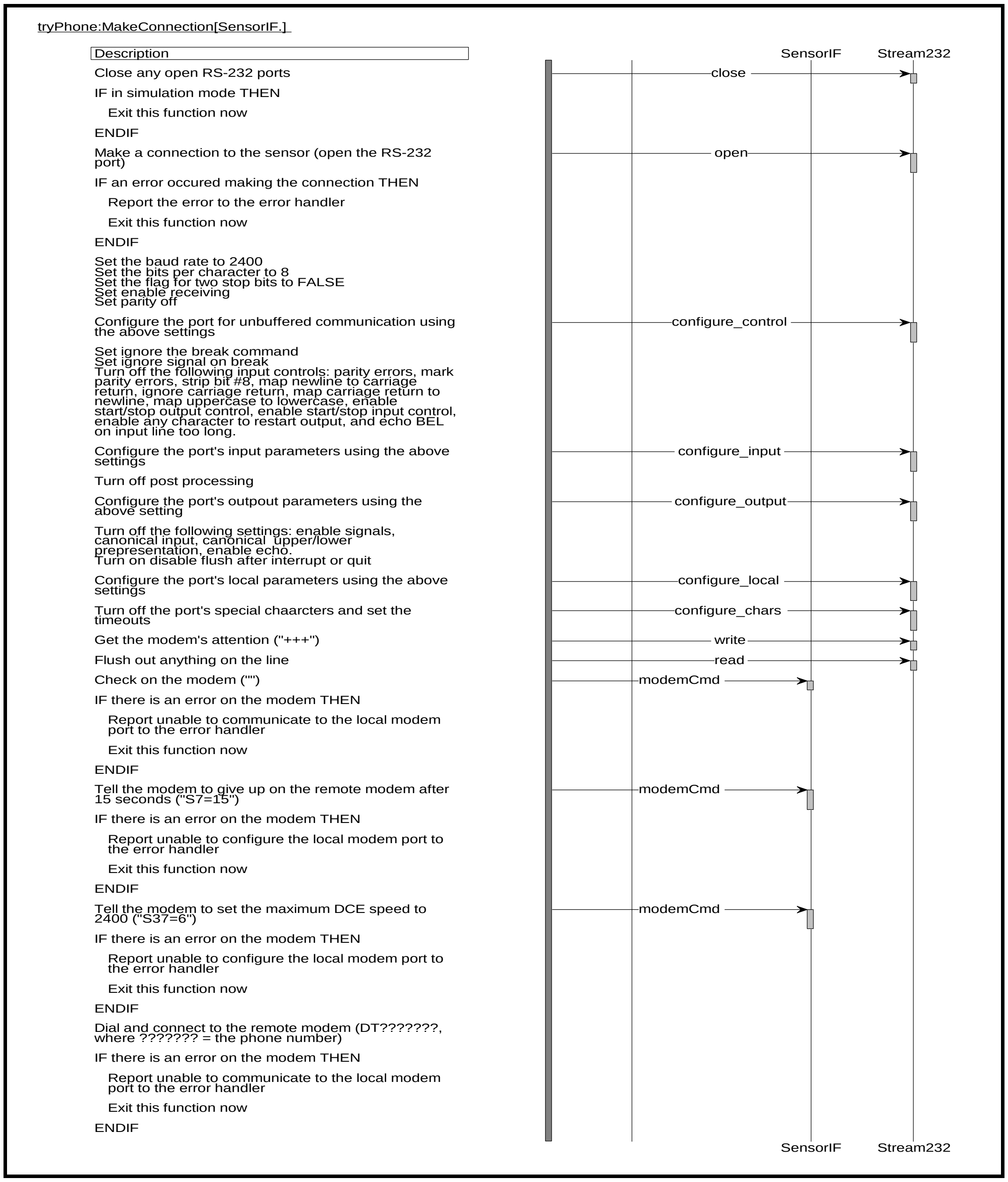
`tryPhone`

Attempts to establish a communication channel using the sunPort attribute value and then dialing the telephoneNum attribute value to the remote radar unit.
The sequence diagrams for this member function are in given Figure 62 and Figure 63.



```
Description
Perform the "Make connection" Use Case
  (This is documented in the
   MakeConnection Sequence Diagram)
Initialize the two monitoring time structures
   Initialize t1
   Initialize t2
Monitor the phone call
WHILE the state of the call is still receiving (rstate = 0)
AND the time difference is < 25.0 seconds DO
   Get data from the connection and place it in the buffer
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "NO
   DIALTONE" THEN
      Set the state to no dial tone (-1)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "NO
   CARRIER" THEN
      Set the state to no carrier tone (-2)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "RING"
   THEN
      Set the state to phone rings (-3)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "ERROR"
   THEN
      Set the state to error dialing the number (-4)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "BUSY"
   THEN
      Set the state to busy signal (-5)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "NO
   ANSWER" THEN
      Set the state to phone rings but no answer (-6)
   ENDIF
   IF the state is still receiving (rstate = 0) AND the buffer contains the string "CONNECT"
   THEN
      IF the buffer contains the string "2400" THEN
         Set the state to connected (1)
      ELSE
         Set the state to connected (2)
      ENDIF
   ENDIF
   Sleep for 100000 micro seconds to allow for more data to be transmitted
   Update the second time structure
ENDWHILE
SWITCH on the data received state
CASE no dial tone DO
   Set error string to no dial tone
CASE no carrier tone DO
   Set error string to no carrier tone
CASE phone rings DO
   Fall through to the phone rings but no answer CASE
CASE phone rings but no answer DO
   Set error string to phone rings but no answer
CASE error dialing number DO
   Set error string to error dialing sensor
CASE busy signal DO
   Set error string to busy signal
CASE still in receive state DO
   Set error string to unsuccessful connection attempt
CASE connected state #1 DO
   Fall through to the connected state #2 CASE
CASE connected state #2 DO
   Set error string to connected to sensor
ENDSWITCH
IF the connection was not successful THEN
   Send the hangup command to the modem ("HO")
   Report the error contained in the error string to the error handler
   Exit this function now returning FALSE for unsuccessful
ENDIF
Exit this function now returning TRUE for successful
```

Figure 62. SensorIF::tryPhone Sequence Diagram



```
Description
Close any open RS-232 ports
IF in simulation mode THEN
   Exit this function now
ENDIF
Make a connection to the sensor (open the RS-232
port)
IF an error occured making the connection THEN
   Report the error to the error handler
   Exit this function now
ENDIF
Set the baud rate to 2400
Set the bits per character to 8
Set the flag for two stop bits to FALSE
Set enable receiving
Set parity off
Configure the port for unbuffered communication using
the above settings
Set ignore the break command
Set ignore signal on break
Turn off the following input controls: parity errors, mark
parity errors, strip bit #8, map newline to carriage
return, ignore carriage return, map carriage return to
newline, map uppercase to lowercase, enable
start/stop output control, enable start/stop input control,
enable any character to restart output, and echo BEL
on input line too long.
Configure the port's input parameters using the above
settings
Turn off post processing
Configure the port's outpout parameters using the
above setting
Turn off the following settings: enable signals,
canonical input, canonical  upper/lower
prepresentation, enable echo.
Turn on disable flush after interrupt or quit
Configure the port's local parameters using the above
settings
Turn off the port's special chaarcters and set the
timeouts
Get the modem's attention ("+++")
Flush out anything on the line
Check on the modem ("")
IF there is an error on the modem THEN
   Report unable to communicate to the local modem
   port to the error handler
   Exit this function now
ENDIF
Tell the modem to give up on the remote modem after
15 seconds ("S7=15")
IF there is an error on the modem THEN
   Report unable to configure the local modem port to
   the error handler
   Exit this function now
ENDIF
Tell the modem to set the maximum DCE speed to
2400 ("S37=6")
IF there is an error on the modem THEN
   Report unable to configure the local modem port to
   the error handler
   Exit this function now
ENDIF
Dial and connect to the remote modem (DT???????,
where ??????? = the phone number)
IF there is an error on the modem THEN
   Report unable to communicate to the local modem
   port to the error handler
   Exit this function now
ENDIF
```

Figure 63. SensorIF::tryPhone:MakeConnection (Use Case) Sequence Diagram

5.4.2.3 RS-232 Port (Stream232) Class

This class is used for dealing with RS232 I/O. This class allows for a file descriptor supporting both reading and writing. This class cannot be copied.

The Stream232 class contains the following attributes:

PUBLIC:

fd	File descriptor to the Sun port, non-negative means a valid descriptor.

PRIVATE:

maxlen	The maximum number of characters to read from the port.
s*buf*	The character read buffer.
sbuflen	The length of sbuf.

PROTECTED:

tio	The structure for the Terminal I/O characteristics.

The Stream232 class contains the following operations:

PUBLIC:

`Stream232`	Default constructor.
`close`	Standard file closure.
`configure_chars`	Configure the character processing control parameters of the RS 232 port.
`configure_input`	Configure the input control parameters of the RS 232 port.
`configure_local`	Configure the local control parameters of the RS 232 port.
`configure_output`	Configure the output control parameters of the RS 232 port.
`open`	The routine opens the device for reading and writing, then confirms that ioctl system calls can be made on the device using the TCGETA flag.
`read`	Read from the RS 232 port using the current settings. The system will read until some sort of end of line condition is reached or timeout condition occurs (depending on the current settings) or the maxlen is read.

`read;2`	Overloaded read function that reads from the RS 232 port and performs pseudo-line processing.
`write`	Writes to the RS 232 port using the current settings.
`~Stream232`	Class destructor.

5.4.2.4 Value Time Stamp (ValTime) Class

This class is responsible for containing a value and an associated timestamp. This class should be able to be used in an ordered list because it has an equality operator, a copy constructor, an assignment operator, and a default constructor.

The ValTime class contains the following attributes:

PRIVATE:

stamp	The timestamp associated with the value.
val	The value.

The ValTime class contains the following operations:

PUBLIC:

`ReadConfig`	Read original configuration information.
`ValTime`	Default constructor.
`ValTime;2`	Constructor that takes a ValTime instance as an initializer.
`WriteConfig`	Writes actual or example configuration data to a file.
`newValue`	Updates the object with new values.
`operator=`	Overloaded assignment operator to assign one ValTime object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`time`	Returns the timestamp associated with the value.
`value`	Returns the value.
`~ValTime`	Destructor.

5.4.2.5 Sensor Status (senseStatus) Class

This class is used to keep track if a virtual sensor is up or down.

The senseStatus class contains the following attributes:

PRIVATE:

id	The virtual sensor id, s1..sM, it is associated with.
working	Boolean flag that states whether the virtual sensor is working.

The senseStatus class contains the following operations:

PUBLIC:

`isUp`	Returns the Boolean flag that states whether the virtual sensor is working or not working.
`operator=`	Overloaded assignment operator to assign one senseStatus object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical. This operator accepts a string to compare against the *id*.
`operator==;2`	Overloaded equality operator used to detect that two instances are identical. This operator accepts another senseStatus instance compare against.
`senseStatus`	The default constructor.
`senseStatus;2`	Constructor where the values must be passed in.
`senseStatus;3`	Constructor that takes a senseStatus instance as an initializer.
`setUpDown`	Accepts TRUE or FALSE to set the virtual sensor up or down.
`setUpDown;2`	Accepts a string and a TRUE or FALSE to set the virtual sensor up or down. Only sets the value if string passed matches the *id*. Then it returns TRUE, otherwise it returns FALSE.
`setvId`	Sets the id for the object.
`vId`	Returns the id of the object.

5.4.2.6 Virtual Sensor (VSensor) Class

This class is designed to make configuration of AWARD to be straightforward. Specifically, each radar sensor provides data for 2 virtual sensors. One for trains receding from the sensor, the other for approaching trains. This class maintains a downstream list of crossings and other sensors and a summary of any train the sensor is perceiving. After being provided current information from the sensor, it creates or updates the train summary. This class is like the SensorIF class in that it implements a finite state machine.

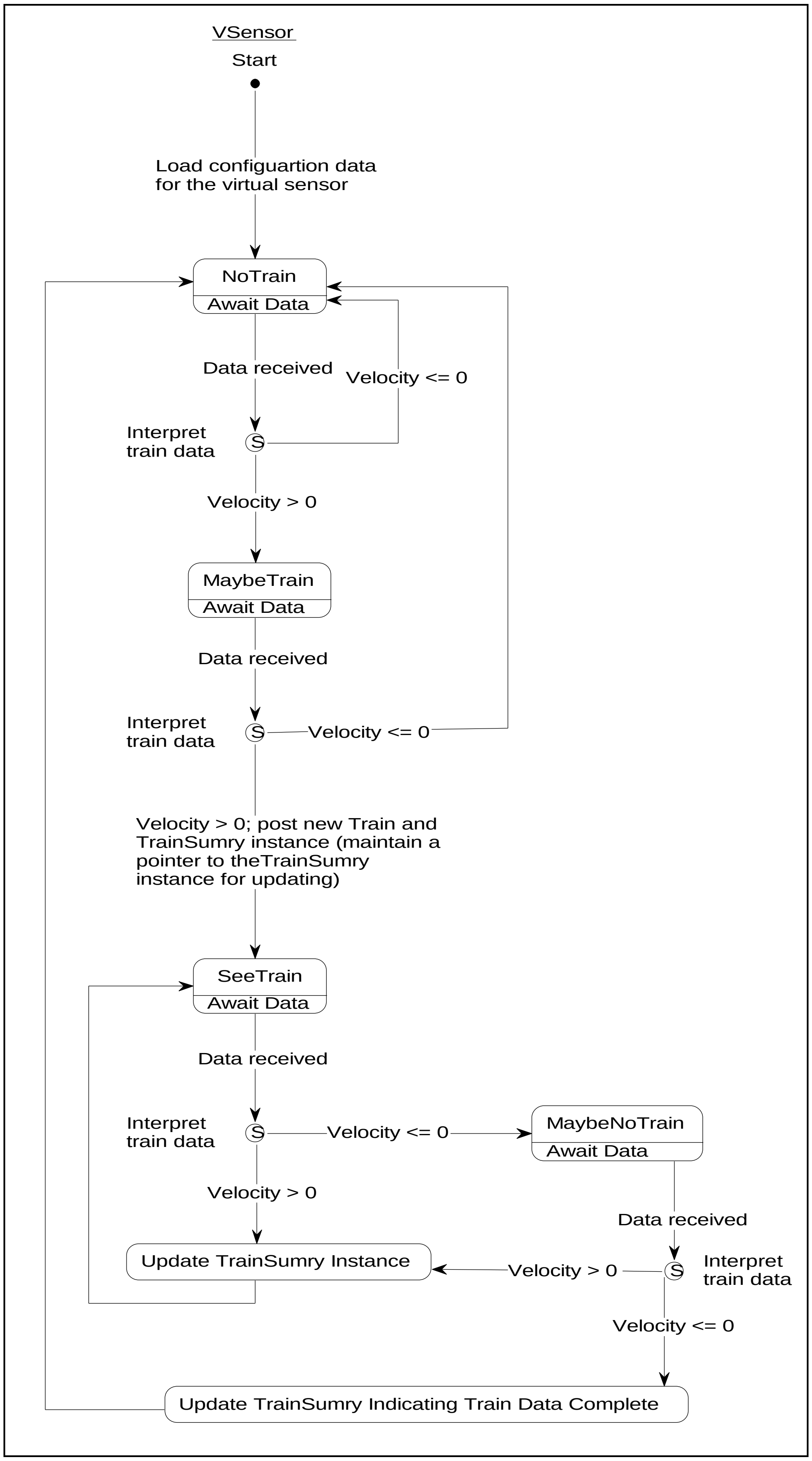


Figure 64. Virtual Sensor (Vsensor) Class State Diagram

The VSensor class contains the following attributes:

PRIVATE:

connects	List of downstream connections from the virtual sensor.
directed	Which direction, with respect to sensor, belongs to this virtual sensor.
id	Virtual sensor identifier, s1..sM.
lastTime	Time of previous update.
lastVel	Value of previous velocity.
name	Name of this sensor.
rules	List of acceleration modification rules.
sensorId	Identifier of associate actual sensor, a1..aN.
state	Current state of the class, with respect to the finite state machine.
sumry	The train summary pointer.

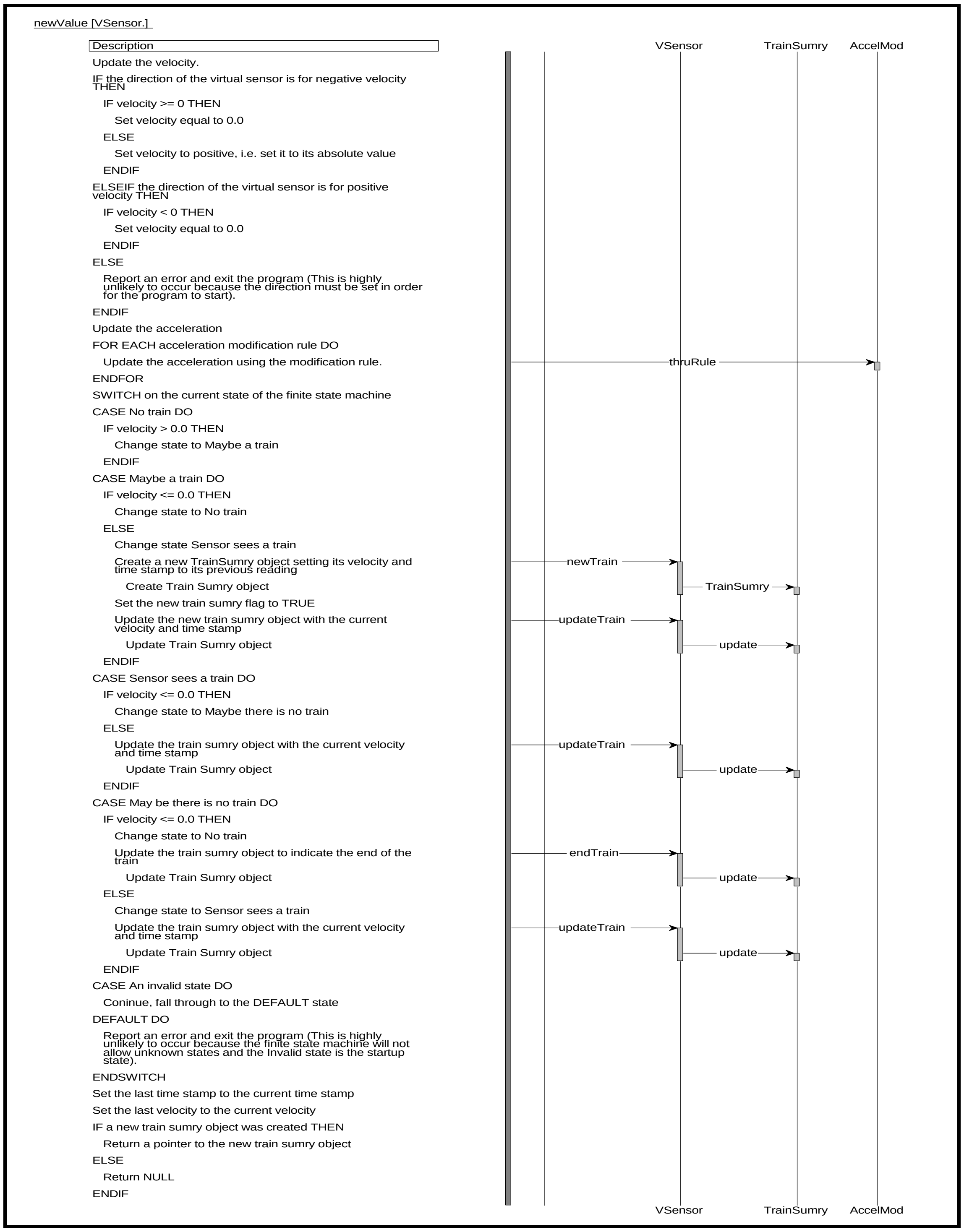
The VSensor class contains the following operations:

PUBLIC:

`connections`	Access routine that returns the list of downstream connections from the virtual sensor.
`direction`	Access routine that returns the sensor direction.
`identifier`	Access routine that returns the virtual sensor ID.

`newValue`

Given current sensor values of velocity and acceleration, the data applies to this virtual sensor, then update the train summary.
The sequence diagram for this member function is given in Figure 65.



newValue [VSensor.]

Description

```
Update the velocity.
IF the direction of the virtual sensor is for negative velocity THEN
   IF velocity >= 0 THEN
      Set velocity equal to 0.0
   ELSE
      Set velocity to positive, i.e. set it to its absolute value
   ENDIF
ELSEIF the direction of the virtual sensor is for positive velocity THEN
   IF velocity < 0 THEN
      Set velocity equal to 0.0
   ENDIF
ELSE
   Report an error and exit the program (This is highly unlikely to occur because the direction must be set in order for the program to start).
ENDIF
Update the acceleration
FOR EACH acceleration modification rule DO
   Update the acceleration using the modification rule.
ENDFOR
SWITCH on the current state of the finite state machine
CASE No train DO
   IF velocity > 0.0 THEN
      Change state to Maybe a train
   ENDIF
CASE Maybe a train DO
   IF velocity <= 0.0 THEN
      Change state to No train
   ELSE
      Change state Sensor sees a train
      Create a new TrainSumry object setting its velocity and time stamp to its previous reading
         Create Train Sumry object
      Set the new train sumry flag to TRUE
      Update the new train sumry object with the current velocity and time stamp
         Update Train Sumry object
   ENDIF
CASE Sensor sees a train DO
   IF velocity <= 0.0 THEN
      Change state to Maybe there is no train
   ELSE
      Update the train sumry object with the current velocity and time stamp
         Update Train Sumry object
   ENDIF
CASE May be there is no train DO
   IF velocity <= 0.0 THEN
      Change state to No train
      Update the train sumry object to indicate the end of the train
         Update Train Sumry object
   ELSE
      Change state to Sensor sees a train
      Update the train sumry object with the current velocity and time stamp
         Update Train Sumry object
   ENDIF
CASE An invalid state DO
   Coninue, fall through to the DEFAULT state
DEFAULT DO
   Report an error and exit the program (This is highly unlikely to occur because the finite state machine will not allow unknown states and the Invalid state is the startup state).
ENDSWITCH
Set the last time stamp to the current time stamp
Set the last velocity to the current velocity
IF a new train sumry object was created THEN
   Return a pointer to the new train sumry object
ELSE
   Return NULL
ENDIF
```

Figure 65. VSensor::newValue Sequence Diagram

`operator=`	Overloaded assignment operator to assign one virtual sensor object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`sensorIdentifier`	Access routine that returns the actual sensor ID.
`summary`	Access routine that returns the TrainSumry pointer.
`vName`	Access routine that returns the virtual sensor name.
`VSensor`	Default constructor.
`VSensor;2`	Constructor that takes a configuration file name, a list of downstream connections, and a list of acceleration modifications as its initializers.
`VSensor;3`	Constructor that takes a Virtual Sensor instance as an initializer.
`~VSensor`	Destructor.

PRIVATE:

`endTrain`	Sets the end of the train through the variables.
`newTrain`	Creates a new TrainSumry object.
`ReadConfig`	Reads and loads the object with data from its configuration file.
`updateTrain`	Updates the TrainSumry object associated with this virtual sensor.
`WriteConfig`	Write actual or example configuration data to a file.

5.4.2.7 Acceleration Modification (AccelMod) Class

This class contains acceleration modification rules. These rules are a way for known behavior to be entered into the AWARD system, especially train behavior that occurs when trains are not being observed by a sensor. Acceleration may be modified by setting the acceleration to some constant or adjusting the measured acceleration. Adjustment may include adding and/or multiplying by some constants. This class should be able to be used in an ordered list because it has an equality operator, a copy constructor, an assignment operator, and a default constructor.

The AccelMod class contains the following attributes:

PRIVATE:

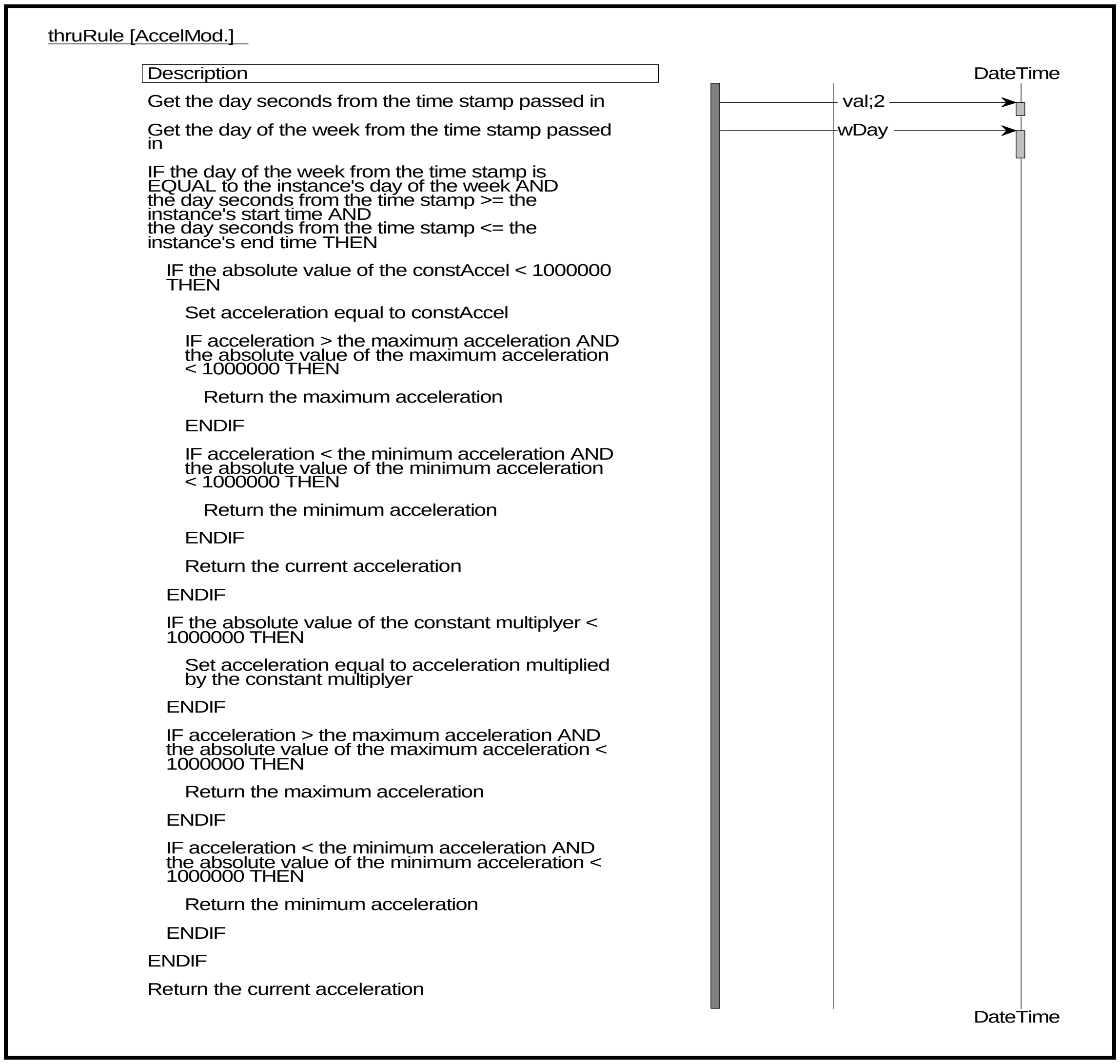
constAccel	Constant value to set acceleration to (or infinity).
constMult	Constant multiplicand to apply to acceleration (after the offset).
constOffset	The constant offset to apply.
dayOfWeek	The day of week, 1..7, rule to apply, Sunday is 1.
endTime	The time of day the rule stops applying.
maxA	The maximum (absolute) acceleration after calculations.
minA	The minimum (absolute) acceleration after calculations.
startTime	The time of day the rule begins to apply.
vSensorId	The associated virtual sensor ID, s1..sM.

The AccelMod class contains the following operations:

PUBLIC:

`AccelMod`	Default constructor. Note: The default constructor values are not really meaningful. So the default constructor just provides a placeholder. Assigning data requires using the other constructors.
`AccelMod;2`	Constructor that loads in configuration data from a supplied data file.
`AccelMod;3`	Constructor that takes a AccelMod instance as an initializer.
`ReadConfig`	Reads original configuration information.
`WriteConfig`	Writes actual or example configuration data to a file.
`compare`	Detects that another instance has overlapping conditions. Returns TRUE if this instance and the other instance overlap.
`operator=`	Overloaded assignment operator to assign one AccelMod object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.

`thruRule`	Applies the acceleration modification rule to the provided value, returning a (potentially) modified value. The sequence diagram for this member function is given in Figure 66.



```
Description
Get the day seconds from the time stamp passed in
Get the day of the week from the time stamp passed in
IF the day of the week from the time stamp is EQUAL to the instance's day of the week AND
the day seconds from the time stamp >= the instance's start time AND
the day seconds from the time stamp <= the instance's end time THEN
    IF the absolute value of the constAccel < 1000000 THEN
        Set acceleration equal to constAccel
        IF acceleration > the maximum acceleration AND the absolute value of the maximum acceleration < 1000000 THEN
            Return the maximum acceleration
        ENDIF
        IF acceleration < the minimum acceleration AND the absolute value of the minimum acceleration < 1000000 THEN
            Return the minimum acceleration
        ENDIF
        Return the current acceleration
    ENDIF
    IF the absolute value of the constant multiplyer < 1000000 THEN
        Set acceleration equal to acceleration multiplied by the constant multiplyer
    ENDIF
    IF acceleration > the maximum acceleration AND the absolute value of the maximum acceleration < 1000000 THEN
        Return the maximum acceleration
    ENDIF
    IF acceleration < the minimum acceleration AND the absolute value of the minimum acceleration < 1000000 THEN
        Return the minimum acceleration
    ENDIF
ENDIF
Return the current acceleration
```

Figure 66. AccelMod::thruRule Sequence Diagram

`vSensorID`	Returns the associated virtual sensor ID.
`~AccelMod`	Destructor.

5.4.2.8 Crossing (Crossing) Class

This class is responsible for containing information about one railroad crossing. The data contained in the class is simply a description of the distance between the crossing and where the signals detect a train (under normal circumstances). The class also has a textual name of the railroad crossing.

The Crossing class contains the following attributes:

PRIVATE:

id	The crossing item identifier, c1..cP.
majorAlarmTime	Time when a major alarm was sent to ATMS.
name	The textual name of the crossing.
nominalTime	The nominal time from the train crossing to where the train is sensed (0 if no signals at the crossing).

The Crossing class contains the following operations:

PUBLIC:

`Crossing`	Default constructor. Note: The default constructor is not really useful. It is just a placeholder because assigning data requires using the other constructors.
`Crossing;2`	Constructor that loads in configuration data from a supplied data file.
`Crossing;3`	Constructor that takes a Crossing instance as an initializer.
`ID`	Returns the crossing ID.
`crossingName`	Returns textual crossing name.
`majAlarmTime`	Returns the time when a major alarm was sent to ATMS.
`nomTime`	Returns the nominal time from the train crossing to where the train is sensed.
`operator=`	Overloaded assignment operator to assign one crossing object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.

`setMajorAlarmTime`	Sets the time when a major alarm was sent to ATMS.
`~Crossing`	Destructor.

PRIVATE:

`ReadConfig`	Reads original configuration information.
`WriteConfig`	Writes actual or example configuration data to a file.

5.4.2.9 Train Class

This class is responsible for maintaining (a unified summary of) a train and a list of Estimated Time of Arrival (ETA) and Estimated Time unTil Arrival (ETTA) for all downstream connections. There may be more than one sensor that perceives a particular train, so there may be multiple train summaries that a train instance includes. Part of interfacing to other classes includes the ability to manipulate train summaries without ever copying them so that the Virtual Sensors may be able to update the summaries irrespective of this class's activities.

The Train class contains the following attributes:

PRIVATE:

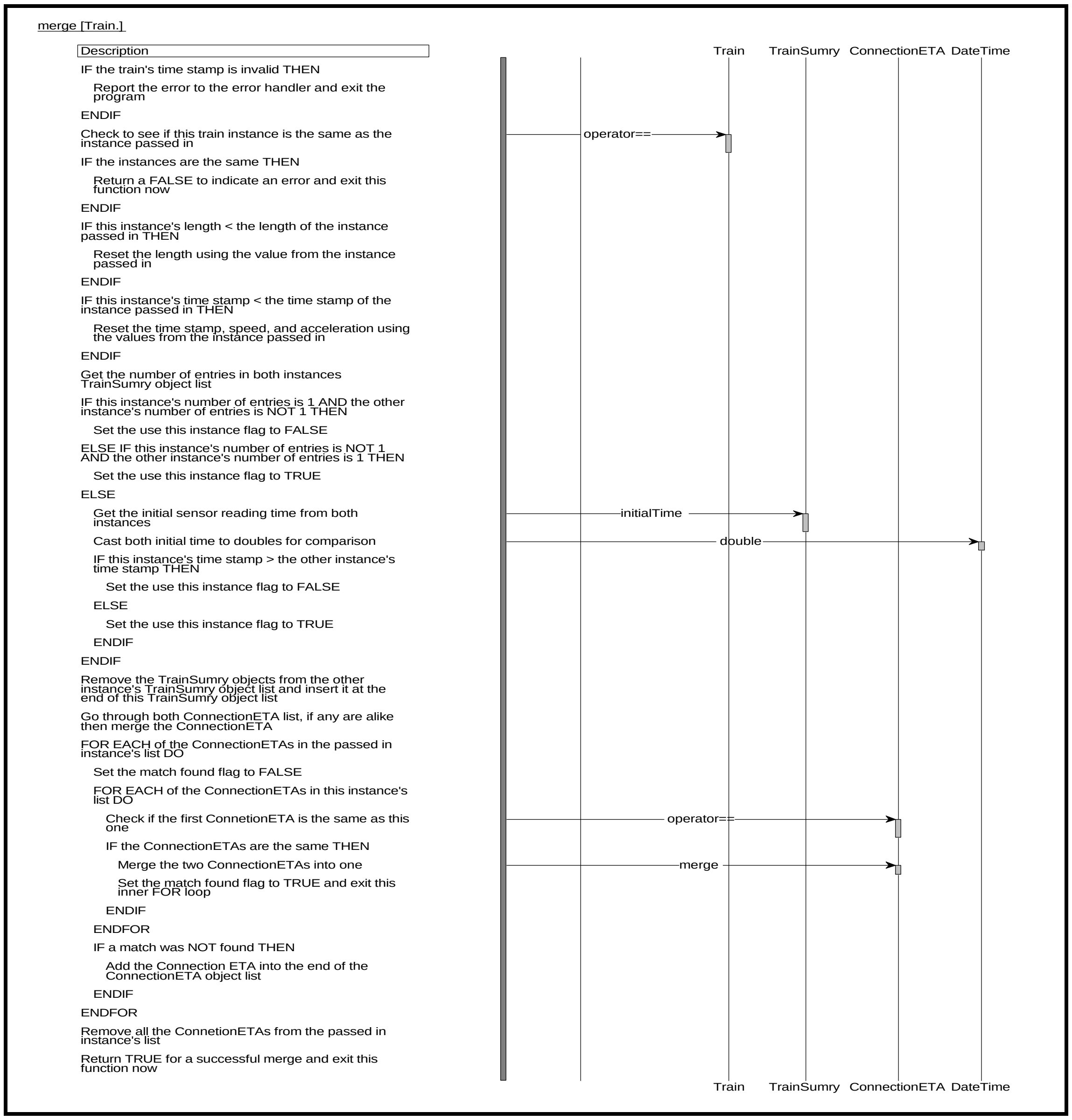
acceleration	The train’s acceleration.
atTime	The timestamp when speed, acceleration., and length last updated.
eta	List of train ETAs and ETTAs for all downstream connections. Downstream connections maybe crossings or virtual sensors.
length	The train’s length.
speed	The train’s speed.
sumries	List of train summary (TrainSumry object) pointers. The train summary structure contains information regarding one train that is seen by one (virtual) sensor.

The Train class contains the following operations:

PUBLIC:

`cancelAlarm`	Cancel an alarm sent to ATMS.

`crossingExists`	Determines if a crossing is in the train's connection list.
`currentAccl`	Return the train's acceleration.
`currentLength`	Return the train's length.
`currentSpeed`	Return the train's speed.
`currentTime`	Return the train's timestamp.
`majorAlarmSent`	Determines if a train sent a major alarm to ATMS.
`merge`	Determining that two trains refer to the same actual train and merge them (without interrupting train summary memory locations) - the second train looses it's summaries, but still needs to be discarded after return (only discard if merge returns true). The sequence diagram for this member function is given in Figure 67.

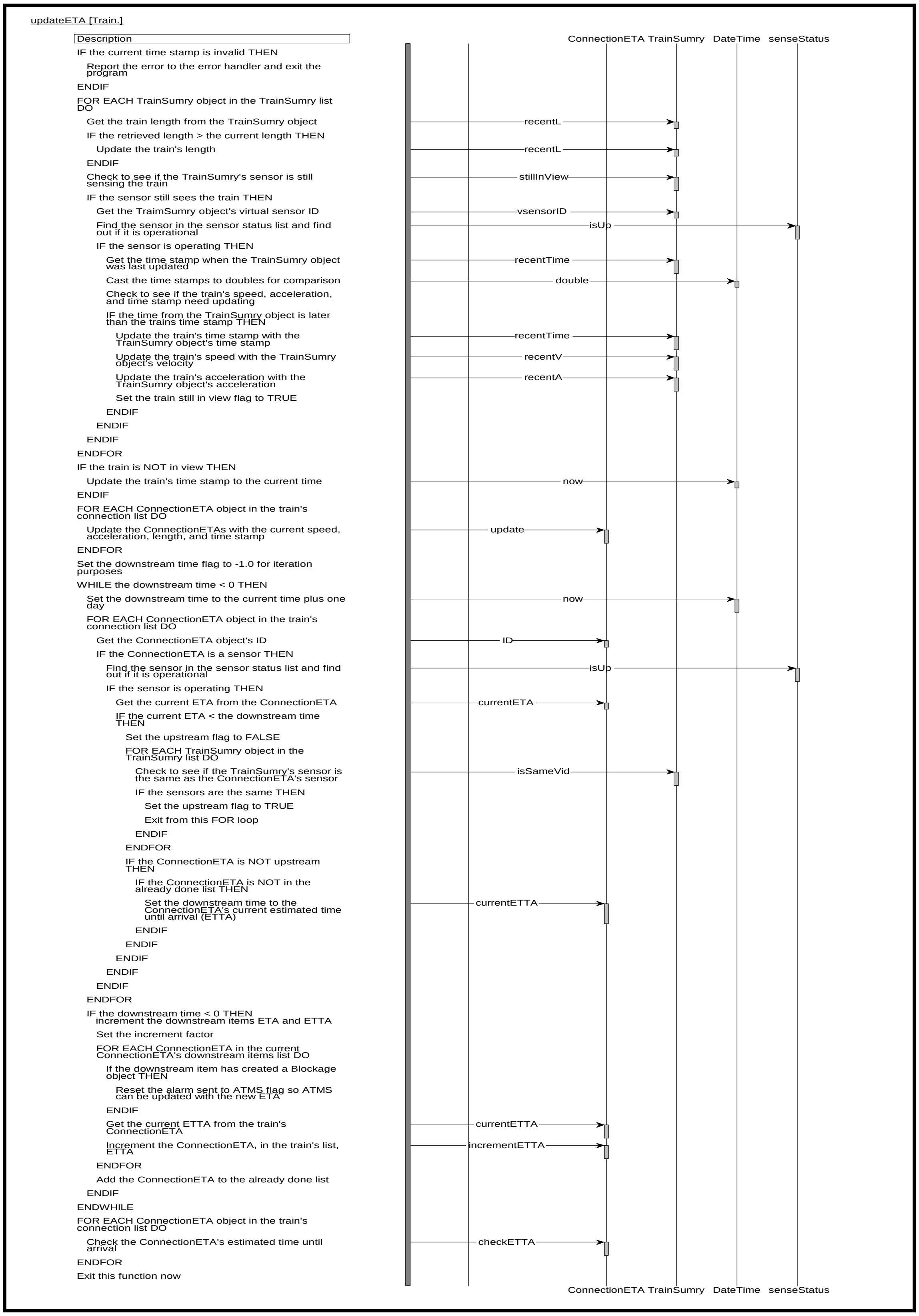


```
Description
IF the train's time stamp is invalid THEN
   Report the error to the error handler and exit the program
ENDIF
Check to see if this train instance is the same as the instance passed in
IF the instances are the same THEN
   Return a FALSE to indicate an error and exit this function now
ENDIF
IF this instance's length < the length of the instance passed in THEN
   Reset the length using the value from the instance passed in
ENDIF
IF this instance's time stamp < the time stamp of the instance passed in THEN
   Reset the time stamp, speed, and acceleration using the values from the instance passed in
ENDIF
Get the number of entries in both instances TrainSumry object list
IF this instance's number of entries is 1 AND the other instance's number of entries is NOT 1 THEN
   Set the use this instance flag to FALSE
ELSE IF this instance's number of entries is NOT 1 AND the other instance's number of entries is 1 THEN
   Set the use this instance flag to TRUE
ELSE
   Get the initial sensor reading time from both instances
   Cast both initial time to doubles for comparison
   IF this instance's time stamp > the other instance's time stamp THEN
      Set the use this instance flag to FALSE
   ELSE
      Set the use this instance flag to TRUE
   ENDIF
ENDIF
Remove the TrainSumry objects from the other instance's TrainSumry object list and insert it at the end of this TrainSumry object list
Go through both ConnectionETA list, if any are alike then merge the ConnectionETA
FOR EACH of the ConnectionETAs in the passed in instance's list DO
   Set the match found flag to FALSE
   FOR EACH of the ConnectionETAs in this instance's list DO
      Check if the first ConnetionETA is the same as this one
      IF the ConnectionETAs are the same THEN
         Merge the two ConnectionETAs into one
         Set the match found flag to TRUE and exit this inner FOR loop
      ENDIF
   ENDFOR
   IF a match was NOT found THEN
      Add the Connection ETA into the end of the ConnectionETA object list
   ENDIF
ENDFOR
Remove all the ConnetionETAs from the passed in instance's list
Return TRUE for a successful merge and exit this function now
```

Figure 67. Train::merge Sequence Diagram

`operator=`	Overloaded assignment operator to assign one train equal to another.
`operator==`	Overloaded equality operator to determine that two train instances are identical.
`pastAll`	Determine that a train has left all areas of interest.
`Train`	Default constructor. Note: the default constructor generates a runtime error if it is used without an initializer.

`Train;2`	Constructor that takes a Train summary pointer as an initializer.
`Train;3`	Constructor that takes a Train object as an initializer.
`updateETA`	Update internal data, list of ConnectionETA objects for all downstream connections and generate appropriate events. The sequence diagram for this member function is given in Figure 68.



```
Description
IF the current time stamp is invalid THEN
   Report the error to the error handler and exit the program
ENDIF
FOR EACH TrainSumry object in the TrainSumry list DO
   Get the train length from the TrainSumry object
   IF the retrieved length > the current length THEN
      Update the train's length
   ENDIF
   Check to see if the TrainSumry's sensor is still sensing the train
   IF the sensor still sees the train THEN
      Get the TraimSumry object's virtual sensor ID
      Find the sensor in the sensor status list and find out if it is operational
      IF the sensor is operating THEN
         Get the time stamp when the TrainSumry object was last updated
         Cast the time stamps to doubles for comparison
         Check to see if the train's speed, acceleration, and time stamp need updating
         IF the time from the TrainSumry object is later than the trains time stamp THEN
            Update the train's time stamp with the TrainSumry object's time stamp
            Update the train's speed with the TrainSumry object's velocity
            Update the train's acceleration with the TrainSumry object's acceleration
            Set the train still in view flag to TRUE
         ENDIF
      ENDIF
   ENDIF
ENDFOR
IF the train is NOT in view THEN
   Update the train's time stamp to the current time
ENDIF
FOR EACH ConnectionETA object in the train's connection list DO
   Update the ConnectionETAs with the current speed, acceleration, length, and time stamp
ENDFOR
Set the downstream time flag to -1.0 for iteration purposes
WHILE the downstream time < 0 THEN
   Set the downstream time to the current time plus one day
   FOR EACH ConnectionETA object in the train's connection list DO
      Get the ConnectionETA object's ID
      IF the ConnectionETA is a sensor THEN
         Find the sensor in the sensor status list and find out if it is operational
         IF the sensor is operating THEN
            Get the current ETA from the ConnectionETA
            IF the current ETA < the downstream time THEN
               Set the upstream flag to FALSE
               FOR EACH TrainSumry object in the TrainSumry list DO
                  Check to see if the TrainSumry's sensor is the same as the ConnectionETA's sensor
                  IF the sensors are the same THEN
                     Set the upstream flag to TRUE
                     Exit from this FOR loop
                  ENDIF
               ENDFOR
               IF the ConnectionETA is NOT upstream THEN
                  IF the ConnectionETA is NOT in the already done list THEN
                     Set the downstream time to the ConnectionETA's current estimated time until arrival (ETTA)
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDFOR
   IF the downstream time < 0 THEN
      increment the downstream items ETA and ETTA
      Set the increment factor
      FOR EACH ConnectionETA in the current ConnectionETA's downstream items list DO
         If the downstream item has created a Blockage object THEN
            Reset the alarm sent to ATMS flag so ATMS can be updated with the new ETA
         ENDIF
         Get the current ETTA from the train's ConnectionETA
         Increment the ConnectionETA, in the train's list, ETTA
      ENDFOR
      Add the ConnectionETA to the already done list
   ENDIF
ENDWHILE
FOR EACH ConnectionETA object in the train's connection list DO
   Check the ConnectionETA's estimated time until arrival
ENDFOR
Exit this function now
```

Figure 68. Train::updateETA Sequence Diagram

`~Train`	Destructor

5.4.2.10 Train Summary (TrainSumry) Class

This class is responsible for containing information regarding one train that is seen by one (virtual) sensor. Only the virtual sensor is expected to update the data contained herein, however the data is expected to be used by the Train class and is expected to handle deleting instances of TrainSumry objects.

The TrainSumry class contains the following attributes:

PRIVATE:

connects	List of downstream connections from the virtual sensor.
currAcc	Current train acceleration.
currSpd	Current train speed.
currTime	Time of current values.
estLngth	Current estimate of the length.
initSpd	Initial train speed.
initTime	Time of initial train speed.
inView	If true, sensor is still perceiving the train, otherwise it has completed its passing. This also indicates that the virtual sensor, which created this summary, is no longer referencing this instance.
vId	Virtual sensor ID that created this train summary.

The TrainSumry class contains the following operations:

PUBLIC:

`connections`	Access routine to the list of downstream connections from the virtual sensor.
`initialA`	Access routine that returns the initial acceleration. Always just returns 0.
`initialTime`	Access routine that returns the initial time stamp.
`initialV`	Access routine that returns the initial speed.

`isSameVid`	Returns TRUE if the vid passed in is the same as the instance *vId*.
`operator=`	Overloaded assignment operator to assign one TrainSumry object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`recentA`	Access routine to the most recent acceleration.
`recentL`	Access routine to the most recent length.
`recentTime`	Access routine to the most recent time stamp.
`recentV`	Access routine to the most recent speed.
`stillInView`	Returns TRUE if the virtual sensor still detects a train.
`TrainSumry`	Default constructor. Note: the default constructor generates a runtime error if it is used without a vel (must be 0 or greater) and atTime set.
`TrainSumry;2`	Constructor that takes a TrainSumry instance as an initializer.
`update`	Update internal data using new sensor information if stillInView is false, then vel and acc are ignored. vel must be greater than 0.
`vsensorID`	Access routine to the virtual sensor ID that created the instance.
`~TrainSumry`	Destructor.

5.4.2.11 Connection Description (ConnectionETA) Class

This class, part of the AWARD system, is responsible for containing information relating one virtual sensor to all of its downstream components (include virtual sensors and crossings). Part of the data contained in the class is simply a description of the distance between the sensor and the downstream item and the maximum velocity the train will ever travel between the two locations. The remaining information includes ETA and ETTA for a particular train. Creation and update of this data is the responsibility of class Train. This class generates crossing blockages (information passed to MDI DataServer) and ATMS alarm/incidents based on event scenarios for the crossing. This class assumes that whenever two different lists of ConnectionETA are merged, that the merge function is called prior to any attempts to update information. Otherwise, some data may become corrupted because some update functions and crossing Blockage logic requires that only one crossing Blockage may be associated with a particular downstream item.

The ConnectionETA class contains the following attributes:

PRIVATE:

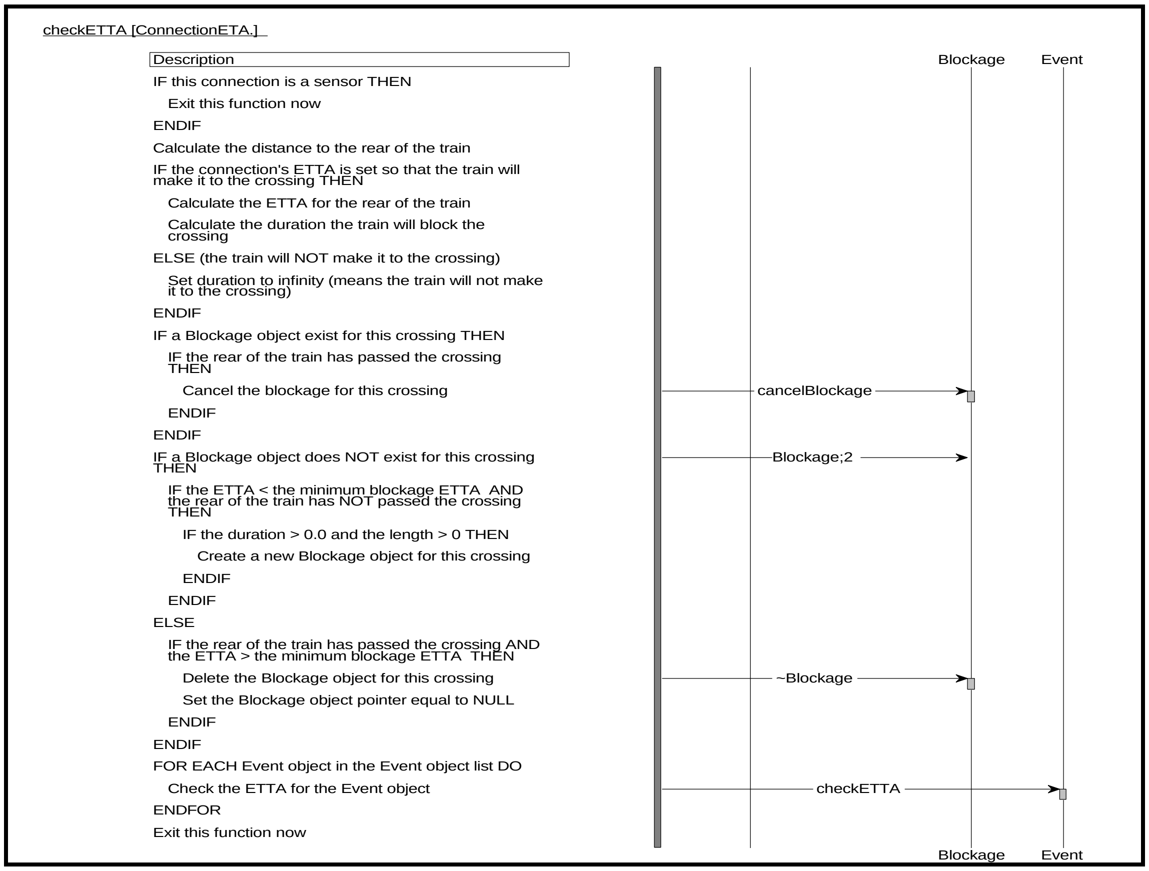
ETA	Estimated Time of Arrival (ETA).
ETATime	Time vel and acc acquired.
ETTA	Estimated Time unTil Arrival (ETTA).
acc	Acceleration used in ETA calculation.
blkg	Crossing Blockage (event) associated with this downstream item.
distTime	Time distance last calculated (a 0 indicates distance not set).
distance	Current distance to item (virtual sensor or crossing).
downstream	If this ConnectionETA is a crossing, then this list is empty. If it is a virtual sensor (and it is not a part of some other ConnectionETA's downstream list), then this is a list of all the downstream items.
events	Conditions that cause creation of ATMS alarms and associated alarm descriptors.
id	Item identifier, s1..sM for virtual sensors or c1..cP for crossings.
length	Length of the train.
maxVel	Either max velocity for this downstream sensor or max velocity when 'near' crossing.
name	Name of the downstream item by crossing guard.
nomTime	Nominal time from the train crossing where train is sensed.
origDist	Original distance to downstream item.
runLevel	The program's execution level. 0 is normal operations, -3 is simulation mode.
vSensorId	The associated virtual sensor ID.
Vel	Velocity used in the ETA calculation.

The ConnectionETA class contains the following operations:

PUBLIC:

`ConnectionETA`	Default constructor. Not really useful – just a placeholder because assigning data requires using the other constructors.
`ConnectionETA;2`	This constructor should be used to first load in ALL downstream crossings and put them in a list. Then, make a copy of the list and pass that copy to each item in the original using the finishCreate() function.
`ConnectionETA;3`	Constructor that takes a ConnectionETA instance as an initializer.
`ID`	Returns the item identifier.
`BlockageL`	Length used/reported in Blockage.

`checkETTA`	Check ETTA and scenarios and generate any appropriate crossing Blockages and ATMS events. The sequence diagram for this member function is given in Figure 69.

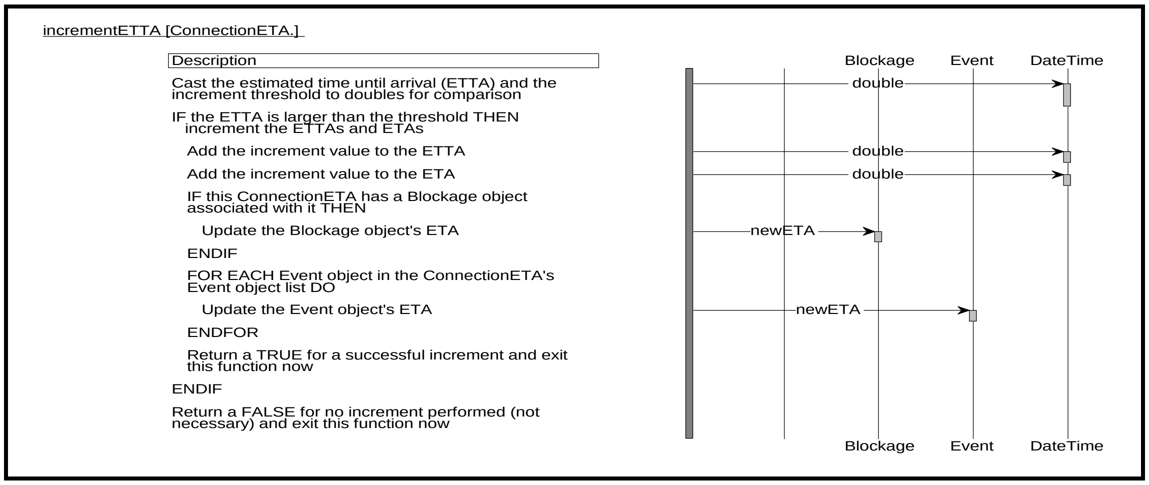


```
IF this connection is a sensor THEN
   Exit this function now
ENDIF
Calculate the distance to the rear of the train
IF the connection's ETTA is set so that the train will make it to the crossing THEN
   Calculate the ETTA for the rear of the train
   Calculate the duration the train will block the crossing
ELSE (the train will NOT make it to the crossing)
   Set duration to infinity (means the train will not make it to the crossing)
ENDIF
IF a Blockage object exist for this crossing THEN
   IF the rear of the train has passed the crossing THEN
      Cancel the blockage for this crossing
   ENDIF
ENDIF
IF a Blockage object does NOT exist for this crossing THEN
   IF the ETTA < the minimum blockage ETTA  AND the rear of the train has NOT passed the crossing THEN
      IF the duration > 0.0 and the length > 0 THEN
         Create a new Blockage object for this crossing
      ENDIF
   ENDIF
ELSE
   IF the rear of the train has passed the crossing AND the ETTA > the minimum blockage ETTA  THEN
      Delete the Blockage object for this crossing
      Set the Blockage object pointer equal to NULL
   ENDIF
ENDIF
FOR EACH Event object in the Event object list DO
   Check the ETTA for the Event object
ENDFOR
Exit this function now
```

Figure 69. ConnectionETA::checkETTA Sequence Diagram

`currentA`	Returns the acceleration used in ETA calculation.
`currentBlockage`	Crossing Blockage (empty/real). NOTE: May return an empty/invalid blockage. Use isBlockage() to determine if valid.
`currentDistance`	Returns the current distance to the downstream item.
`currentETA`	Returns the Estimated Time of Arrival (ETA).
`currentETTA`	Returns the Estimated Time unTil Arrival (ETTA).
`currentV`	Returns the velocity used in ETA calculation.
`downstreamList`	Returns a list of the downstream connections (none if it is a crossing).

`eventsList`	Returns a list of Scenarios and ATMS events.
`finishCreate`	This function is called after all the ConnectionETAs have been loaded from the configuration file and made into a list. Make a copy of the list and pass it to each of the items in the original list using this function. The maximum velocity for this sensor or connection must be provided (and be greater than 0).
`incrementETTA`	Increases ETTA/ETA if greater than some length of time (threshold) by a fixed amount (increment). Performs additional logic on these items which have a crossing Blockage or ATMS event (returns TRUE if the ETTA was increased). The sequence diagram for this member function is given in Figure 70.

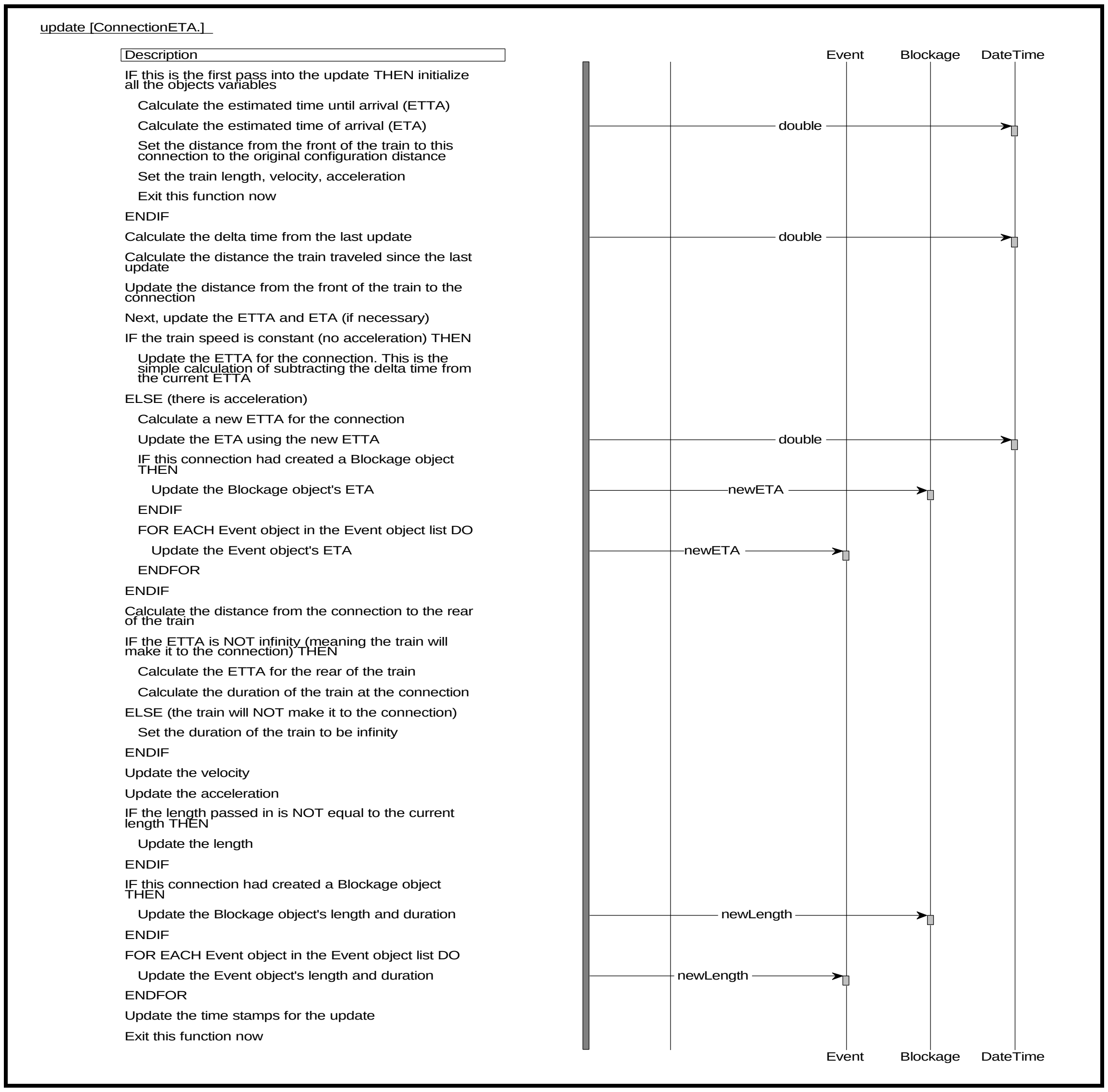


```
Cast the estimated time until arrival (ETTA) and the
increment threshold to doubles for comparison
IF the ETTA is larger than the threshold THEN
   increment the ETTAs and ETAs
  Add the increment value to the ETTA
  Add the increment value to the ETA
  IF this ConnectionETA has a Blockage object
  associated with it THEN
    Update the Blockage object's ETA
  ENDIF
  FOR EACH Event object in the ConnectionETA's
  Event object list DO
    Update the Event object's ETA
  ENDFOR
  Return a TRUE for a successful increment and exit
  this function now
ENDIF
Return a FALSE for no increment performed (not
necessary) and exit this function now
```

Figure 70. ConnectionETA::incrementETTA Sequence Diagram

`isBlockage`	Returns TRUE if Blockage is real. Must be used to know if currentBlockage returned an actual blockage or an empty one.
`isOverlappingTimes`	Determines if the two connections have the same overlapping times.
`lastETATime`	Returns the time vel and acc were acquired.
`majAlarmSent`	Returns TRUE if a major alarm was sent to ATMS.
`maxVelocity`	Returns the maximum velocity used in ETA calculation.
`merge`	Determines that two connections refer to the same downstream item and merge them together. Maintain any crossing Blockage (complain if ther are two). May require Blockage updating as well.

	Use the maximum velocity from instance with shortest original distance to downstream object.
`operator=`	Overloaded assignment operator to assign one ConnectionETA object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`origL`	Returns the configuration distance.
`sendNormalAlarm`	Sends a normal (cancel) alarm to ATMS.
`update`	Updates the internal data using new train information. The sequence diagram for this member function is given in Figure 71.


update [ConnectionETA.]

Description

```
IF this is the first pass into the update THEN initialize all the objects variables
  Calculate the estimated time until arrival (ETTA)
  Calculate the estimated time of arrival (ETA)
  Set the distance from the front of the train to this connection to the original configuration distance
  Set the train length, velocity, acceleration
  Exit this function now
ENDIF
Calculate the delta time from the last update
Calculate the distance the train traveled since the last update
Update the distance from the front of the train to the connection
Next, update the ETTA and ETA (if necessary)
IF the train speed is constant (no acceleration) THEN
  Update the ETTA for the connection. This is the simple calculation of subtracting the delta time from the current ETTA
ELSE (there is acceleration)
  Calculate a new ETTA for the connection
  Update the ETA using the new ETTA
  IF this connection had created a Blockage object THEN
    Update the Blockage object's ETA
  ENDIF
  FOR EACH Event object in the Event object list DO
    Update the Event object's ETA
  ENDFOR
ENDIF
Calculate the distance from the connection to the rear of the train
IF the ETTA is NOT infinity (meaning the train will make it to the connection) THEN
  Calculate the ETTA for the rear of the train
  Calculate the duration of the train at the connection
ELSE (the train will NOT make it to the connection)
  Set the duration of the train to be infinity
ENDIF
Update the velocity
Update the acceleration
IF the length passed in is NOT equal to the current length THEN
  Update the length
ENDIF
IF this connection had created a Blockage object THEN
  Update the Blockage object's length and duration
ENDIF
FOR EACH Event object in the Event object list DO
  Update the Event object's length and duration
ENDFOR
Update the time stamps for the update
Exit this function now
```

Figure 71. ConnectionETA::update

`vSensorID`	Returns the associated virtual sensor ID.
`~ConnectionETA`	Destructor - Deletes any blockages created.

PRIVATE:

`ReadConfig`	Reads the original configuration information.
`WriteConfig`	Write configuration information.

5.4.2.12 Event (Event) Class

This class holds information about when crossing blockage information should be provided to the ATMS as an incident/alarm. It also maintains the blockage details. The class is responsible for communicating to the ATMS (not the Data Server). This class inherits from the Blockage class, i.e. this class is a child of the Blockage class.

The Event class contains the following attributes:

PRIVATE:

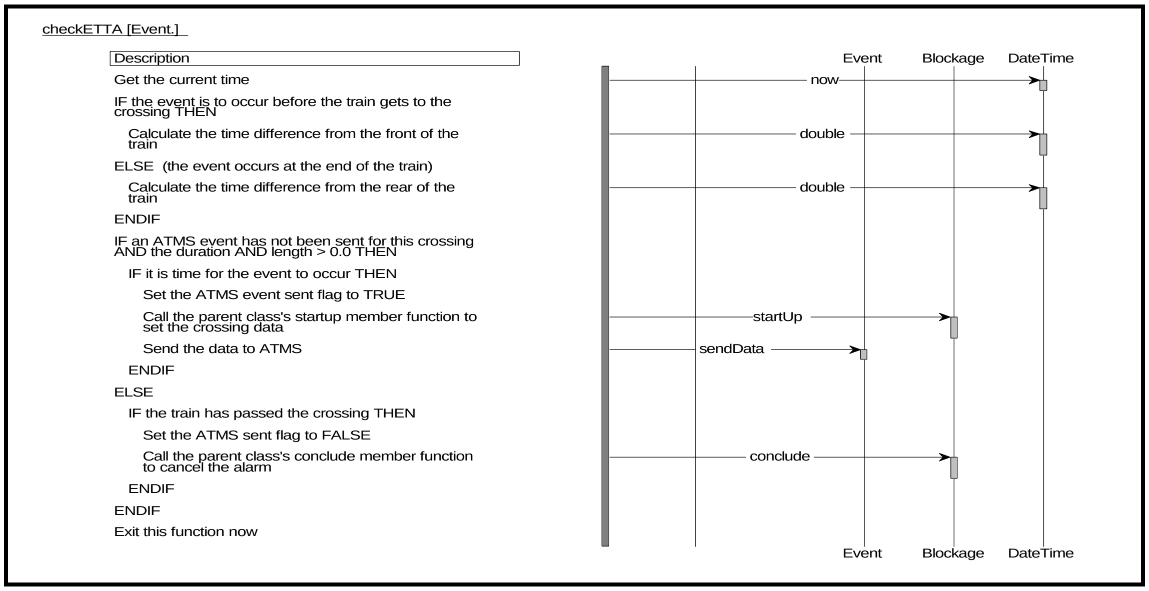
atmsEvent	Set to TRUE if an ATMS incident.
cancelSent	True if a normal alarm has been sent
crossingId	Crossing identifier, c1..cP, associated with this event.
endTime	The time before the end of the train to end the event.
eventId	Event or other ATMS identifier.
lastDuration	The duration of the train at the last sensor polling (this is used due to a loss of communication with the sensor and the time to reconnect could cause the event not to start or end).
lastLength	The length of the train at the last sensor polling (this is used due to a loss of communication with the sensor and the time to reconnect could cause the event not to start or end).
lastTimeCheck	The time of the last sensor polling (this is used due to a loss of communication with the sensor and the time to reconnect could cause the event not to start or end).
lastTrainETA	The ETA of the train at the last sensor polling (this is used due to a loss of communication with the sensor and the time to reconnect could cause the event not to start or end).
majorAlarmSent	Set to TRUE if ATMS major alarm has been sent.
name	The event name.
preceed	Set to TRUE if pretermination time means for the time to be before the train begins the blockage, set to FALSE to indicate time before train leaves the blockage.

pretermination	Time before crossing blockage starts or ends to initiate the ATMS event or less than 0 to ignore.
startTime	The time before the beginning of the train to start the event.

The Event class contains the following operations:

PUBLIC:

`Event`	Default constructor. Not really useful, just a placeholder because assigning data requires using the other constructors.
`Event;2`	Constructor that loads in configuration data from a supplied data file.
`Event;3`	Constructor that takes an Event instance as an initializer.
`atmsID`	Returns the event or other ATMS identifier.
`cID`	Returns the crossing ID.
`cancelAlarm`	Cancels an alarm sent to ATMS.
`cancelSENT`	Returns TRUE if a normal alarm for the event has been sent to ATMS.
`checkETTA`	Check event and generate any appropriate events. The sequence diagram for this member function is given in Figure 72.



```
Get the current time
IF the event is to occur before the train gets to the crossing THEN
   Calculate the time difference from the front of the train
ELSE  (the event occurs at the end of the train)
   Calculate the time difference from the rear of the train
ENDIF
IF an ATMS event has not been sent for this crossing AND the duration AND length > 0.0 THEN
   IF it is time for the event to occur THEN
      Set the ATMS event sent flag to TRUE
      Call the parent class's startup member function to set the crossing data
      Send the data to ATMS
   ENDIF
ELSE
   IF the train has passed the crossing THEN
      Set the ATMS sent flag to FALSE
      Call the parent class's conclude member function to cancel the alarm
   ENDIF
ENDIF
Exit this function now
```

Figure 72. Event::checkETTA Sequence Diagram

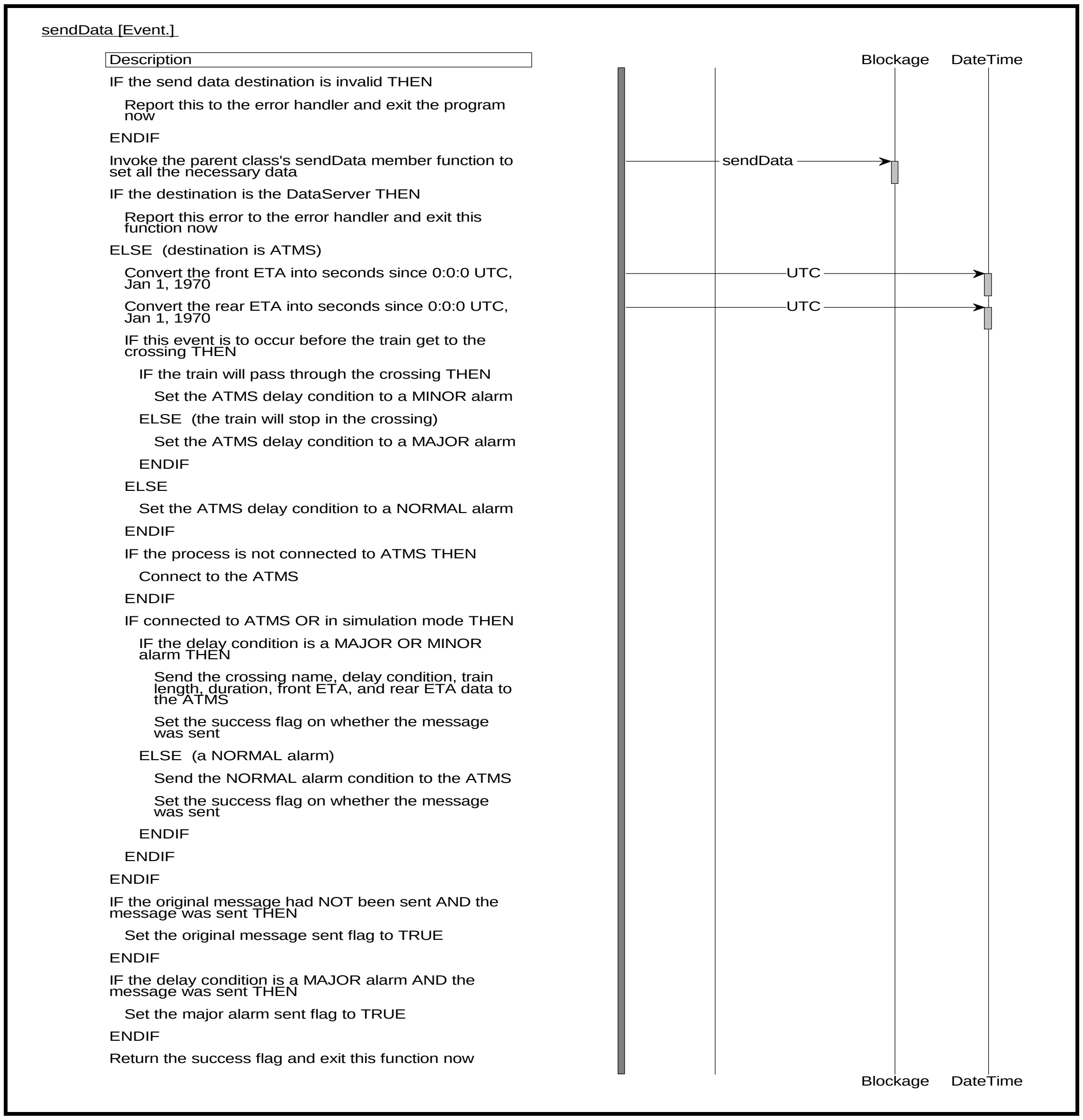
`endTIME`	Returns the end time of the event.
`hasEvent`	Returns TRUE if information has been sent to the ATMS.
`majAlarmSent`	Returns TRUE if a major alarm has been sent to ATMS.
`merge`	Determines that two events refer to the same downstream event and merge them. Uses maximum velocity from instance with the shortest original distance to the downstream object.
`newETA`	Updates the object with a new ETA.
`newLength`	Updates the object with a new length and estimated duration.
`operator=`	Overloaded assignment operator to assign one Event object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`preTime`	Returns the pretermination time.
`setCancelSent`	Sets the cancelSent flag to TRUE indicating the normal alarm has been sent to ATMS.
`sName`	Returns the event name.
`startTIME`	Returns the start time of the event.
`~Event`	Destructor. Will cancel any events sent to ATMS except Major Alarms.

PRIVATE:

`ReadConfig`	Reads the original configuration information.
`WriteConfig`	Write configuration information.

PROTECTED:

`sendData`	Sends/updates data to the destination. Note: This member function is overloaded from its parent. The overloaded function calls its parent's function as well. The sequence diagram for this member function is given in Figure 73.



```
IF the send data destination is invalid THEN
   Report this to the error handler and exit the program now
ENDIF
Invoke the parent class's sendData member function to set all the necessary data
IF the destination is the DataServer THEN
   Report this error to the error handler and exit this function now
ELSE  (destination is ATMS)
   Convert the front ETA into seconds since 0:0:0 UTC, Jan 1, 1970
   Convert the rear ETA into seconds since 0:0:0 UTC, Jan 1, 1970
   IF this event is to occur before the train get to the crossing THEN
      IF the train will pass through the crossing THEN
         Set the ATMS delay condition to a MINOR alarm
      ELSE  (the train will stop in the crossing)
         Set the ATMS delay condition to a MAJOR alarm
      ENDIF
   ELSE
      Set the ATMS delay condition to a NORMAL alarm
   ENDIF
   IF the process is not connected to ATMS THEN
      Connect to the ATMS
   ENDIF
   IF connected to ATMS OR in simulation mode THEN
      IF the delay condition is a MAJOR OR MINOR alarm THEN
         Send the crossing name, delay condition, train length, duration, front ETA, and rear ETA data to the ATMS
         Set the success flag on whether the message was sent
      ELSE  (a NORMAL alarm)
         Send the NORMAL alarm condition to the ATMS
         Set the success flag on whether the message was sent
      ENDIF
   ENDIF
ENDIF
IF the original message had NOT been sent AND the message was sent THEN
   Set the original message sent flag to TRUE
ENDIF
IF the delay condition is a MAJOR alarm AND the message was sent THEN
   Set the major alarm sent flag to TRUE
ENDIF
Return the success flag and exit this function now
```

Figure 73. Event::sendData Sequence Diagram

5.4.2.13 Blockage (Blockage) Class

This class is responsible for containing information summarizing the blockage of a railroad crossing and communicating the information to either the DataServer or the ATMS. The data is primarily a summary of when a train is expected to be at a crossing and the length and duration of the blockage. This class is the parent class of the Event class. Notice that this class should be able to be used in an ordered list because it has an equality operator, a copy constructor, an assignment operator, and a default constructor.

The Blockage class contains the following attributes:

PROTECTED:

ETA	Estimated Time of Arrival (ETA).
blockId	The unique blockage id.
blocks	A global list of active blockages. Note: This global list is only visible to the Blockage objects and its children.
cancelSent	Flag indicating that this blockage cancel notification has been sent successfully (at least once) to the destination.
destination	Destination for the data and updates provided to this class.
detail	Communication details for talking to the ATMS.
duration	Estimated duration of the crossing blockage.
eventFrontETA	The calculated Estimated Time of Arrival (ETA) of the front of the train.
eventRearETA	The calculated Estimated Time of Arrival (ETA) of the rear of the train.
id	Crossing item identifier, c1..cP.
lastBlock	Global value of the last block ID generated. Note: This global ID is only visible to the Blockage objects and its children.
length	Length of the train.
name	The name of the downstream item.
origSent	Flag indicating that this blockage notification has been sent successfully (at least once) to the destination.
runLevel	The program's execution level. 0 is normal operations, -3 is simulation mode.

The Blockage class contains the following operations:

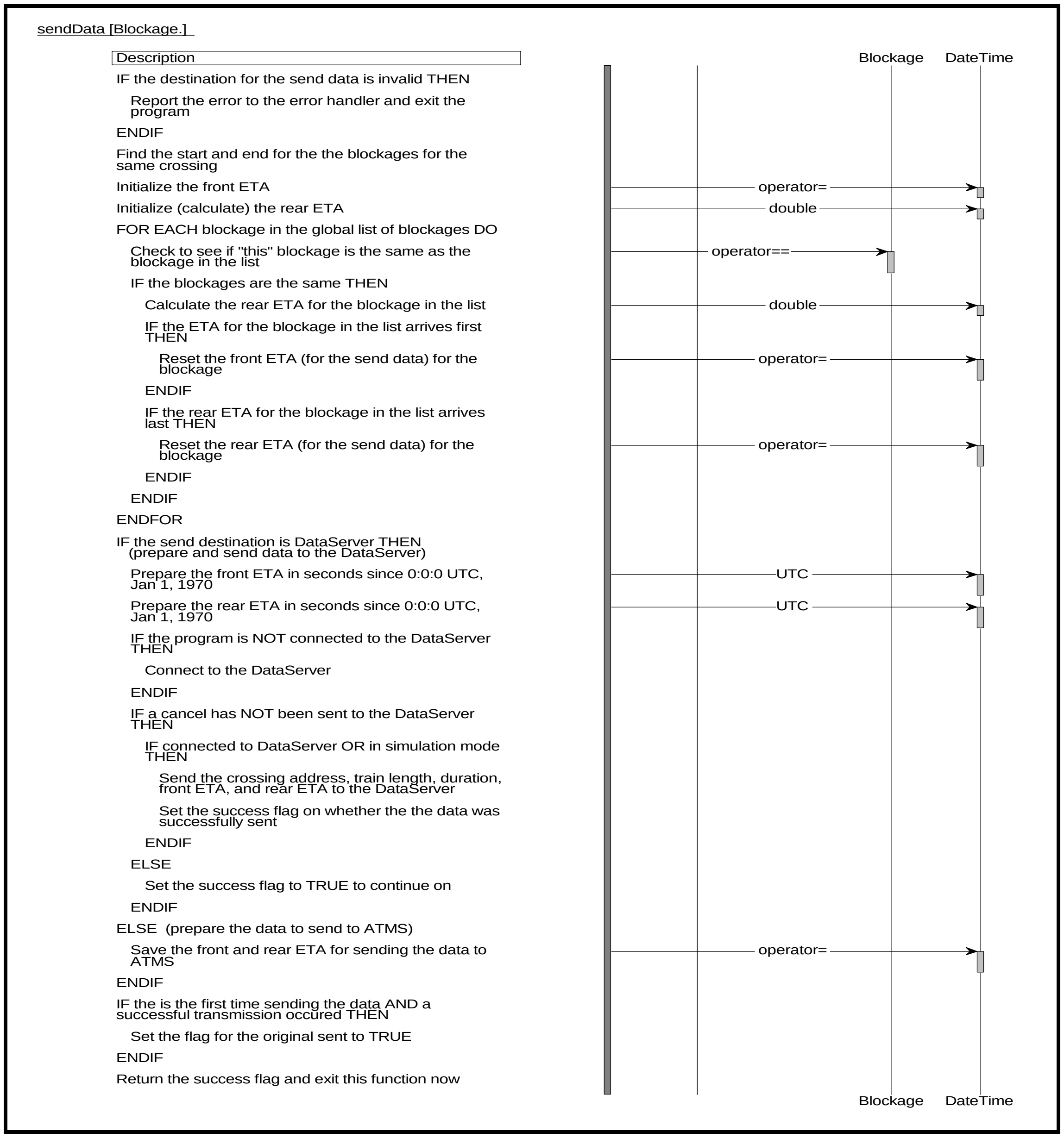
PUBLIC:

`Blockage`	Default constructor. Note: the default constructor does nothing useful, but allows for having the class as part of an ordered list. Additionally initialization data may only be entered into the class by utilizing a non-default constructor.
`Blockage;2`	Constructor where all values passed must be non-NULL and have a valid function.
`Blockage;3`	Constructor that takes a Blockage instance as an initializer.
`atETA`	Returns the ETA of the blockage.
`blockID`	Returns the unique block ID.
`cancelBlockage`	Train has disappeared or whatever, handle the situation appropriately (based on the destination).
`itemID`	Returns the item identifier.
`itemName`	Returns the name of the downstream item.
`newETA`	Updates the object with a new ETA.
`newLength`	Updates the object with a new length and duration.
`operator=`	Overloaded assignment operator to assign one Event object to another.
`operator==`	Overloaded equality operator used to detect that two instances are identical.
`trainDuration`	Returns the train duration of the item.
`trainLength`	Returns the train length of the item.
`~Blockage`	Destructor.

PROTECTED:

`conclude`	Performs all the destructor operations.
`NewBlockID`	Return a new valid block ID.

`sendData` Sends and updates data to the destination.
The sequence diagram for this member function is given in Figure 74.



```
Description
IF the destination for the send data is invalid THEN
   Report the error to the error handler and exit the program
ENDIF
Find the start and end for the the blockages for the same crossing
Initialize the front ETA
Initialize (calculate) the rear ETA
FOR EACH blockage in the global list of blockages DO
   Check to see if "this" blockage is the same as the blockage in the list
   IF the blockages are the same THEN
      Calculate the rear ETA for the blockage in the list
      IF the ETA for the blockage in the list arrives first THEN
         Reset the front ETA (for the send data) for the blockage
      ENDIF
      IF the rear ETA for the blockage in the list arrives last THEN
         Reset the rear ETA (for the send data) for the blockage
      ENDIF
   ENDIF
ENDFOR
IF the send destination is DataServer THEN
   (prepare and send data to the DataServer)
   Prepare the front ETA in seconds since 0:0:0 UTC, Jan 1, 1970
   Prepare the rear ETA in seconds since 0:0:0 UTC, Jan 1, 1970
   IF the program is NOT connected to the DataServer THEN
      Connect to the DataServer
   ENDIF
   IF a cancel has NOT been sent to the DataServer THEN
      IF connected to DataServer OR in simulation mode THEN
         Send the crossing address, train length, duration, front ETA, and rear ETA to the DataServer
         Set the success flag on whether the the data was successfully sent
      ENDIF
   ELSE
      Set the success flag to TRUE to continue on
   ENDIF
ELSE  (prepare the data to send to ATMS)
   Save the front and rear ETA for sending the data to ATMS
ENDIF
IF the is the first time sending the data AND a successful transmission occured THEN
   Set the flag for the original sent to TRUE
ENDIF
Return the success flag and exit this function now
```

Figure 74. Blockage::sendData Sequence Diagram

`startUp` Performs all the constructor operations.

5.4.2.14 Date and Time (DateTime) Class

This class is used to manipulate and store time and to time short events.

The DateTime class contains the following attributes:

PRIVATE:

jday	Julian day (i.e. days since) since the start of year 0..364 (365 in a leap year).
sec	The seconds into the jday 0..86399 (may be fractional).
yr	The year A. D. 0...

The DateTime class contains the following operations:

PUBLIC:

`DateTime`	The default constructor.
`DateTime;2`	The constructor function given normal date and time data.
`DateTime;3`	The constructor function given Julian date and time data.
`DateTime;4`	The constructor function given time data (seconds since time 0).
`DateTime;5`	Constructor that takes a DateTime instance as an initializer.
`UTC`	Returns seconds since 0:0:0 UTC, Jan 1, 1970.
`double`	Convert to seconds since 0. This is most appropriate for a delta time.
`now`	Sets the current DateTime instance to be the local date and time on the computer.
`operator+`	Overloaded addition operator that uses another DateTime instance to add to this DateTime instance.
`operator+;2`	Overloaded addition operator that uses a double to add to this DateTime instance.
`operator-`	Overloaded subtractor operator that uses another DateTime instance to subtract from this DateTime instance.

`operator-;2`	Overloaded subtractor operator that uses a double to subtract from this DateTime instance.
`operator=`	Overloaded assignment operator to assign one DateTime object to another.
`val`	Provides the date and time in the current instance in standard format.
`val;2`	Overloaded function that provides the date and time in the current instance in Julian date format.
`wDay`	Returns the day of the week (1..7 - Sunday..Saturday) as long as the year is greater than 1900.
`~DateTime`	Destructor.

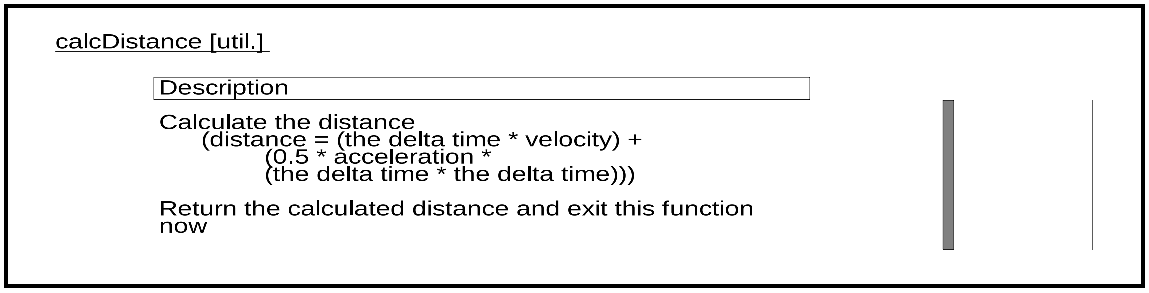
PRIVATE:

`cvt_jday`	Returns the Julian day for the provided month and day and year. Note: This includes the affects of leap years.
`cvt_mo_da`	Returns the month and day of the month for the current instance. Note: This includes the affects of leap years.

5.4.2.15 Utility Functions

The following functions areutility routines that are used by many different classes in the AWARD system. These routines were made generic so that any class can use them and it won't be necessary to remake these as member functions in the classes.

`calcDistance`	Calculate the distance traveled by an object given the velocity, acceleration, and a time (the delta time from the start and end times). The sequence diagram for this member function is given in Figure 75.



```
Calculate the distance
     (distance = (the delta time * velocity) +
             (0.5 * acceleration *
             (the delta time * the delta time)))

Return the calculated distance and exit this function
now
```

Figure 75. calcDistance Sequence Diagram

`calcETTA` Calculate the Estimated Time unTil Arrival (ETTA) by an object given the velocity, acceleration, and distance the object has traveled.
The sequence diagram for this member function is given in Figure 76.

calcETTA [util.]

Description

```
Initialize the return ETTA to (representation of) infinity
Initialize the minimum velocity to 0.0
Get the absolute value of the acceleration
IF the velocity is close to 0.0 THEN
   Return the (representation of) infinity and exit this function now
ELSE
   Calculate the approximate time (distance / velocity)
   Calculate the approximate velocity (velocity + (acceleration * time approximate))
ENDIF
IF the time approximate is less than zero THEN
       the object will never cover the distance will never be
       traversed
   Return the (representation of) infinity and exit this function now
ELSE IF the accelleration is or near zero THEN
       the ETTA will be distance / velocity
   Return the calculated time approximate
ELSE IF the (velocity + accelleration * time approximate) is
     between the minimum velocity and maximum velocity THEN
       calculate ETTA using the quadratic formula: t =
       (-v +|- square root of (v squared + 4(1/2acc*dist)))/
       2(1/2a).
       Other checks are necessary before the square root is
       taken
   Calculate the value the square root will be performed on
     square_root_of =
       (velocity * velocity) + (2 * acceleration * distance)
   IF acceleration > 0.0 THEN
      Set the return ETTA = -velocity +
                  sqare root of (square_root_of)
      IF the return ETTA < 0.0 THEN
         Set the return ETTA = -velocity -
                  sqare root of (square_root_of)
      ENDIF
      Set the return ETTA = the return ETTA  / acceleration
   ELSE  (the train is decelerating)
      IF the square_root_of < 0.0 THEN
         Return the (representation of) infinity and exit this function now
      ELSE
         Calculate the first time (t1) =
               (-velocity + square root of (square_root_of)) /
               acceleration
         Calculate the first time (t2) =
               (-velocity - square root of (square_root_of)) /
               acceleration
         IF the first time ( t1) < 0.0 THEN
            Set return ETTA to the second calculated time (t2)
         ELSE IF the second time ( t2) < 0.0 THEN
            Set return ETTA to the first calculated time (t1)
         ELSE
            IF the first calculated time (t1) < the second calculated time (t2) THEN
               Set return ETTA to the first calculated time (t1)
            ELSE
               Set return ETTA to the second calculated time (t2)
            ENDIF
         ENDIF
      ENDIF
   ENDIF
ELSE IF the aproximate velocity < the minimum velocity THEN
       the object will never traverse the distance given
   Return the (representation of) infinity and exit this function now
ELSE  (the object's current accelleration will cause the object
       to exceed its maximum velocity. Calculate the time
       and place when the train will reach its maximum
       velocity and assume constant velocity from there to
       the destination)
   Calculate the time the object will reach its maximum velocity
       (t_maxv = (maxvel - vel) / acceleration)
   Calculate the distance the object will traveled at its maximum velocity
       (dist = distance -
               calcDistance(t_maxvel, vel, acceleration))
   Calculate the time the object will travel at its maximum velocity
       (time_prime = dist / maxvel)
   Set the return ETTA to the calculated value
       (return ETTA = t_maxv + time_prime)
ENDIF
Return the calculated value of the return ETTA and exit this function now
```

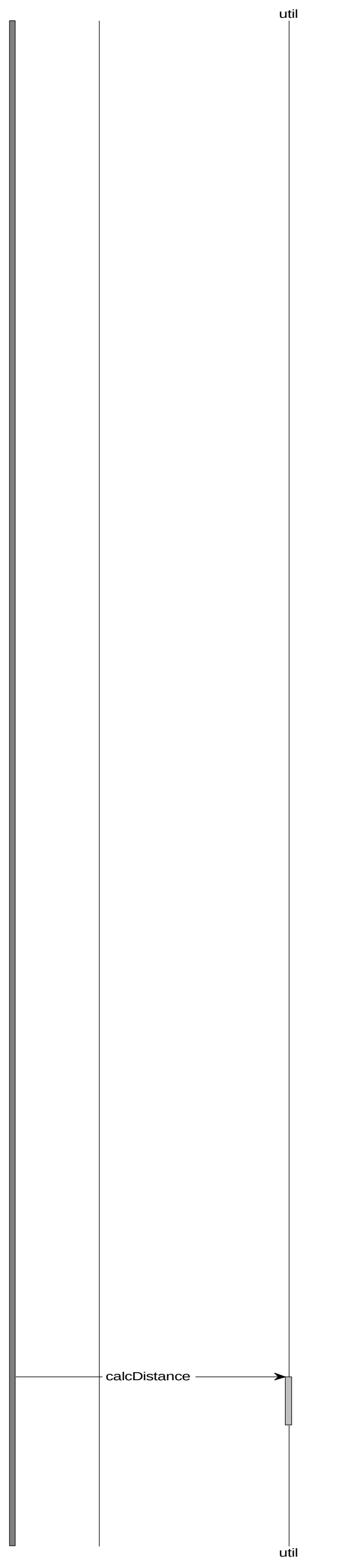


Figure 76. calcETTA Sequence Diagram

`ConnectToATMS`	This connects to the ATMS process for sending crossing delay data.
`ConnectToDataServer`	This connects to the DataServer process for sending sensor and crossing data.
`ConnectToHeartBeat`	This connects to the HeartBeat process for sending heartbeats informing it that AWARD is still running.
`ConnectToStatusLogger`	This connects to the status logger process for sending status messages about AWARD.
`SendCrDataToATMS`	This sends crossing delay information to ATMS.
`SendCrDataToDS`	This sends crossing delay information to the DataServer.
`SendHeartBeat`	This sends a heartbeat to the HeartBeat process.
`SendMessageToStatusLogger`	This sends message information to the status logger.
`SendSensorDataToDS`	This sends sensor information to the DataServer.

6. Traceability Matrix

The traceability matrix for the Railroad Delay Advance Warning System (AWARD) is presented in this section. It lists the requirements of the system that were presented in Section 3.0 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.

Table 69, shown on the following pages, 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 requirements in the traceability matrix are organized by requirement number. Each requirement in the matrix has a unique requirements identification (ID) label which maps the particular requirement to a subsystem with the AWARD System. The ID labels are defined as:

RR-GEN-XX AWARD General (Programmatic) Requirements
RR-SYS-XX AWARD System Requirements
RR-SNS-XX AWARD Sensor Subsystem Requirement
RR-COM-XX AWARD Communications Subsystem Requirement
RR-MEC-XX AWARD Mechanical Subsystem Requirement
RR-RRS-XX AWARD-Railroad Software Subsystem Requirement
RR-TGS-XX AWARD-TransGuide Operational Software Subsystem Requirement

Table 69. **AWARD System Traceability Matrix**

REQUIREMENT ID	REQUIREMENT DESCRIPTION	REQUIREMENT SOURCE P - PROPOSAL T - TXDOT RFO	DESIGN DOCUMENT PARAGRAPH
RR-GEN-01	An 80% System design document shall be delivered on February 14, 1997.	P-2.1.2.8.3	
RR-GEN-02	A 100% design document shall be delivered on December 31, 1997	P-2.1.2.8.3	
RR-GEN-03	A Software Acceptance Test Plan shall be delivered	P-2.1.2.8.3	
RR-GEN-04	A Version Description Document shall be delivered.	P-2.1.2.8.3	
RR-GEN-05	Monthly status reports shall be provided via a presentation with the customer.	P-2.1.2.8.3 (revised)	
RR-GEN-06	A training program shall be presented	P-2.1.2.8.3	
RR-GEN-07	A videotape of the training program shall be delivered.	P-2.1.2.8.3	
RR-GEN-08	A final report shall be delivered.	P-2.1.2.8.3	
RR-SYS-01	The system shall deliver advance warning to motorists of expected delays at railroad crossings	T-28	4.1, 4.7.2.2
RR-SYS-02	The system shall determine the speed and length of a train engine and attached railroad cars.	T-28	4.7.1.3, 4.7.2.1
RR-SYS-03	The system shall determine expected delay times at selected grade crossings.	T-28	4.7.1.3, 4.7.2.1
RR-SYS-04	The system shall transmit an expected delay to TransGuide Operators as an alarm through a software interface with the existing TransGuide ITS system.	T-28	4.7.2.2

REQUIREMENT ID	REQUIREMENT DESCRIPTION	REQUIREMENT SOURCE P - PROPOSAL T - TXDOT RFO	DESIGN DOCUMENT PARAGRAPH
RR-SYS-05	Expected railroad delays shall be transmitted to the traveling public by use of existing variable message signs and also to the MDI Data Server.	T-28	4.7.2.3
RR-SYS-06	The system shall provide warnings for grade crossings at IH 10 and Fredricksburg Road, IH 10 and Hildebrand Road, and IH 410 and Vance Jackson Road.	T-29.1.3	4.2
RR-SYS-07	The field equipment shall be mounted on a suitable structure at some location along the railroad line in advance of the crossing for which warnings are to be given.	T-29.2.1	4.2
RR-SYS-08	Field equipment shall be located in TxDOT or the City of San Antonio right-of-way.	T-29.2.1	4.2, 4.3.3
RR-SYS-09	The field equipment shall determine length and speed of trains through observation only. No connection to the railroad tracks or controlling equipment will be used.	T-29.2.2	4.3.2
RR-SNS-01	The train speed sensor shall have a range to allow measurement of the train speed from a location outside the railroad right-of-way. This distance is normally 50 feet on either side of the track center line but may vary in some locations	RR-SYS-04	4.3.2
RR-SNS-02	The detector unit shall measure locomotive speed within 2 miles per hour (+/-) at the maximum train speeds allowed for the section of track where sensors are installed.	T-29.2.4	4.3.2
RR-COM-01	The field unit shall communicate to the TransGuide equipment using a non-proprietary protocol.	T-29.1.2	2.1, 2.4.1.1

REQUIREMENT ID	REQUIREMENT DESCRIPTION	REQUIREMENT SOURCE P - PROPOSAL T - TXDOT RFO	DESIGN DOCUMENT PARAGRAPH
RR-ELC-01	The field unit shall operate on standard line power. (nominal 120 VAC)	RR-545-07	4.5
RR-MEC-01	The equipment will be designed to operate within an ambient temperature range of -12°C to 49°C (10°F to 120°F) and will not allow condensation accumulations which would interfere with its operation.	RR-SYS-07	4.6.1.1
RR-MEC-02	The system enclosure will be able to be mounted to a pole or other suitable structure.	RR-SYS-07 and RR-SYS-08	4.6.1.3
RR-MEC-03	The system will provide an internal mechanism for accurate pointing of the sensor.	RR-SNS-01	4.6.1.2.1
RR-RRS-01	The RR-Delay Master Computer shall calculate the length of the train from measured train speed integrated over time.	P-2.4.1	4.7.1.3
RR-RRS-02	The RR-Delay Master Computer shall calculate the expected time of arrival of the first element of the train and the last element of the train at selected downrail crossings.	P-2.4.1	4.7.1.3
RR-RRS-03	The RR-Delay Master Computer shall determine expected delay times at railroad crossings. The RR-Delay Master Computer shall estimate delay time within±30 seconds.	T-29.3.3	4.7.1.3
RR-RRS-04	The RR-Delay Master Computer shall transmit the railroad delay data to the existing TransGuide ITS system.	T-29.3.3	4.7.1.5
RR-TGS-01	The TransGuide Operational Software shall interface with and receive railroad delay data from the Railroad Operational Software.	RR-RRS-04	4.7.2

REQUIREMENT ID	REQUIREMENT DESCRIPTION	REQUIREMENT SOURCE P - PROPOSAL T - TXDOT RFO	DESIGN DOCUMENT PARAGRAPH
RR-TGS-02	The TransGuide Operational Software shall transmit expected delay information to TransGuide operators as an alarm.	T-28	4.7.2.2
RR-TGS-03	The TransGuide Operational Software shall be capable of performing a scenario search for a RR delay incident.	P-2.4.1	4.7.2.3
RR-TGS-02.01	The AIH shall accept a RR delay alarm from the RSS.	RR-TGS-02	4.7.2.2
RR-TGS-02.02	The AIH shall indicate the RR delay alarm as an update alarm if the RR delay alarm is related to a current RR delay incident.	RR-TGS-02	4.7.2.2
RR-TGS-02.03	The AIH shall create a new AIH RR incident if the RR delay alarm is not related to a current RR delay incident.	RR-TGS-02	4.7.2.2
RR-TGS-02.04	The AIH RR incident shall contain data from the railroad delay information contained in the RR delay alarm.	RR-TGS-02	4.7.2.2
RR-TGS-02.05	The AIH shall build the AIH RR incident screen for new RR delay alarms.	RR-TGS-92	4.7.2.2
RR-TGS-02.06	The AIH shall display the AIH RR incident screen, as an icon, on the workstation of the manager responsible for the sector containing the RR incident.	RR-TGS-02	4.7.2.2
RR-TGS-02.07	The AIH shall generate an audio notification of new RR incident alarms at the workstation of the manager responsible for the sector containing the RR incident.	RR-TGS-02	4.7.2.2

REQUIREMENT ID	REQUIREMENT DESCRIPTION	REQUIREMENT SOURCE P - PROPOSAL T - TXDOT RFO	DESIGN DOCUMENT PARAGRAPH
RR-TGS-02.08	The AIH shall update the railroad delay information for an existing incident using the railroad delay information contained in the associated RR delay update alarm.	RR-TGS-02	4.7.2.2
RR-TGS-02.09	The AIH RR incident screen shall provide the same actions currently provided by the AIH-NewIncidentScreen.	RR-TGS-02	4.7.2.2
RR-TGS-03.01	The SCM-ScenarioSearchScreen shall contain the RR incident type for selection by a TransGuide operator.	RR-TGS-03	4.7.2.3