Fantastic Data

207 Prospect Avenue, San Francisco, California 94110, USA • (415) 643-9555 • (415) 643-9554 • www.fandat.com

Sensor Node User's Guide

25 May 2006

TABLE OF CONTENTS

LIST OF FIGURES

1. Normal Operation

A network of Fantastic Data sensor nodes provides ground target surveillance over a wide outdoor area. The sensor nodes detect moving targets with passive infrared detectors and collaboratively from target tracks and predict target movement automatically within the sensor network. Target tracks may be extracted from the sensor network by executing a network query on a laptop computer attached to any sensor node. Raw detections, node locations, and node status may also be extracted using network queries.

Each individual node is capable of detecting moving personnel in one direction for a distance of about 20 meters. Cars and trucks may be detected at much longer ranges. The sensor nodes rely on each other to cover a field and to provide self protection. An individual node has no rear or side facing sensor and thus may be vulnerable to attack from those directions. It may be placed so that the environment protects it from attack from these angles (for example, mounted on a building or tree) or it may rely on its fellow nodes for protection. A randomly distributed network of 50 sensor nodes can cover an area larger than a normal athletic field with interlocking sensor beams as shown in Figure 1. There is no known limit to the size or extent of the sensor network.

1.1 Applicability

This guide describes the following sensor node versions.

Hardware: Sensor 2.2, 7/2005; Processor 1.0, 3/2005; Modem 1.2, 7/2005
Firmware: 1.4.22, 1/14/2006
Software: 1.5.10, 5/19/2006

Other node versions may be similar.

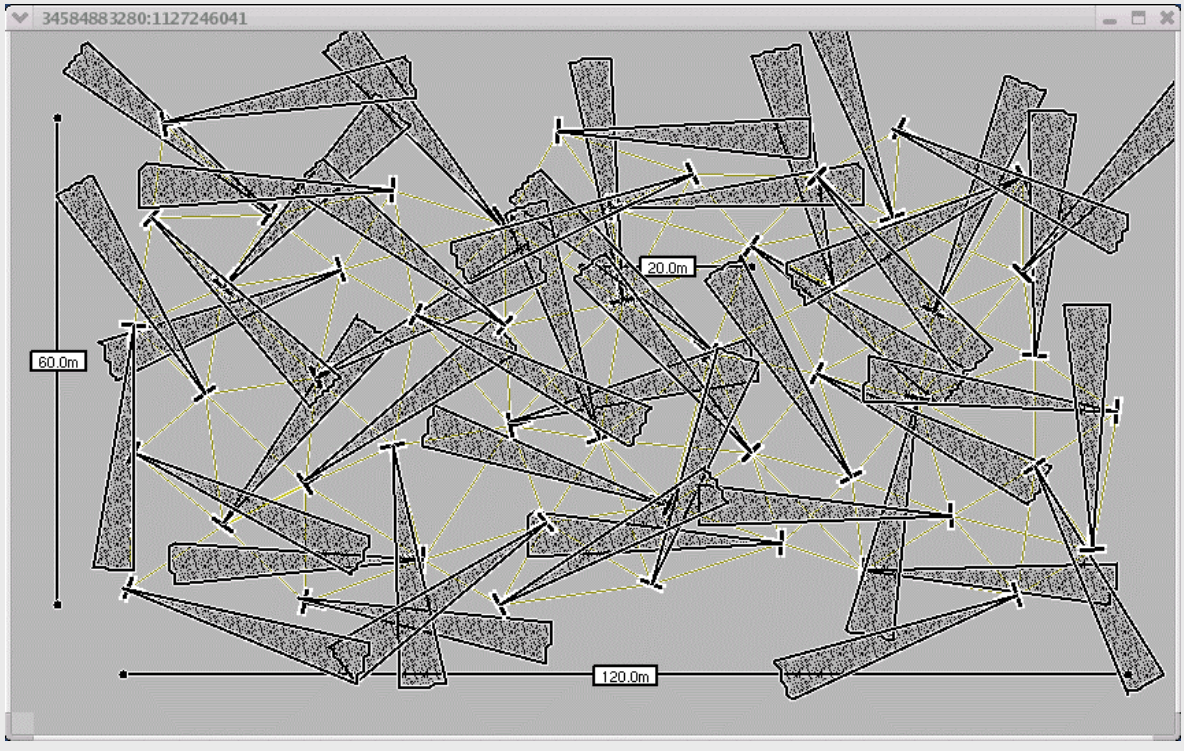

Figure 1. In this example network deployment 50 nodes cover an area larger than a standard athletic field with interlocking sensor beams.

Figure 2. The Fantastic Data sensor node.

1.2 The Sensor Node

The Fantastic Data sensor node is delivered ready for operation as shown in Figure 2. Software and firmware are loaded on the node and all required calibration has been performed. .

The node is housed in a weatherproof case painted the standard Army green. The case may be repainted as desired. Do not use metallic paints. Do not paint the sensor lens.

1.3 Battery Installation

You must install batteries to begin operation. Once batteries are installed and the sensor nodes are deployed, network formation proceeds automatically and the network begins target detection and tracking. The node lifetime is estimated between 1 and 6 days on a single set of batteries.

The node may be powered with several types of batteries. The node uses 2 standard AA batteries. Rechargable NiMH batteries are supplied with the node. These are the recommended battery choice. Alkaline or Lithium AA batteries may also be used, although they are not recommended because of the high cost associated with disposable batteries. The dual-AA sized Lithium CRV3 battery cannot be used.

To install batteries you must first locate and remove the 4 small screws securing the rear of the case to the node. After removing the screws, the rear of the case may be removed by pulling it straight back from the front of the case. Sometimes the case fits tightly. Do not twist or turn the case.

2 AA batteries should be inserted in the exposed battery holder as shown in Figure 3. Within seconds the node will begin operation. It starts by flashing all of its internal diodes 10 times. As these diodes are inside the node they may be hard to see. Once the diodes flash, you know that the batteries are correctly installed.

Replace the rear case. Take care that the GPS antenna at the top of the case is correctly aligned with its mounting slots. Reinsert the screws and tighten them until they are medium tight. Do not over tighten.

The node should be deployed within 5 minutes for maximum effectiveness. If possible, install the batteries at the deployment site, as this reduces the chance of incorrect network formation. If multiple

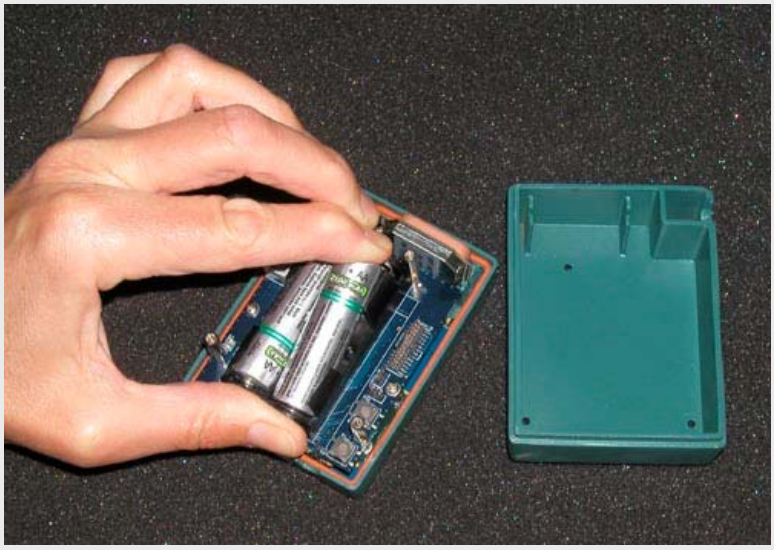

Figure 3. The sensor node shown with the rear case removed to allow insertion of batteries.

nodes are powered up indoors in a confined location, they will form a network prior to deployment. Forgetting about this false network requires some time and expenditure of energy.

1.4 Connecting a Serial Cable for Information Extraction

To extract information or to use any of the built-in test and configuration commands, you must attach the serial cable to the node as shown in Figure 4. This cable has a standard male DB-9 connector on one end for connection to the computer and a special 3 conductor headphone jack on the other end for connection to the node. The cable should be connected to the node before power is applied. The serial port on the computer should be configured to support 38400 baud, 8 bit data, no parity, and 1 stop bit.

Figure 4. The special serial cable is used to connect the node to a computer.

1.5 Viewing the Surveillance Information

The information products of the sensor network—node locations, node status, detections, and tracks—can be viewed as text data records on a simple terminal emulator. However, that process is less than satisfactory since it does lead to a good understanding of the situation. The information products may also be automatically extracted, analyzed, and displayed. Figure 5 shows the view created by one such automatic extraction and display program. This particular program extracts the data to a computer running the Linux operating system. The data records developed on the sensor network are automatically extracted by issuing the appropriate data access commands and the results are stored in the Linux version of the Fantastic Data Cache. The display program is interfaced to the data cache to display the data in a meaningful manner. Detailed directions to configure and operate this display system may be found in Section 4.

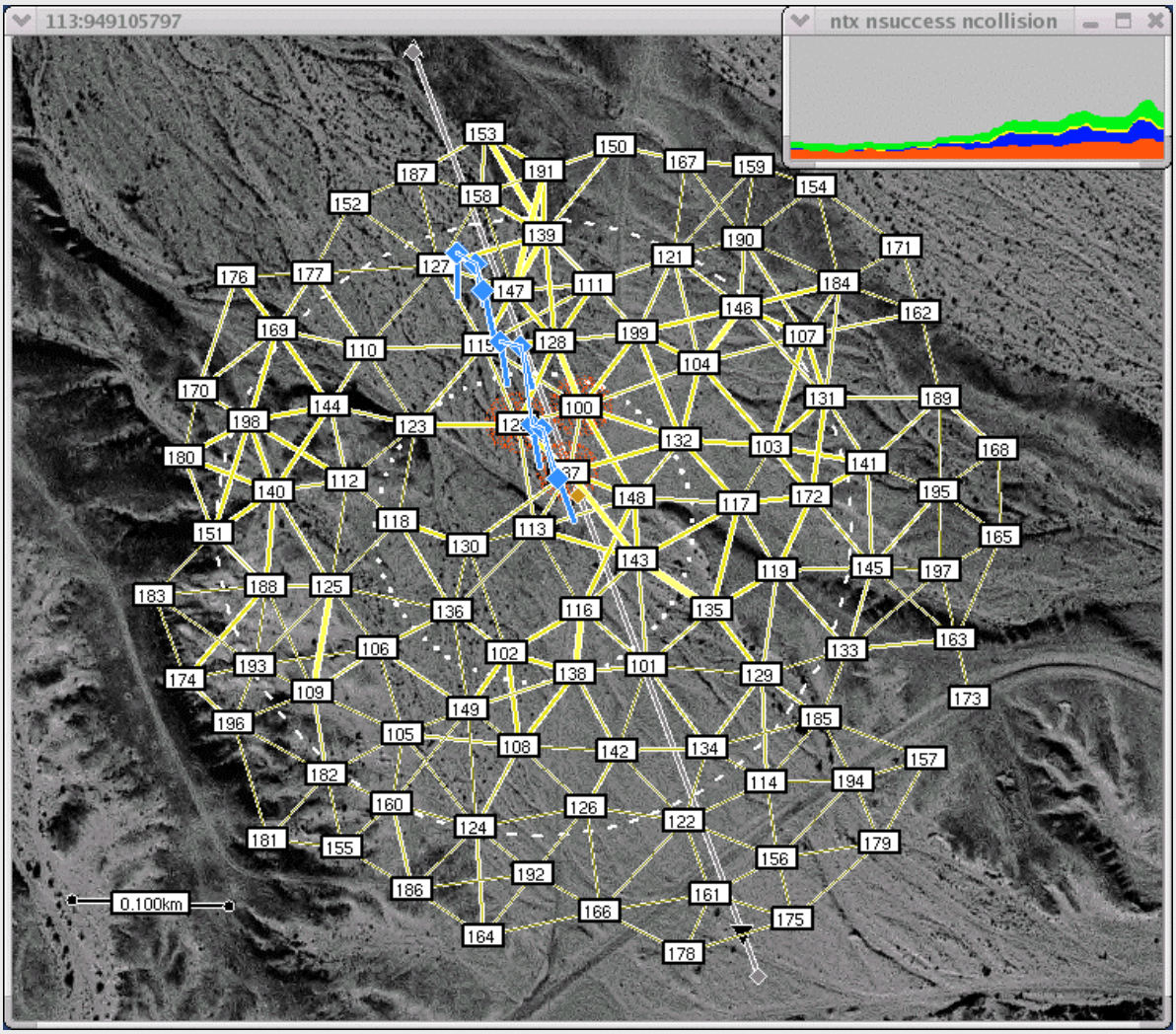

Figure 5. A track (light blue diamond and velocity vector) formed by the protocol suite from simulated detections (red splotches) shown superimposed on the 100 node simulation environment. The tracks and detections are viewed from node 113: detections are filtered by the dotted circle; tracks by the larger dashed circle. In-network tracking is an especially severe test of the communication protocol. The small brown diamond shows the true location of the simulated target that is traveling along the gray route at 20 Km/hr.

2. Configuration and Test

The Fantastic Data Sensor Node has built in self test and configuration functions. The sensor node is delivered ready for normal operation. It can be reconfigured for special operations or tested with the functions described in this section.

All of the commands are made available through the node serial port. Input may be corrected with 3 special characters: backspace (control h) erases the last character, control w erases the last word, and control x erases all of the input. Input is interpreted when the enter key is pressed. The entire command does not have to be typed—just enough to distinguish it from all other possibilities. For example, you must type **log** to use the **log** command but only **loo** to use the **loop** command. Similarly, typing **w** activates the **watch** command.

In this section, all commands, keywords, and parameter names appear in a bold typewriter like font such as **clock**. The default value of each parameter appears in the text enclosed in parentheses following the parameter name as **ClockSleepEnable (1)**. Words for which you should substitute appropriate values when issuing commands appear in square brackets such as **[memory address, hex]**. The use of a vertical bar inside the brackets, denotes a choice from a set of alternatives as **[on|off]**. Responses from the node appear offset from the text and in the regular typewriter font such as

```
Node id: 0f639b007240b199 85c2
Software: 1.5.10 06.05.19
Firmware: 1.4.22 06.01.14i
```

Figure 6 shows a high level view of the organization of the software, firmware, and hardware of the sensor node. An understanding of this organization may facilitate the use of the built-in test and configuration functions.

2.1 Clock and Task Management

The commands that control the software, firmware, and hardware used to control the node clock and task scheduling are described in this section.

2.1.1 Clock Configuration and Test

The node clock is central to the operation of the node. True time of day must be known in order for the node to determine the time and frequency hopping schedule and to join the network. Time synchronization is required to execute the time division multiple access (TDMA) protocol. The correct time is also required to correctly report and interpret detections and tracks.

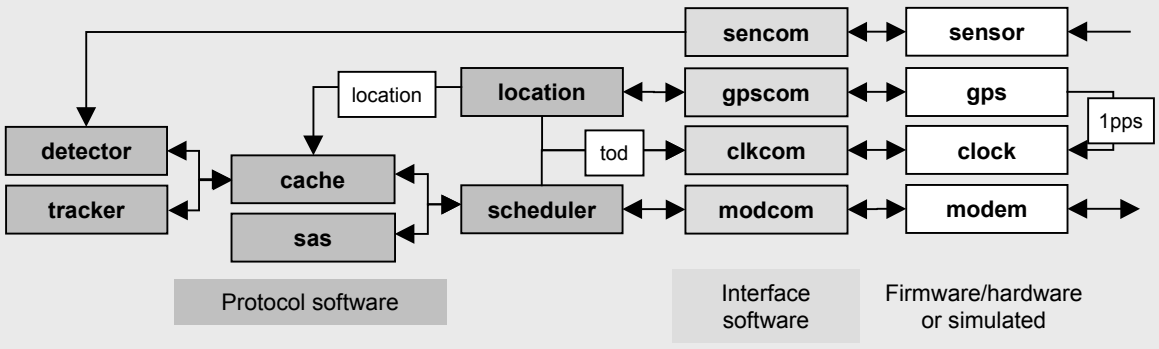

Figure 6. The major modules of the sensor protocol suite and the data flow between them.

The node clock is provided by a low accuracy crystal oscillator. This oscillator is calibrated using either the GPS receiver or by the receipt of messages from other, calibrated nodes. Because the oscillator rate drifts over time (mainly because of temperature effects), the oscillator must be frequently recalibrated. Oscillator calibration is completely automatic. In order to avoid costly mistakes, the clock calibration function is constrained to clock values between `ClockRateMinimum (19800000 Hz)` and `ClockRateMaximum (20200000 Hz)`. Before the crystal is calibrated, it is assumed to run at `ClockRateNominal (20000000 Hz)`. The current clock rate can be viewed in the parameter `ClockRate (20000000 Hz)`. The clock synchronization error can be viewed in the parameter `ClockSkew (0 Hz).`

The operation of the processor is controlled by the node clock. When the clock is operated at high speed, the processor consumes energy quickly. When the clock rate is lowered, the processor consumes much less energy. The node automatically adjusts the clock rate (sometimes several times per second) depending upon the need for processing

The commands shown in Figure 7 allow you to control and test the clock software and firmware. This list may be obtained from the node by typing `clock`.

clock log [on|off] -- turn on/off logging of clock values once per second

The `clock log on` command enables the logging of information about the state of the node clock. The `clock log off` command disables the logging of this information.

clock set [time] -- set seconds part of time

The `clock set` command sets the seconds portion of the time of day. The seconds portion of time of day must be set properly to participate in the time and frequency hopping schedule and to report detections and tracks that make sense to the other nodes and to users. Normally, the node automatically acquires this information from the GPS receiver or from another node.

clock slow -- bypass plls

The `clock slow` command lowers the processor clock rate to 10 MHz by bypassing the clock phase lock loop (PLL). This action slows the processor speed and saves considerable energy. The node automatically takes this action when no tasks are pending if `ClockSleepEnable (1)` is activated.

clock fast -- restore plls

The `clock fast` command raises the processor clock rate to 130 MHz by enabling the clock phase lock loop (PLL). This action significantly increasess the processor speed and expends considerable additional energy. This action is automatically taken by the node when tasks are ready for processing if

```
clock
clock log [on|off] -- control logging of clock messages
clock set [time] -- set seconds part of time
clock slow -- bypass plls
clock fast -- restore plls
clock sleep [interval, seconds] -- sleep for specified interval
clock bypass [on|off] -- control automatic pll bypass
clock drift -- pretend that clock drift is 0
clock on [interval, seconds between samples]
clock off
clock now -- print current time
clock deep [interval, ms] -- put the memory to sleep
```

Figure 7. The `clock` commands.

```
clock now
1135311949.375262 512 19999895 0  104
```

Figure 8. The `clock now` display.

`ClockSleepEnable (1)` is activated.

clock sleep [interval, seconds] -- sleep for specified interval

When the `clock sleep` command is issued the processor is placed in the slow mode for the specified interval. No tasks are performed.

clock bypass [on|off] -- control automatic pll bypass

The `clock bypass` command enables or disables the automatic switching of the processor clock speed. The command `clock bypass on` is equivalent to setting the parameter `ClockSleepEnable=1` and the command `clock bypass off` is equivalent to setting the parameter `ClockSleepEnable=0`.

clock drift -- pretend that clock drift is 0

The node does not transmit if the estimated clock error is greater than the allowed slot guard time. In normal operation, the clock is resynchronized with outside sources as needed to maintain the required accuracy. During testing without any outside time source, it may be desirable to allow the node to transmit without time synchronization. The `clock drift` command tells the node to pretend that the clock error is zero and that the clock drift rate is also zero.

clock on [interval, seconds between samples]

The `clock on` command specifies the automatic display of the clock parameters at the specified interval. It is equivalent to issuing the `clock now` command at the specified interval.

clock off

The `clock off` command cancels a previous clock on command.

clock now -- print current time

The `clock now` command prints the current time, the number of seconds that the node has been operating, the clock rate, clock error, and number of consecutive GPS time pulses as shown in Figure 8.

clock deep [interval, ms] -- put the memory to sleep

The `clock deep` command puts the node into deep sleep. The node memory is put into low-power, self refresh mode and the processor clock is reduced to the minimum. The node consumes very little energy in deep sleep mode. No tasks are performed in deep sleep mode.

2.1.2 Time Synchronization

The node must know the time fairly accurately in order to participate in the network. True time of day is required to participate correctly in the time and frequency hopping schedule. Time synchronization is required to execute the time division multiple access (TDMA) protocol. Timing information is derived from the GPS receiver or over the air from other nodes. There are no commands that control the operation of the time synchronization system. However, its operation is influenced by several control parameters as described below.

Because the on board clock crystal is not very accurate, it must be resynchronized with a known time source. The clock crystal is highly influenced by temperature. The expected drift rate of the clock crystal is specified with the parameter `TimeSynchDriftRate (200 ns/s)`.

The node may be configured to use either GPS time synchronization (a value of 1), over the air (OTA) time synchronization (a value of 2), or both (a value of 3) with the parameter `TimeSynchTechnique (1)`. If both GPS and OTA synchronization are enabled and both are active, the GPS information takes precedence.

In either mode, the clock drift is checked at least as often as `TimeSynchMaximumInterval (60 s)` and no more often than `TimeSynchMinimumInterval (5 s)`. It may also be checked because of external events, such as the turning on of the GPS receiver or the receipt of an incoming message.

2.1.2.1 GPS Time Synchronization

If GPS synchronization is enabled, the GPS receiver is reactivated automatically when required to maintain the required time synchronization. This reactivation is timed to maintain clock synchronization within `TimeSynchAccuracy (500 us)` of the true time. The need for resynchronization is estimated from the time of the last known synchronization, the expected crystal drift rate, and the expected time to reacquire a GPS signal and resynchronize the clock—`TimeSynchGpsReacquire (30 s)`. This parameter may be changed to suit deployment circumstances. For example, in a large, clear area, GPS reacquisition may be as short as 4 seconds and clock synchronization may be accomplished in as little as 10 seconds.

2.1.2.2 Over the Air Time Synchronization

Over the Air time synchronization computes the probable clock synchronization value from a weighted mean of all of the neighbor clocks. OTA time synchronization allows nodes to operate in areas where GPS performance is poor or to operate the GPS receiver less often to conserve energy.

The transmission time, clock accuracy, and clock drift rate are embedded in the header of every message. The receiving node compares the reported transmission time to the actual received time to determine a clock difference between its clock and the transmitter's clock. The clock differences are then weighted by the reported accuracy terms and averaged to produce a probable local clock difference from true time. If the clock difference is large and the estimated accuracy of the computed value is high, then the node adjusts its local clock.

Since nodes communicate rarely, the number of measurements over which the weighted average is computed may be small and the measurements may go stale. Inter-node time difference accuracy values are propagated forward at the estimated clock drift rate. Measurements are used only if they are more recent than `TimeSynchAgeMaximum (600 s)` to prevent the use of stale data. While clock resynchronization is underway, clock difference calculations are erratic. Therefore, the node does not use any measurements made within `TimeSynchLockout (2 s)` of a resynchronization event.

2.1.3 Task Scheduler Configuration and Test

The node operates under the control of a soft real time scheduler that manipulates the processor state to conserve energy. The processor is normally maintained in a low-power sleep state. When a pending task is scheduled for operation the processor is awakened, placed in a fast, high-power state and the task is executed. Switching between states requires approximately 25 ms. The operation of the task scheduler is intimately tied to the node clock, as the former manipulates the latter to conserve energy.

A minimum execution time, maximum execution time, and expected duration are specified for each task. The task scheduler clusters and schedules the tasks so that every task is performed before its maximum execution time and after its minimum time. The task scheduler attempts to minimize the time during

```
task
task log [on|off]
task account -- print accounting information
task reset -- reset accounting information
task list -- list active tasks
```

Figure 9. The **task** commands.

which the processor is awake and to maximize the time during which it is sleeping. In general tasks are performed at their maximum execution time if the processor is not awakened earlier for some other reason. If the processor is awake for some other reason, any task for which the minimum execution time has passed may be performed.

Several commands allow you to control the task scheduler and view information about its performance as shown in Figure 9.

task log [on|off]

The **task log** command allows you to control the logging of messages from the task scheduler. The task scheduler is very verbose. Enabling this log is not recommended unless the operation of the task scheduler is being modified.

task account -- print accounting information

As the tasks operate, the accumulated time spent performing each task is computed. This information may be displayed with the **task account** command as shown in Figure 10. Time spent on the task and the percentage of the total time spent on that task is displayed on the left. In addition, the time spent on subtasks of each task is displayed on the right.

task reset -- reset accounting information

The **task reset** command sets the accumulated time for every task to zero. This allows measurement of accumulated time during distinct periods of operation.

```
task account
Task accounting at 1135311960. 521.852680=523.856226-2.003546
   account            task time          subtask time
                     (s)      (%)          (s)      (%)
        task      0.737447  0.1413    520.801465 99.7986
     unknown      0.064458  0.0124      0.034068  0.0065
       sleep    505.932197 96.9492      0.000000  0.0000
       waste      6.360160  1.2188      0.000000  0.0000
   scheduler      5.952262  1.1406      1.439795  0.2759
         fec      1.439795  0.2759      0.000000  0.0000
      status      0.004588  0.0009      0.000000  0.0000
    detector      0.094061  0.0180      0.000000  0.0000
     compass      0.008696  0.0017      0.000000  0.0000
         gps      0.299969  0.0575      0.004728  0.0009
    location      0.000000  0.0000      0.000000  0.0000
      tsynch      0.036554  0.0070      0.000000  0.0000
        test      0.017813  0.0034      0.000000  0.0000
         sas      0.799467  0.1532      0.000000  0.0000
       cache      0.043137  0.0083      0.000147  0.0000
```

Figure 10. The **task account** display.

```
task list
Current time: 520121 1135311957.121023
tid scheduled  minimum  maximum  duration function name
  2   520123 [  519875,  520275]        1    17608 CommandWakeup
  5   520124 [  519925,  520425]        1    48188 SasWakeup
  0          [  540013,  542013]        1     c248 TimeSynchGpsActivate
  1          [  543013,  545013]        1     bb98 TimeSynchAnalyze
  3          [  628724,  638724]        1    23ea8 ExpireExecute
  4          [  520275,  521275]        1    1f7c4 DetectorRead
  6          [  592898,  594898]        1     8d08 LocationGpsActivate
  7          [  520160,  520170]        5    4e918 WakeupCheck 174
  8          [  521070,  521120]        1    51298 WakeupScheduleNextFrame
 13          [  686883,  696883]        1    68b28 StatusReport
```

Figure 11. The **task list** display.

task list -- list active tasks

The **task list** command displays the current list of tasks as shown in Figure 11. Scheduled tasks are shown at the top in the order they are scheduled to be performed and annotated with the expected execution time. Other tasks are shown at the bottom of the list. The actual task schedule is not necessarily performed as shown. Unscheduled tasks may be advanced the next time the schedule is evaluated

2.2 Location and Orientation

The commands that control the software, firmware, and hardware used to determine the node location and pointing direction are described in this section.

2.2.1 GPS Test and Configuration

The sensor node includes a GPS receiver that is used to determine the node location and the correct time. In normal operation, the sensor node activates the GPS receiver when it starts to determine the node location and to calibrate the clock crystal. The GPS receiver is then turned off to conserve node power.

The following commands allow you to control and test the GPS receiver. This list may be obtained by typing **gps** as shown in Figure 12.

```
gps
gps [on|off] -- turn gps receiver on or off
gps log [on|off] -- control logging of gps messages
gps power [on|off] -- control gps receiver power
gps tx [on|off] -- allow transmission to the gps receiver
gps buffer rx [start] [count] -- print gps rx buffer
gps buffer tx [start] [count] -- print gps tx buffer
gps set [latitude, dN] [longitude, dE] [orientation, dT] [course, dT]
   [speed, m/s] -- set location
gps satellite -- print satellite status
gps poll [NMEA message type] -- report message once
gps enable [NMEA message type] -- enable reporting of message
gps disable [NMEA message type] -- disable reporting of message
gps now -- show current location reported by receiver
gps leap -- find out what the receiver knows about leap seconds
gps location [on|off] -- toggle need for location data
gps time [on|off] -- toggle need for time data
gps test [on|off] - operate receiver in test mode
```

Figure 12. The **gps** commands.

gps [on|off] -- turn gps receiver on or off

The **gps on** command forces the GPS receiver to turn on. The receiver operates even if it is not needed for time synchronization or location determination. The **gps off** command disables the forced operation of the GPS receiver. The receiver operates when required by the time synchronization and location determination functions.

gps log [on|off] -- control logging of gps messages

The **gps log on** command enables the logging of information about the state of the GPS receiver. The **gps log off** command disables the logging of this information.

gps power [on|off] -- control gps receiver power

The **gps power on** command forces the power for the GPS receiver to be turned on. Normally the power to the GPS receiver is turned on whenever the node tries to use the GPS receiver and is turned off when the receiver is not needed. The **gps power off** command cancels this forced operation.

gps tx [on|off] -- allow transmission to the gps receiver

The **gps tx on** command allows messages to be sent to the GPS receiver. The transmit function remains active until the **gps tx off** command is issued. In normal node operation, the transmit function is enabled and disabled as required by the other commands. After initial receiver configuration, this function is normally turned off to conserve energy.

gps buffer rx [start] [count] -- print gps rx buffer

The processor communicates with the GPS firmware and hardware through a circular, dual port memory buffer. The **gps buffer rx** command allows inspection of the buffer used to communicate from the GPS receiver to the processor. The buffer is 1024 bytes long and usually contains only text messages. Occasionally, binary messages may appear in the buffer in response to specialized requests, such as the **gps leap** command request described below.

gps buffer tx [start] [count] -- print gps tx buffer

The processor communicates with the GPS firmware and hardware through a circular, dual port memory buffer. The **gps buffer tx** command allows inspection of the buffer used to communicate to the GPS receiver from the processor. The buffer is 1024 bytes long and after initial configuration it is usually empty. Occasionally, messages may appear in the buffer to implement specialized requests, such as for the **gps leap** command described below.

gps set [latitude, dN] [longitude, dE] [orientation, dT] [course, dT] [speed, m/s]

The **gps set** command sets the location of the node as if the information was obtained from the GPS receiver. It is useful during testing indoors or in other areas where a GPS signal may not be present.

gps satellite -- print satellite status

The **gps satellite** command prints a table of information about the GPS satellites known to the receiver as shown in Figure 13. The receiver must be on in order to acquire and print this information. A minimum of 3 good satellite signals and satellite ephemeris (to determine satellite azimuth and elevation) are required to compute a location and time fix. Satellite signals are good if the reported SNR is about 40 or above. The GPS receiver automatically acquires the satellite ephemeris from the satellites. This may take a few seconds or several minutes depending upon the quality of the received signal.

```
gps satellite
 7 satellites.
 sv  az  el snr
  3 273  48  48
 14 188  45  50
 15  88  61  42
 18  47  48  40
 19 313  26   0
 21 107  35  43
 22 330  73  47
```

Figure 13. The **gps satellite** display.

gps poll [NMEA message type] -- report message once

The **gps poll** command requests that the receiver respond one time with the specified National Marine Electronics Association (NMEA) message. The NMEA message set is a standard for communication with GPS receivers and is described more fully in NMEA 0183. Normally, the node requests only the GPRMC message which supplies all of the information needed for location and time determination.

gps enable [NMEA message type] -- enable reporting of message

The **gps enable** command requests that the receiver respond once per second with the specified National Marine Electronics Association (NMEA) message. The NMEA message set is a standard for communication with GPS receivers and is described more fully in **NMEA 0183** [5]. Normally, the node requests only the GPRMC message which supplies all of the information needed for location and time determination.

gps disable [NMEA message type] -- disable reporting of message

The **gps disable** command cancels the automatic generation of the specified National Marine Electronics Association (NMEA) message. The NMEA message set is a standard for communication with GPS receivers and is described more fully in **NMEA 0183** [5]. Normally, the node requests only the GPRMC message which supplies all of the information needed for location and time determination.

gps now -- show current location reported by receiver

The **gps now** command reports the location of the node as determined by the GPS receiver as shown in Figure 15. It reports the last location estimate and the weighted average of the last few measurements. As the location reported by GPS receiver continuously drifts in an area about the true location, the node uses a long term exponentially weighted average for its location estimate. A large excursion detector provides an instant reset of the average when the node is moved by more than 100 meters.

gps leap -- find out what the receiver knows about leap seconds

The **gps leap** command requests information about leap seconds from the receiver. Leap seconds are the difference between GPS time and Coordinated Universal Time (UTC). GPS time presumes that every

```
gps now
instantaneous: 37.741998 -122.419544  0.0 m/s  0.0 dT
      average: 37.741999 -122.419547  0.0 m/s  0.0 dT
```

Figure 14. The **gps now** display.

```
gps leap
Leap second report: offset=14 s, applied=1
```

Figure 15. The **gps leap** display.

day is exactly the same length. UTC on the other hand inserts or removes leap seconds to account for the inexact rotational speed of the earth.

Unfortunately the GPS receiver does not immediately now the difference between GPS time and Coordinated Universal Time (UTC). Learning this offset, called leap seconds, from the satellite may take up to 12 minutes. Until this constant is discovered from the satellites, the sensor node uses the value **GpsLeapSecond (14 s)**. This value is correct as of 1 January 2006.

The **gps leap** command asks the receiver if it knows the number of leap seconds, and if so, prints the value as shown in Figure 15.

gps location [on|off] -- toggle need for location data

The **gps location on** command activates the GPS receiver as if it was requested by the location determination function. The receiver may be turned back off if the location determination function decides that it is not necessary. The **gps location off** command disables the GPS receiver as if it was requested by the location determination function. If that function decides that the receiver should be operating, it may be reactivated almost immediately.

gps time [on|off] -- toggle need for time data

The **gps time on** command activates the GPS receiver as if it was requested by the time synchronization function. The receiver may be turned back off if the time synchronization function decides that it is not necessary. The **gps time off** command disables the GPS receiver as if it was requested by the time synchronization function. If that function decides that the receiver should be operating, it may be reactivated almost immediately.

gps test [on|off] – operate receiver in test mode

The **gps test on** command activates the GPS receiver in test mode. The test mode continues to operate the receiver even when the normal node functions would turn it off. The node continues to make clock rate and skew measurements but does not adjust the clock. This allows accurate measurement of time synchronization error on the node itself. If either the location determination function or the time synchronization function activate the receiver, the mode switches to normal mode as long as those functions remain active. Then it switches back to test mode. The **gps test off** command disables the test mode.

2.2.2 Compass Test and Configuration

The node includes a 3-axis magnetometer that is used to determine the node orientation. The node orientation is the direction in which the PIR sensor is pointing. Compass readings are acquired automatically once per second whenever the GPS receiver is operating. The instantaneous compass reading is not very accurate. Therefore, the node averages multiple readings to produce a usable value.

The magnetometers must be reset and cleared before a sample is acquired. The reset procedure is accomplished by providing a number of alternating, short duration pulses across the magnetometer just prior to acquiring the sample. The number of pulses is one more than **CompassPulse (15)**, the wait between pulses is one more than **CompassWait (0 ms)**, and the delay from the last pulse until data acquisition is one more than **CompassDelay (0 ms)**. The node applies power to the magnetometer

```
compass
compass [on|off] -- control compass power
compass now -- acquire one compass sample
compass continuous [on|off] -- continuously acquire compass data
compass calibrate [on|off] -- control compass calibration
compass set [orientation, deg true] -- set node orientation
```

Figure 16. The **compass** commands.

CompassTurnOnDelay (100 ms) before starting the pulsing sequence. It is unlikely that any of these parameters should be changed.

The compass requires calibration before use. This calibration has been performed in the factory before delivery of the node. If it appears that recalibration is required in the field, the procedure in Section 3 should be followed.

The following commands allow you to control and test the node compass. This list may be obtained by typing **compass** as shown in Figure 16.

compass [on|off] -- control compass power

The **compass on** command forces the compass power to be on. Normally the node activates the compass power supply whenever it is needed and deactivates the supply when the compass is not in use. The **compass off** command cancels this forced operation.

compass now -- acquire one compass sample

The **compass now** command acquires one sample from the compass and displays the result. The raw sample values, the scaled values, the instantaneous computed orientation, and the averaged compass reading are displayed as shown in Figure 17.

compass continuous [on|off] -- continuously acquire compass data

The **compass continuous on** command begins continuous acquisition of samples from the compass. Compass samples are acquired at an interval of **CompassInterval (200 ms)**. Sampling continues until the **compass continuous off** command is issued.

compass calibrate [on|off] -- control compass calibration

The **compass calibrate** command controls compass calibration. When the **compass calibrate on** command is issued the node begins to accumulate sample data. The minimum and maximum sample values are recorded on each channel independently until the **compass calibrate off** command is issued. Then the range and midpoint of each channel is computed. The channel with the smallest range is assumed to be vertical. The other two channels are used as the x-axis and y-axis of a 2-dimensional compass.

```
compass now
   difference:   1603      643    1426      592    1955      7a3
       scaled: -14000 ffffc950   -1069 fffffbd3   -2032 fffff810
instantaneous: 218 dM  228 dT
      average: 219 dM  229 dT
```

Figure 17. The **compass now** display.

The compass calibration values can be viewed in the parameters: **CompassUp (0)**, **CompassCenter[0] (-2779)**, **CompassRange[0] (235)**, **CompassCenter[1] (-459)**, **CompassRange[1] (1782)**, **CompassCenter[2] (-931)**, and **CompassRange[2] (1340)**. The values shown are from a particular node—there are no default values for these parameters.

No compass samples are acquired as a result of this command. You must also issue the **compass continuous on** or the **compass now** command to actually acquire samples. Please see Section 3 for the complete compass calibration procedure.

compass set [orientation, deg true] -- set node orientation

The **compass set** command can be used to set the node orientation in the same manner as if it was acquired from a calibrated compass. The orientation should be supplied in degrees true, not magnetic.

2.2.3 Location Determination

The node location is determined using the GPS receiver. The node orientation is determined using the built-in compass. The node must know its location and orientation in order to participate in detection and tracking. There are no commands that control the operation of the node location system. However, its operation is influenced by several control parameters as described below.

The GPS receiver and compass are operated for a minimum duration of **LocationStartup (300 s)** when the node starts to allow you to position the node after installing the batteries. The GPS receiver is operated until a minimum **LocationCountRequired (10)** reports are obtained and until the mean value of successive reported locations is in the same place to an accuracy of **LocationAccuracyRequired (5 m)**. Occasionally, during operation the GPS receiver is reactivated to check the node location. This happens at increasing long intervals of n***LocationRecheckMinimum (300 s)**, where n is the number of times the location has been checked, up to a maximum of **LocationRecheckMaximum (7200 s)**.

In order to prevent long operation of the GPS receiver when insufficient satellite signals are present, a timeout value is imposed. The first time the receiver attempts to obtain a location fix, the timeout value is set at **LocationNoFixMinimum (60 s)**. If no fix is obtained within this time, the GPS receiver is turned off and another attempt is scheduled at the normal recheck interval. On subsequent attempts, the timeout value is doubled up to a maximum of **LocationNoFixMaximum (600 s)**.

The node location record is updated when the location or orientation has changed sufficiently or when enough time has elapsed since the last update. Udates are made when the location changes by more than **LocationReportDistanceFactor (5 m)**, the orientation changes by more than **LocationReportOrientationFactor (10 d)**, or more than **LocationReportTimeFactor (3600 s)** has elapsed. A combination of partial changes in more than one of these parameters may also trigger a record update.

The node orientation is measured with the built-in magnetic compass. The orientation is converted from degrees magnetic to degrees true by adding **LocationMagneticDeclination (-1000 d)**. You may enter the correct magnetic declination for your area of deployment, or you may allow the node to calculate a rough estimate using the bearing between its current location and the estimated location of the magnetic north pole stored in **NorthPoleLatitude (79.74 dN)** and **NorthPoleLongitude (-71.78 dE)**. If the magnetic declination is left at its default value of –1000, the node calculates the declination the first time a location fix is obtained. The rough estimate is sufficient for most deployments.

The node location and orientation may be viewed by executing a query against the table **location** (see Section 2.6.3.3). It may also be viewed in the parameters **LocationLatitude (-1000 dN)**, **LocationLongitude (-1000 dE)**, and **LocationOrientation (-1000 dT)**. There are no default values for these parameters.

2.3 Communication

The commands that control the software, firmware, and hardware used to communicate between nodes are described in this section.

2.3.1 Modem Test and Configuration

An intergated modem and 2.4 GHz radio frequency (RF) front end chip is at the heart of the sensor node communication system. The modem chip is supported by a power amplifier to increase transmit power and a low noise amplifier to increase receive sensitivity. A small antenna is integrated inside the sensor node case, making for a very small, high performance communication system. The sensor node communication system exhibits a link margin of approximately 110 dBm.

Three parameters influence the normal operation of the communication system. Because of frequency hopping, the sensor node typically uses a different frequency for each communication opportunity. Consequently, the RF front end must be retuned before every transmit or receive operation. The node retunes the front end **ModemTuneTime (150 us)** before every operation. It is not recommended that you change this parameter.

The parameter **ModemTransmitPower (-5 dB)** controls the output power of the sensor node. It can be adjusted in three steps: -15 dB, -5 dB, or 0 dB. Higher power levels extend the range of the communication system, but may compromise overall network performance by causing additional interference if that range is not required. Higher power levels also consume additional energy, thus shortening node lifetime.

The modem receive sensitivity can be adjusted by the parameter **ModemReceiveSensitivity (-80 dB)**. This parameter can be adjusted in 4 dB increments over the range from –90 dB to 0 dB. The receive sensitivity is specified at the input of the modem after the low noise amplifier. The low noise amplifier supplies an additional 10 dB of receiver gain. Increased range can be obtained by lowering the receive sensitivity at the expense of additional false receptions from noise in the environment.

The default parameter values for transmit power and receive sensitivity provide a range of approximately 20 meters at a node height of 15 cm. Raising transmit power to 0 dB and lowering the receive sensitivity to –90 dB provides a range of approximately 60 m at the same deployment height. With higher node deployments the corresponding ranges are much greater—approximately 100 m and 400 m respectively.

The following commands allow you to control and test the modem and associated circuitry. This list may be obtained by typing **modem** as shown in Figure 18.

```
modem
modem [on|off] -- control state of modem
modem read [minimum register, hex] [maximum register, hex]-- read registers
modem write [register, hex] [value, hex] -- write register
modem log [on|off] -- control state of modem debug messages
modem lna [on|off] [restore interval, ms] -- control the low noise amplifier
modem pa [on|off] [restore interval, ms] -- control the power amplifier
modem configure -- configure the modem
modem test [on|off] -- continuously transmit test pattern
modem noop -- send noop to the firmware
modem buffer rx [start] [count] -- print modem rx buffer
modem buffer tx [start] [count] -- print modem tx buffer
modem halt -- reset everything about the modem
```

Figure 18. The **modem** commands.

modem [on|off] -- control state of modem

The **modem on** command activates the modem and modem firmware. The modem is automatically configured. The **modem off** command resets the firmware and turns off power to the modem.

modem read [minimum register, hex] [maximum register, hex] -- read registers

The **modem read** command reads the specified list of modem registers and displays their values as shown in Figure 19. The modem registers are described in **CC2400 Preliminary Data Sheet** [2].

modem write [register, hex] [value, hex] -- write register

The **modem write** command sets the value of the specified modem register. The modem registers are described in **CC2400 Preliminary Data Sheet** [2].

modem log [on|off] -- control state of modem debug messages

The **modem log on** command enables the logging of information about the state of the modem. The **modem log off** command disables the logging of this information.

modem lna [on|off] [restore interval, ms] -- control the low noise amplifier

The **modem lna** command influences the state of the low noise amplifier in the receiver chain. In normal operation, the low noise amplifier is turned on **ModemTuneTime (150 us)** in advance of a reception and turned off at the end of the reception. If the **modem lna on** command is issued the low noise amplifier is forced on. The low noise amplifier remains on for the specified interval or until the **modem lna off** command is issued.

modem pa [on|off] [restore interval, ms] -- control the power amplifier

The **modem pa** command influences the state of the power amplifier. In normal operation, the power amplifier is turned on **ModemTuneTime (150 us)** in advance of a transmission and turned off at the end of the transmission. If the **modem pa on** command is issued the power amplifier is forced on. The power amplifier remains on for the specified interval or until the **modem pa off** command is issued. The power amplifier is a significant energy consumer and should not be left in the active state for a significant period of time.

modem configure -- configure the modem

The **modem configure** command executes the sequence of modem register write operations required to configure the modem for normal operation. In normal operation, this command is executed automatically when the modem is turned on.

modem test [on|off] -- continuously transmit test pattern

The **modem test** command causes the modem to continuously transmit a test pattern. This command facilitates the measuring of transmit power by an external spectrum analyzer.

```
modem read 6 8
mr[06] = 7fe6
mr[07] = 0000
mr[08] = 1450
```

Figure 19. The **modem read** display.

modem noop -- send noop to the firmware

The **modem noop** command places a noop command into the command buffer for execution by the firmware. This command may be used to test the flow of commands from the processor, through the transmit buffer to the firmware, and then back through the receive buffer to the processor. The noop command does not use or test the modem.

modem buffer rx [start] [count] -- print modem rx buffer

The modem firmware communicates commands and data to the processor through a 4096-byte, dual-ported circular buffer. The firmware writes commands and data into this buffer as the commands are executed; the processor reads the commands and data. The **modem buffer rx** command displays the contents of this buffer. Each command consists of 12 bytes and is followed immediately by any applicable data.

modem buffer tx [start] [count] -- print modem tx buffer

The processor communicates commands and data to the modem firmware through a 4096-byte, dual-ported circular buffer. The processor writes commands and data into this buffer; the firmware reads the commands and data and then executes the commands. The **modem buffer tx** command displays the contents of this buffer. Each command consists of 12 bytes and is followed immediately by any applicable data.

modem halt -- reset everything about the modem

The **modem halt** command resets the modem, the modem firmware, and the modem software. The modem is turned off. If the communication scheduler is running, it will probably reactivate and reconfigure the modem within one second.

2.3.2 Link Test

The following commands allow you to test link quality between any pair or groups of nodes. This list may be obtained by typing **link** as shown in Figure 20. The link commands override any normally scheduled slot usage.

link rx [time slot] [frequency channel] [repeat count]

The **link rx** command causes the node to receive messages for the specified number of frames on the specified frequency channel and in the specified time slot. The **link rx** command overrides any normally scheduled slot usage. As the command executes the node may print a single character or a single

```
link
link rx [time slot] [frequency channel] [repeat count]
link tx [time slot] [frequency channel] [repeat count] [message]
link mrx [number of slots] [minimum slot] [slot increment] [minimum
  channel] [channel increment] [repeat count]
link mtx [number of slots] [minimum slot] [slot increment] [minimum
  channel] [channel increment] [repeat count] [message length]
link stop
link log [on|off] -- log debug messages
link print [on|off] -- single line, real-time status
link scroll [on|off] -- single character, real-time status
link summary -- print summary statistics
```

Figure 20. The **link** commands.

line status message about each receive opportunity as it occurs. See the commands **link print** and **link scroll**.

link tx [time slot] [frequency channel] [repeat count] [length]

The **link tx** command causes the node to transmit messages of the specified length for the specified number of frames on the specified frequency channel and in the specified time slot. The messages are filled with random numbers. The **link tx** command overrides any normally scheduled slot usage.

link mrx [number of slots] [minimum slot] [slot increment] [minimum channel] [channel increment] [repeat count]

The **link mrx** command is similar to the **link rx** command. The only difference is that this command sets up the specified number of slots using the specified pattern.

link mtx [number of slots] [minimum slot] [slot increment] [minimum channel] [channel increment] [repeat count] [message length]

The **link mtx** command is similar to the **link tx** command. The only difference is that this command sets up the specified number of slots using the specified pattern.

```
link rx 100 0 10
Receiving link test on slot 100, frequency=0, repeat=10
`              time`    raw`  it`is`   s`  f`  qs` qf` e` len` td`rssi`
`1135312823.917750`     48`   0` 0` 100`  0` 183` 33` 0` 120`  0` -27`
`1135312824.182750`     49`   1` 0` 100`  0`  36` 46` 0` 120`  0` -27`
`1135312825.307750`     50`   2` 0` 100`  0`  61` 41` 0` 120`  0` -27`
`1135312826.452750`     51`   3` 0` 100`  0`  90` 21` 0` 120`  0` -33`
`1135312827.452751`     52`   4` 0` 100`  0`  90` 23` 0` 120`  1` -31`
`1135312828.102751`     53`   5` 0` 100`  0`  20`  4` 0` 120`  1` -38`
`1135312829.567750`     54`   6` 0` 100`  0` 113` 33` 0` 120`  0` -28`
`1135312830.352749`     55`   7` 0` 100`  0`  70` 33` 0` 120` -1` -26`
`1135312831.677751`     56`   8` 0` 100`  0` 135` 42` 0` 120`  1` -26`
`1135312832.562751`     57`   9` 0` 100`  0` 112` 20` 0` 120`  1` -32`
```

time is the time at which the receive opportunity occurred.
raw is the number of seconds since the node started.
it is the iteration counter on the current test
is is the slot index for multiple slot tests.
s is the scheduled time slot.
f is the scheduled frequency channel.
qs is the time slot after scrambling.
qf is the frequency channel after scrambling.
e is the number of corrected errors if it is greater than or equal to 0. –1 indicates that no signal was received. –2 indicates that the header and trailer were not identified. –4 indicates that the packet was uncorrectable.
len is the length of the incoming packet for successful receptions. For unsuccessful receptions it is the length of the acquired data.
td is the time difference between the transmitter clock and the receiver clock for successful packets. For unsuccessful packets, it is the time from when the receiver turned on until the carrier detect indicator went active.
rssi is the measured receive signal strength. A value of –999 indicates no signal.

Figure 21. The **link print** display.

```
++++++++++1+++++++H2++U4.7+++++4++++++++1++++H++++++++++++++++++++
```

U indicates an uncorrectable packet.
H indicates a packet for which the header could not be identified.
. indicates that no signal was received.
+ indicates a packet received without error.
1-16 indicates the number of errors corrected in the incoming packet.

Figure 22. The **link scroll** display.

link stop

The **link stop** command halts any ongoing **link rx** or **link tx** operation.

link log [on|off] -- log debug messages

The **link log on** command enables the logging of information about the link commands. The **link log off** command disables the logging of this information.

link print [on|off] -- single line, real-time status

The **link print on** command prints a single line about each receive operation as it happens as shown in Figure 21. The **link print off** command cancels this printing. The single line display is on by default.

link scroll [on|off] -- single character, real-time status

The **link scroll on** command prints one or two characters about each receive operation as it happens as shown in Figure 22. The **link scroll off** command cancels this printing. The scrolling display is off by default.

link summary -- print summary statistics

The **link summary** command prints summary statistics about the most recent link operation as shown in Figure 23.

```
link summary
LINK TEST
now=1135312838 start=1135312823 end=1135312833 duration=10
received packets: 10 of 10, 1.000
slot index=0, time slot=100 frequency=0
no signal packets: 0 of 10, 0.000
no header packets: 0 of 10, 0.000
no trailer packets: 0 of 10, 0.000
uncorrected packets: 0 of 10, 0.000
     good packets: 10 of 10, 1.000
corrected packets: 0 of 10, 0.000
                  mean   minimum  maximum
  length (B)   120.000       120      120
   rssi (dBm)  -29.500       -38      -26
   error (B)     0.000         0        0
error histogram [0:16] 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rssi histogram [-99:0] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
  0 0 0 0 1 1 1 0 0 1 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0
```

Figure 23. The **link summary** display.

2.3.3 Communication Scheduler

The nodes communicate using a time division multiple access (TDMA) protocol. The operation of the TDMA protocol is under the control of the communication scheduler that matches queued messages to communication opportunities. There are no commands that control the operation of the operation of the communication scheduler. However, its operation is influenced by several control parameters as described below.

The TDMA protocol uses a 1 second frame synchronized to the beginning of each second as shown in Figure 24. A short interval at the beginning and end of each frame are ignored—**FrameGuardMinimum (1000 us)** and **FrameGuardMaximum (1000 us)** respectively. The rest of each frame is divided into a number of fixed length data slots of length **SlotLength (4000 us)** interspersed with acknowledgement slots of length **AckLength (2500 us)**.

To allow for time synchronization error, the transmitter is turned on **SlotGuardTime (1000 us)** after the nominal beginning of each data slot. The receiver is operated looking for signal power for a minimum of **SlotCarrierDetectTime (2000 us)** after the nominal beginning of each slot.

Acknowledgements are timed from the beginning of the transmitted data. Acknowledgements are delayed **AckDelay (300000 us)** from the beginning of the data transmission. Because the timing of an acknowledgment is well known to both the transmitter and the receiver, the guard time and carrier detect times are considerably shorter— **AckGuardTime (10 us)** and **AckCarrierDetectTime (100 us)** respectively—than for data slots. Multiple acknowledgements may occur in a single acknowledgement slot. These acknowledgements are separated by **AckSpacing (300 us)**.

The communication scheduler executes both time slot and frequency channel hopping if these features are enabled by the parameters **ScrambleTime (1)** and **ScrambleFrequency (1)** respectively. A value of 1 enables scrambling; a value of 0 disables scrambling. On each frame, the scheduled time slot and

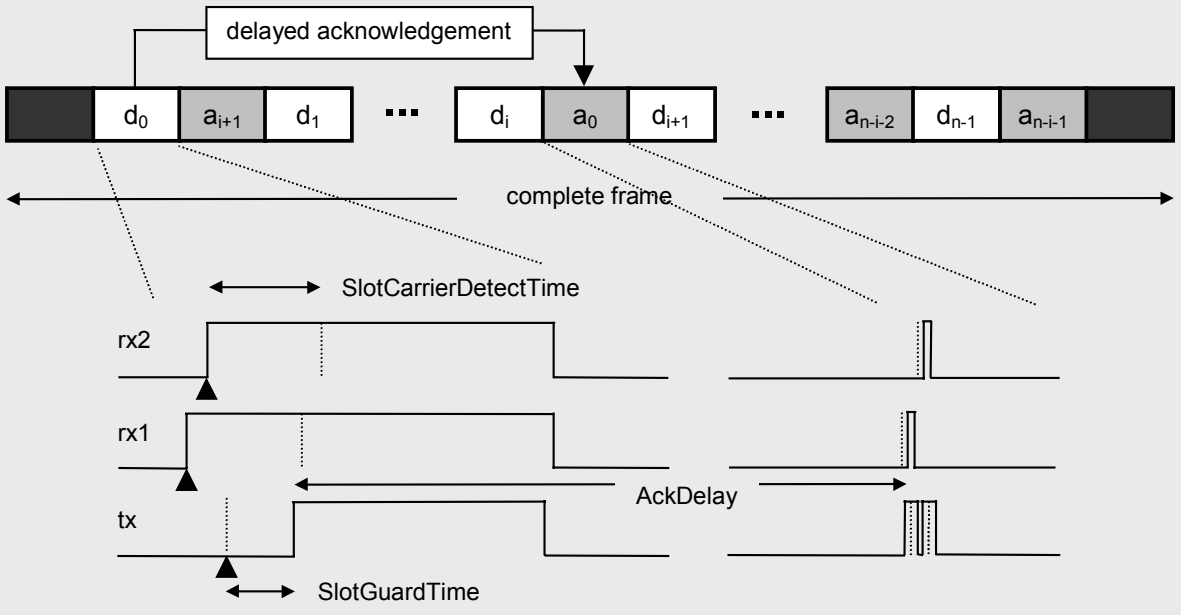

Figure 24. The frame plan alternates between data and acknowledgement slots labeled d_i and a_i respectively. A small portion at the beginning and the end of the frame are unused. Data slots have long carrier detect and guard times to allow for time synchronization errors. Acknowledgements are substantially delayed from the data transmission but tightly timed, so corresponding carrier detect and guard times are short. This allows acknowledgement from multiple receivers within a single slot.

frequency channel are mapped onto a different time slot and frequency channel for execution. Time slot mapping covers the entire frame. Frequency channel mapping may be restricted to a portion of the allowed band with the parameters **FrequencyBase (2)** and **FrequencyMany (50)**. The time slot scrambling sequence is 4001 points long; the frequency scrambling sequence is 3001 points long. As both of these numbers are prime, the combined sequence runs for 12,007,001 frames (138 days) without repeating. The sequence is aligned with time of day and based on a secret key stored in the parameter **ScrambleKey (1)**.

Fan-in receive slots are commonly used by the sensor node during periods of low data traffic. Any of the neighbor nodes may elect to try to communicate to the receiving node during this slot. The sensor nodes utilize a dual mode backoff mechanism to manage access to these slots as shown in Figure 25. The comb algorithm is used by nodes that have completed the neighbor handshake; the probabilistic backoff is used by other nodes.

The comb transmission probability adapts to the collision rate. Transmission is allowed when the destination index assigned to the transmitting node by the common receiver is equal to the frame number (time) modulo the comb value. The comb value is initially set to 1, which allows transmission in every frame. If a collision occurs the comb value is increased up to a maximum of **CombMaximum (16)**. The comb value is decreased after double the comb value or **CombCountMaximum (16)** frames have occurred. The comb algorithm allows more organized (and faster) access control under heavy contention, while allowing immediate transmission under light load.

Each transmitting node maintains a transmission backoff probability for any contention slot. This probability starts at 1.0. With each collision it is decreased by multiplication by **BackoffDecrease (0.9)** raised to the power of **NeighborExpected (5)** to a minimum value of **BackoffMinimum (0.1)**. With each non-collision the probability is raised by addition of **BackoffIncrease (0.05)**. On each frame the node may transmit if a random number is greater than the backoff probability.

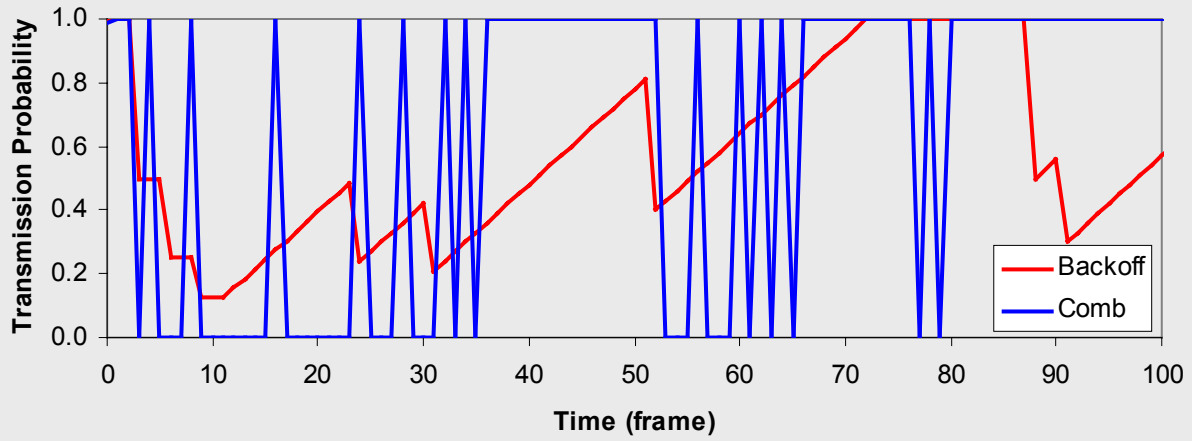

Figure 25. The communication scheduler manages contention slots through a dual mode backoff mechanism. The comb algorithm is used by nodes that have completed the linkup handshake; the probabilistic backoff algorithm is used by other nodes. The comb algorithm takes on only 2 values: one relatively high (in this example about 0.99) and the other 0. The high value is taken when the frame number modulo the comb size is equal to the node's mask index for communication to the receiving node for the slot. The comb size is increased by collisions and decreases after a short period. The backoff probability drops by about half on each collision. It climbs slowly when there is no collision. The comb algorithm organizes the competing transmitters so that they take turns communicating to the designated receiver. It recovers quicker than the probabilistic backoff algorithm and provides more communication opportunities with fewer collisions.

The node does not transmit messages if time of day is unknown or if time synchronization is lost. It continues to receive at the scheduled opportunities in the hope that it will hear a message from another node and regain knowledge of time of day and time synchronization.

Since the GPS receiver and the transmitter both consume large amounts of power, the node may not be able to operate both simultaneously when the batteries are partially consumed. The node will not transmit when the GPS receiver is operating if the node battery voltage is below `TxVoltageMinimum (2100 mV)`.

2.3.4 Synchronous/Asynchronous (SAS) Medium Access Protocol

The communications system operates a time division multiple access (TDMA) scheme with support for dynamic frequency selection (DFS). The sas medium access protocol is capable of rapidly switching from asynchronous, contention service to synchronous, clear-channel service depending upon node state, data load, and data criticality, The hybrid approach allows optimization for performance when that is critical and optimization for low power consumption when performance is not important.

The sas protocol schedules the use of time slots for a specific purpose. Slot scheduling is fully distributed—any node can schedule a slot at any time based upon its knowledge of the state of the neighboring network, its load, interference statistics, data criticality, and other factors.

Slots may be scheduled for communication to or from a specific node, subset of nodes, or all nodes. Before scheduling a slot, the node checks its knowledge of slot schedules for conflicts. After scheduling a slot, the node informs its neighbors, and so on. Slots may be scheduled for transmit or receive and may be scheduled for point-to-point, fan-in (1 receiver from multiple potential transmitters) or fan-out (1 transmitter to multiple receivers) operation.

Schedule conflicts are rare since the number of slots is far greater than the number required at a particular time. Conflicts are resolved by having one of the conflicted nodes change its schedule. Conflicts occur when 2 nodes in a neighborhood simultaneously schedule the use of the same slot, or more often during network formation when 2 nodes have scheduled the use of the same slot before they become connected. Slots are scheduled with an effective first and last use time and are only used within that time period. Slots scheduled by the transmitter are not used until acknowledged by the intended receiver. Time slots and frequency channels are scrambled on every frame based on a shared secret key and time of day.

The least expensive type of slot is an asynchronous, fan in, receive slot—only a single receiver is on when there is no data to transmit. The most expensive slot is a synchronous, fan out, transmit slot to all neighbors since many receivers are on even if no data is transmitted. Conversely, such a receive slot is capable of moving very little data; such a transmit slot is capable of moving a lot of data efficiently and with low latency. The sas protocol switches operation between the types of slots to best meet the current communications needs of the nodes.

The sas protocol may be configured to alter the slot schedule depending upon a number of types of events. By far the most interesting event type for a sensor network is a cue from an external process. The sas protocol reacts to a cue from the surveillance system that sensor data may be forthcoming to produce a broadcast slot. With approximately 1-2 seconds of lead-time, the sas protocol can reconfigure the network neighborhood from a laid-back, barely active state to a state supporting the rapid movement of sensor data. These slots are created with a valid interval determined by the queuing application.

Operation of a node begins by scheduling `ListenMany (1)` fan in, receive slots. Every so the sas protocol formats a message containing its receive slot information, and announces this in an unscheduled use of a randomly selected, unused slot. If any neighboring node happens to hear this transmission, it now knows how to talk back to the originating node. On subsequent frames it attempts to contact the originating node with a message containing its own receive slots. Once 2-way communication is

established, the nodes exchange other information about themselves and their neighbors, so that both nodes may build knowledge of activities in a neighborhood about themselves.

Upon learning of a neighbor of a neighbor, the node attempts to contact that neighbor of a neighbor for **NonRetry (10)** at intervals of n***NonInterval (10 s)** where n is the number of previous contact attempts. This procedure establishes additional links more quickly and more efficiently than the unscheduled search that discovered the first link.

The number of receive slots is constrained to lie between **ListenManyMinimum (1)** and **ListenManyMaximum (5)**. It is increased if there is a high collision rate on the existing receive slots and decreased if the number of neighbors is large and the collision rate is low. The first receive slot is created for **ListenValid (86400 s)**. A new receive slot is created a minimum of **ReallocateTime (60 s)** before the existing slot expires. Receive slots created to deal with temporary high collison rates are created for **ListenValidTemporary (60 s)**. The number of receive slots is evaluated no more often than **ListenInterval (10 s)**. After network formation, the network typically rests with one receive slot for each node and little data traffic.

The probability of performing an announcement on each frame is dependent on the number of neighbors and the time the node has been operating. The probability is **AnnounceProbabilityConstant (1)** times **AnnounceProbabilityFactor (0.8)** raised to the power of the number of neighbors plus the operating time divided by **AnnouceTimeFactor (300)**. It is further constrained to lie between **AnnounceProbabilityMinimum (0.01)** and **AnnounceProbabilityMaximum (1)**.

If a high outgoing load exists to a particular node, the sas protocol can create a point to point transmit slot to deal with the load. Loads less than or equal to **LoadIgnore (20 packet)** are ignored. Higher loads create a slot if the ratio of load to slots is higher than **LoadRatio (10 packet/slot)**. Point to point transmit slots are valid for **LoadInterval (60 s)**. Neighbors are not generally informed of point to point slots since their lifetime is so short and the probability of interference is very low. The number of point-to-point slots between any pair of nodes is constrained to be less than **TransmitManyMaximum (1)**. The actual number of point to point slots may be (and generally is) zero.

Keep alive messages are generated to each neighbor every **KeepAliveInterval (300 s)**. Keep alive messages contain the node's contact information—node id and destination mask and receive slot definition. Keep alive messages represent overhead and their use should be minimized. Neighbors are dropped if they have not been heard from for **NeighborDropInterval (900 s)**.

The quality of links to neighbor nodes is continuously evaluated by computing an exponential average of the number of expected acknowledgements received. Node quality ranges between 0 and 1000. When an acknowledgement is received, the new value of 1000 is added to the average, when an expected acknowledgement is not received the value 0 is added to the average. The exponential weighting constant is **NodeQualityResponse (10)** in an allowed range of 0 to 100. When node quality falls below **NodeQualityMinimum (100)** and at least 20 communication attempts have occurred the neighbor node is suspended from active use for n*120 seconds where n is the retry counter. The retry counter is increased every time the link is suspended up to a maximum of 5. It is decreased for every 900 seconds that a link is good.

The following commands allow you to control and test the sas protocol software. This list may be obtained by typing **sas** as shown in Figure 26.

sas listen [slot number] [frequency channel] [repeat count]

Normally, the sas protocol uses a randomly selected time slot and frequency to listen for transmissions from known and unknown neighbor nodes. The **sas listen** command forces the sas protocol to use the specified slot and frequency channel to listen for messages. The forced use continues for the specified number of repeat intervals or seconds. The forced listen slot is created in addition to any other randomly

```
sas
sas listen [slot number] [frequency channel] [repeat count]
sas announce [slot number] [frequency channel] [repeat count]
sas [on|off] -- control sas functions
sas critical [duration, s] [other node mask]
```

Figure 26. The `sas` commands.

created listen slots. This command may be useful to speed network formation during testing, by instructing one node to listen in a slot on which another node is known to be making announcements.

sas announce [slot number] [frequency channel] [repeat count]

Normally, the sas protocol uses a randomly selected time slot and frequency for each announcement message. The announcement messages allow the nodes to form a network by broadcasting communication schedule information to previously unknown neighbors. The `sas announce` command forces the sas protocol to use the specified slot and frequency channel for announcement messages. The forced use continues for the specified number of repeat intervals. Normally the repeat interval is 1 second or longer, depending upon the number of neighbors that the node has already acquired. This command may be useful to speed network formation during testing, by instructing one node to make announcements in a slot on which another node is known to be listening.

sas [on|off] -- control sas functions

The `sas on` command starts the sas protocol. The `sas off` command stops the operation of the sas protocol. Normally, the sas protocol is started automatically when the node begins execution.

sas critical [duration, s] [other node mask]

The `sas criticial` command declares a critical data event. A critical data event causes the sas protocol to create a clear channel, broadcast slot for communication to the specified nodes. This is the same operation as is performed by the tracker when it appears that a target track is approaching the node.

2.3.5 Message Test

The network of sensor nodes works by interchanging messages. These messages may be control information or they may be high-level information products. The delivery of all types of messages is under the control of the communication scheduler. The scheduler matches messages with the communication opportunities created by the sas protocol. Messages are delivered to neighboring nodes and acknowledged. The status of messages is communicated back to the sending application by the communication scheduler.

The following commands allow you to create and send special messages to test the scheduler or to verify the operation of the network.

send [destination mask or node id, hex] [repeat] [interval, seconds] [message]

The `send` command allows you to send messages from one node to another node or to a group of nodes. These messages are entered into the communication scheduler queue in the same way as normal data and control messages. They are processed in turn by the scheduler, are sent in the normally scheduled time and frequency slots, and are acknowledged as appropriate.

The destination of the message may be a single node or a group of nodes. A group of nodes is specified as the destination by entering a destination mask. Each bit in the mask corresponds to a single node. The

```
send a2d7 hello
PA: 1148068335.784928 sn=67 queue= 8 a2d7
PT: 1148068335.785707 sn=67 destination=8 length=7 message=" hello"
PA: 1148068337.136532 sn=67 send= 8 a2d7 queue= 8 a2d7
PA: 1148068337.398522 sn=67 send= 8 a2d7 ack= 8 a2d7 queue=0 DONE
```

Figure 27. The trace of a message sent with the **send** command.

correspondence between nodes and bits may be discovered by inspecting the **_node** table. Bits are dynamically assigned as the nodes establish links and may be different from one incarnation to another.

A single node may be specified as the destination by specifying a destination mask with only 1 bit set or the actual node id may be specified. If a node id is given, it is converted to the proper destination bit mask before queueing the message for delivery.

The repeat count and interval parameters are optional. If given the message is repeated for the specified number of times at the specified interval. When the repeat count is greater than 1, an iteration counter is appended to the end of each message. It counts down from repeat-1 to 0.

The message may be any text.

The status of the messages is displayed as they are sent and acknowledged as shown in Figure 27. When the message is queued for transmission a **PT** line is printed. When the status of a message changes a **PA** line is printed. The **PA** line shows the nodes for which the message is queued, the nodes for which it has been dropped, the nodes to which it has bent sent, and the nodes which have acknowledged the message.

prompt [destination mask or node id, hex] [repeat] [interval, seconds] [message]

The **prompt** command acts in the same manner as the **send** command except that it also causes each of the destination nodes to send the same message back to the originating node. Since data messages and acknowledgements are handled differently by the scheduler, it is possible that data messages may be transferred correctly while acknowledgements are not or vice versa. The **prompt** command allows testing of data and acknowledgement transfers in both directions. The **PR** line shows the receipt of a return message from one of the other nodes. The example in Figure 28 shows the use of the **prompt** command.

```
prompt f hello
PA: 1148334237.424227 sn=1216 queue=f 8ab2 6ae8 ba47 8918
PT: 1148334237.425008 sn=1216 destination=f length=7 message=" hello"
PA: 1148334237.597624 sn=1216 send=d 8ab2 ba47 8918 queue=f 8ab2 6ae8 ba47
  8918
PR: 1148334238.257280 source=8918 length=50 message="t=1148334237.547967
  dt=-33 rssi=-81 error=0 hello"
PA: 1148334239.970566 sn=1216 send=d 8ab2 ba47 8918 ack=1 8ab2 queue=e 6ae8
  ba47 8918
PA: 1148334239.973125 sn=1216 send=d 8ab2 ba47 8918 ack=4 ba47 queue=a 6ae8
  8918
PA: 1148334239.975621 sn=1216 send=d 8ab2 ba47 8918 ack=8 8918 queue=2 6ae8
PR: 1148334240.793845 source=8ab2 length=50 message="t=1148334237.547964
  dt=-36 rssi=-77 error=0 hello"
PR: 1148334242.954545 source=ba47 length=50 message="t=1148334237.548208
  dt=208 rssi=-84 error=0 hello"
```

Figure 28. The trace of a message sent with the **prompt** command.

2.3.6 Data Cache Interface

The data cache interface is the standard method of extracting information from the network of sensor nodes. The **query** command provides access to the distributed data cache maintained on the network. Data records may be selected, inserted, updated, and deleted. In general, the nodes develop data products and you initiate a query on a node to extract the information for display or analysis. The data cache is organized as a relational database. Data is stored in named tables with named fields or columns. One record or row of data exists for each modeled object.

The data cache is a distributed database spread over the network of sensor nodes. It maintains a portion of the entire database on each node, based on its evaluation of the need for data at that node. It uses queries executed on the node as the predominant indicator of the need for the data on a particular node. Data records are filtered and routed among the nodes to satisfy these needs. It efficiently distributes data using dynamic multicast messages. It guarantees delivery as long as the data remains valid and it ensures consistency of the copies.

Data attribute routing has many advantages for a sensor network. First it removes any need for prior knowledge of the nodes or their addresses. Data messages are disseminated by matching attributes of the data and interests of the nodes. In a sensor network, node interests naturally cluster in a neighborhood allowing the system to take advantage of the natural properties of radio communication to reach many nodes with few messages. Secondly it establishes the primacy of the data over the source node. This is a more natural means of specifying which data is of interest to an application program.

query [data cache access statement]

All access to the database is performed with the **query** command. This command takes the form of the command **query** followed by a database access statement. The node interface language is modeled after the Structured Query Language (SQL). For normal commands—**insert, update, delete,** and **select**—the language is virtually identical to the SQL specification and can be easily used by anyone familiar with SQL databases. The node also implements some special commands and short hands. These are described more fully in the following section.

When the **query** command is issued, the node responds with the results of the command as shown in Figure 29. All responses begin with a capital **Q**, two integers enclosed in square brackets, and a colon. The first integer is the code assigned to your operation. All responses to the same operation use the same code. The second number is a count of the number of rows changed or returned by the database operation. The responses to the database operation follow after the colon. You will receive a response for each record of the database that is changed by an **insert**, **update**, or **delete** statement or for each record obtained by a **select** or **watch** statement.

The Fantastic Data Distributed Cache operates independently and asynchronously from the data access requests. Submitted requests are acted on and results are returned at a later time. Generally, this is done quite quickly, as the data is resident in memory.

```
q _load
Q[86,0]: d _load
Q[86,0]: t `node x32 !`cm i32`om i32`im i32`cb i32`ob i32`ib i32`
Q[86,1]: . `669f`0`0`0`0`0`0`
Q[86,2]: . `ad8e`0`0`0`0`0`0`
Q[86,3]: . `8ebd`0`0`0`0`0`0`
Q[86,4]: . `b1ea`11`0`0`677`0`0`
Q[86,5]: . `0`19`0`0`1224`0`0`
Q[86,5]: 0 Done.
```

Figure 29. The query results delivered by the **query** command.

2.3.6.1 Data Cache Interface Language

The Fantastic Data Distributed Cache presents an SQL based interface to application programs. The interface offers the important SQL statements—**insert**, **update**, **delete**, and **select**—although not in their full generality. This is especially true for the **select** statement: joins, unions, grouping, and sorting are not offered. It also offers some additional and highly useful statements, such as **watch**, **put**, and **undelete**, and supports a more general set of data types and analysis functions than are generally offered by SQL databases.

In the following description of the statements, words enclosed in square brackets as **[field list]** represent values that you should specify for your particular statement. Other words in bold font such as **select** are keywords and must appear exactly as shown. You may not use the keywords as a table name or as a field name. Key words are not case sensitive. In general, the punctuation (except for the square brackets) is required, although the data cache parser is quite forgiving.

Statements may be any of the following:

select [field list] from [table] where [condition] scope [extent]; Return the rows of the table that match the condition. The keyword **select** may be abbreviated by **s**. If a table name is given as a single word statement as **[table]**, it is interpreted as the equivalent of **select * from [table]**. The **where** and **scope** clauses are optional.

watch [field list] from [table] where [condition] scope [extent]; Return the rows of the table that match the condition as they are inserted, updated, or deleted. The keyword **watch** may be abbreviated by **w**. The **where** and **scope** clauses are optional.

select and watch [field list] from [table] where [condition] scope [extent]; Do both select and watch operations. The keywords **select and watch** may be abbreviated by **sw** . The **where** and **scope** clauses are optional.

cancel [operation]; Cancel a previously specified operation, for example, an ongoing watch operation. An operation code is returned by every **query** operation. The keyword **cancel** may be abbreviated by **c**.

insert into [table] [field list] values [value list]; Insert a row into the table. The row must not exist. The keywords **insert into** may be abbreviated by **i**.

put into [table] [field list] values [value list]; Insert a row into the table. If the row already exists, this operation is treated as the equivalent update. The keywords **put into** may be abbreviated by **p**.

update table set [field list] = [value list] where [condition]; Update all existing rows of the table that meet the condition. The keyword **update** may be abbreviated by **u**. The **where** clause is optional.

delete from [table] where [condition]; Delete all existing rows that meet the condition. If no condition is specified, this statement deletes all rows of the table. . The keyword **delete** may be abbreviated by **d**. The **where** clause is optional.

undelete from [table] where [condition]; Restore all deleted rows that meet the condition. The **where** clause is optional.

purge from [table] where [condition]; Permanently removes all rows of the table that meet the condition. The presence of deleted data is necessary to ensure consistency of the redundant caches so records should be purged only if some other mechanism makes sure that all nodes purge the same records at the same time. For example, the data expiration feature specified in the **create table** statement is safe. The **where** clause is optional.

Data Type	Size (bit)	Code	Example
signed integer	32	`i32, integer, long`	21456789
signed integer	16	`i16, short`	-32368, 0, 34, 32367
signed integer	8	`i8, byte`	-128, 127
unsigned integer	32	`u32, unsigned`	34, -78
unsigned integer	16	`u16`	0, 65535
unsigned integer	8	`u8`	0, 255
hex integer	32	`x32`	0000b1ea, ffffffff
hex integer	16	`x16`	b1ea, ffff
hex integer	8	`x8`	b1, ff
floating point	64	`f64, double`	4.356789123, 1e20
floating point	32	`f32, float`	4.3567
text	variable	`c, char`	"this is an example"
binary data	variable	`b, blob`	"1jkkjdsi99e2dsldkfslsl"
location	128	`geo, g`	"37.436211 -112.678453"

Figure 30. The data cache field types.

`show table [table];` Return the specification of a table. If no table is specified, all tables are returned.

`show type [type];` Return a description of the allowed data types. If no type is specified, all types are returned.

`show function [function];` Return the specification of the function. If no function is specified, all functions are returned.

`create table [table] ([field] [type], ... , [field] [type], primary [field list], private, expire [age]);` Create a new table. All tables must have a `primary` key. If the table is marked `private` it is never replicated. If an `expire` specification is given, rows are removed from the table when the specified number of seconds has elapsed since the row was changed.

`drop table [table];` Delete a table and all of its contents.

The word `[table]` in the above descriptions means the name of a table. In the `create table` statement, it must be a new name. In every other statement it must be the name of an existing table. All table names must start with a letter. They may contain digits. Names are not case sensitive. Cache creates some internals table to store information about its environment and about its own state. These table names

Name	Type	Description
`_b`	u64	the node that changed the record.
`_s`	u32	the series of the record change.
`_n`	u32	the sequence number of the record change.
`_t`	f64	the time at which the record change was originated.
`_lt`	f64	the time at which the record change was performed on the local node.

Figure 31. The special fields added to every data record.

Error	Code	Explanation
CacheEmpty	-99	Empty statement.
CacheUnrecognized	-98	Unrecognized statement \"%s\".
CacheInvalid	-97	Invalid \"%s\" statement.
CacheExpected	-96	Expected \"%s\" statement.
CacheExtra	-95	Extra stuff after statement.
CacheBadStatement	-94	Bad \"%s\" statement.
CacheBadClause	-93	Bad \"%s\" clause.
CacheExpectedToken	-92	Expected \"%s\" at position %d.
CacheBadFieldList	-91	Bad field list.
CacheBadValueList	-90	Bad value list.
CacheUnknownCode	-89	Invalid code \"%s\".
CacheUnknownTable	-88	Invalid table \"%s\".
CacheUnknownField	-87	Invalid field \"%s\" in \"%s\" clause.
CacheUnknownFunction	-86	Invalid function \"%s\" in condition.
CacheUnknownType	-85	Invalid type \"%s\".
CacheCanceled	-84	Canceled by request.
CacheNullField	-83	Null field not allowed.
CacheNullTable	-82	Null table not allowed.
CacheValueCount	-81	Number of fields (%d) != number of values (%d).
CacheTableExists	-80	Table "%s" already exists.
CacheInvalidValue	-79	Invalid value "%s" for "%s".
CacheBadTable	-78	Bad character "%c" in table name "%s".
CacheBadField	-77	Bad character "%c" in field name "%s".
CacheMissingClause	-76	Missing "%s" clause.
CacheNewer	-75	Newer data exists.
CachePrevious	-74	Previously processed.
CacheDuplicate	-73	Duplicate record with key "%s".
CacheNoRecord	-72	Can't create record "%s".
CacheNoData	-71	Can't create data segment.
CacheNoKey	-70	Can't create key.
CacheNoTable	-69	Can't create table "%s".
CacheNoWatch	-68	Can't create watch request."
CacheInternal	-67	Internal screwup.
CacheBadSn	-66	Bad sequence number "%s".
CacheBadStatus	-65	Bad record status "%s".
CacheReadOnly	-64	Can't write into table "%s".
CacheBadWhere	-63	Bad "where" clause: %s.
CacheError	-1	Error.
CacheDone	0	Done.
CacheData	.	Data returned from query.
CacheInsert	+	New record inserted.
CacheDelete	-	Record deleted.
CacheUpdate	>	Record updated.
CacheBefore	<	Record updated. This is the previous value.
CacheNotify	!	Watch query accepted.
CacheType	t	Data field names and types.

Figure 32. The data cache error codes.

begin with the character _. You may not create table names beginning with _, but you may query the internally created tables.

The word **[field]** in the above descriptions indicates the name of a field in a table. In the **create table** statement it must be a new field, except in the **primary** clause where it must be a name defined previously in the statement. In all other statements, the word field must indicate an existing field in the table. All field names must start with a letter. They may contain digits. Names are not case sensitive. Field names must be unique within the table. You may use the same field names in different tables.

The words **[field list]** in the above descriptions means a list of valid field names. Field lists are usually enclosed in parentheses with commas between the individual fields, but this is not always strictly required. In a field list the value ***** means all of the regular fields in the order specified in the **create table** statement. The absence of a field list is also interpreted as all of the regular fields. The special value **@** returns the primary key fields in the order specified in the **create table** statement. The special value **#** returns the special fields automatically set by cache and described below. If these special values are used, care must be taken to correctly interpret the returned values, as the table definition may be different than that expected by the user.

The word **[extent]** in the above descriptions may be **private**, **local**, or **global**. Queries with **scope private** are performed on the local node. They do not cause the flow of any data between nodes. Queries with **scope local** and **scope global** are network queries. They differ only in the scope of distribution of the query. Queries with **scope global** are distributed to the entire network; queries with **scope local** are distributed to all neighboring nodes. Either type of query may return data records created on any particular node. If no **scope** clause is present, **scope private** is assumed.

The words **[value list]** in the above descriptions means a list of valid constants that correspond to the fields specified in the field list. Value lists are usually enclosed in parentheses with commas between the individual values, but this is not strictly required. Equations are not allowed in value lists.

The word **[type]** in the above descriptions refers to any of the recognized data types. The data types are described in Figure 30. You may use the first or primary type code or any of the aliases. The primary name is reported by the **show table** command. The statement **show type** yields a complete and correct list of data types.

The word **[condition]** in the above descriptions, refers to an equation made up of field values, constants, operators, and functions that evaluates to either true (not equal to 0) or false (equal to 0). Different operators and functions are defined for each data type. The operators— **+**, **-**, *****, **/**, **&**, **|**, **!**, **^**, **<**, **<=**, **=**, **>=**, **>**, **!=** —are supported for most data types. Many functions are also supported for some data types. The statement **show function** yields a complete and accurate list of operators and functions. If no condition is specified with a **where** clause, all of the data in the table is affected by the operation.

All tables are automatically supplied with several special fields. The values of these fields may not be set with **insert**, **update**, or **put** statements, but they may be returned by queries and used in conditions. The special fields are shown in Figure 31.

Every operation returns a status code as shown in Figure 32. Negative values are error codes. Positive values indicate that data is returned. Zero indicates that the operation was completed successfully.

2.3.6.2 Control Data Tables

Certain control information is available only about the specific connected node. This information includes specific status information about the operation of the node. You may examine the information in these tables, but you may not change the information. All of these tables have names beginning with _. These tables are described below.

_node

The **_node** table holds information about nodes in the neighborhood of the node. It contains one record for the node itself, one record for each of its neighbors, and one record for the neighbors of the neighbors. This record is created or changed whenever the status of the node changes. The definition of the **_node** table is shown in Figure 33.

```
create table _node (node x32, forward i32, reverse i32, active i32, lrx
  u32, ltx u32, lack u32, long c, start u32, retry u32, sent u32, ack u32,
  other c, primary(long), private);
```

node is the short (16-bit) node id.
forward is the index used to communicate to the node. This field is only valid for neighbors.
reverse is the index used for communication from the node. This field is only valid for neighbors.
active indicates that the node is a neighbor when set to 1.
lrx is the last time a message was received from the node.
ltx is the last time a message was sent to the node.
lack is the last time an acknowledgement was received from the node.
long is the long (64-bit) id for the node. The long id is unique.
start is the time that the connection with the node was established.
retry is the number of times the node has been contacted to establish a link.
sent is the last time the connected node's listen schedule was sent to the node.
ack is the time the node acknowledged the connected node's listen schedule messsage.
other is a list of indexes used by other neighbors to talk to the node.

```
`1`674e`-1`-1`1148068242`1148068226`1148068225`06639b007240e198`1148067545`
  0`0`0`1 1 0 1 1 0 -1`

`1`1d70`0`1`1148068110`1148068097`1148068098`09639b0072400f98`1148067265`
  0`1148067936`1148067938`-2 0 2 4 -2 5 1`

`1`738e`6`-1`1148068093`1148068226`1148068225`06639b007240c198`1148067415`
  4`1148067924`1148067929`5 4 3 -2 -2

`1`c060`1`1`1148067980`1148068097`1148068098`00639b007240558c`1148067268`
  0`1148067681`1148067819`2 -2 1 2 -2 4 2`

`1`3e91`2`0`1148067970`1148068097`1148068099`01639b007240689a`1148067145`
  0`1148067626`1148067858`3 2 -2 0 3 3

`1`a2d7`3`1`1148068242`1148068097`1148068099`0f639b0072402d9a`1148067505`
  0`1148067642`1148067647`6 5 4 -2 0 2

`1`1c5c`4`1`1148068139`1148068097`1148068098`09639b007240a598`1148067625`
  1`1148067686`1148068139`-2 -2 5 3 -2

`0`3255`-1`-1`0`0`0`unknown`1148067116`5`1148068208`
  0`-2 -2 -2 -2 -2 -2 -2`
```

Figure 33. The definition of the **_node** table.

_qual

The **_qual** table holds information about the quality of the communication links to each neighbor. It contains one record for each node that has ever been a neighbor or that the node has tried to link to as a neighbor. The information in this table is used by the node to decide whether links should be maintained or dropped and when to attempt to contact new nodes. The definition of the **_qual** table is shown in Figure 34.

```
create table _qual (active i32, node x32, q i32, r i32, mg i32, ml u32, md
  i32, mr i32, a i32, al u32, ar i32, rr i32, rd i32, b i32, bl i32, br
  i32, tc i32, tr i32, primary(node), private);
```

active is 1 if the node is in use as a neighbor and 0 if it is not.
node is the short (16-bit) node id.
q is the node quality in the range 0 (bad) to 1000 (good).quality
r is the index used for communication from the node. This field is only valid for neighbors.arun
mg indicates that the node is a neighbor when set to 1.
ml is the last time a message was received from the node.
mr is the average RSSI on messages received from the node.
md is the average time synch error measured on messages received from the node.
a is the number of acknowledgements received from the node.
al is the last time an acknowledgement was received from the node.
ar is the average RSSI on acknowledgements received from the node.
rr is the average RSSI measured by the other node on messages it has received.
rd is the average time synch error as measured by the other node.
b is the number of acknowledgements expected but not received from the node.
bl is the last time an acknowledgement was expected but not received.
br is the average RSSI on acknowledgements with errors.
tc is the time at which the node status was last changed.
tr is the time that the next attempt to contact the node is scheduled.

```
`0`1d70`93`-12`65`1148075847`-76`-351`65`1148075892`-76`-79`466`86`
  1148075984`-97`1148075985`1148076945`

`0`738e`0`-37`1`1148076030`-80`63`0`1148072074`-80`-80`-757`20`
  1148076090`-100`1148076090`1148079930`

`1`a2d7`289`-5`488`1148076387`-75`29`593`1148076361`-76`-76`-29`259`
  1148076389`-93`1148067642`0`

`1`1c5c`702`2`364`1148076391`-62`-44`461`1148076395`-63`-63`41`128`
  1148076387`-74`1148072489`1148072488`

`1`5af5`652`4`634`1148076395`-71`317`513`1148076395`-65`-68`-300`281`
  1148076383`-77`1148067721`0`

`0`3255`0`0`0`0`0`0`0`0`0`0`0`0`0`0`0`0`

`1`826c`0`-53`0`1148075215`-75`-93`0`1148075223`-74`-77`87`2`
  1148076395`-100`1148076393`1148076392`
```

Figure 34. The definition of the **_qual** table.

_slot

The **_slot** table holds information about communication slot usage in the neighborhood of the node. It contains one record for every used slot. Records exist about slots in use by the node itself, any of its neighbors, and for neighbors of the neighbors. Some of the records are used by the node to communicate with its neighbors. Other records provide information to allow the node to attempt to contact new neighbors. Other records have merely been overheard and are maintained in case they might be needed in the future. The **_slot** record is created or changed whenever the status of a slot changes. The definition of the **_slot** table is shown in Figure 35.

```
create table _slot (used i32, node x32, mode c, other x32, start u32, stop
  u32, time i32, frequency i32, use u32, suspend u32, nsuspend i32,
  success f32, empty f32, backoff f32, cv i32, cc i32, ci i32, error i32,
  past c, primary(node, time), private);
```

used is 1 if the slot is in use and 0 if it is not.
node is the short id of the node that defined the slot.
mode is the type of slot. A value of < indicates a slot used for receive. A value of > indicates a slot used for transmit by the node.
other is the list of other nodes allowed to use the slot. A value of 1 indicates all nodes.
start is the beginning of the valid interval for the slot definition.
stop is the end of the valid interval for the slot definition.
time is the scheduled time slot.
frequency is the scheduled frequency channel.
use is the time at which the node started using the slot.
suspend is the time at which the node suspended using the slot.
nsuspend is the number of times slot usage has been suspended.
success is the ratio of successful slot usage to unsuccessful usage.
empty is the portion of the frames that are unused because there is no data to transmit or receive.
backoff is the current transmission backoff value.
cv is the current comb value.
cc is the current comb count.
ci is the comb index.
error is the number of consecutive errors if it is negative and the number of consecutive successful slot uses if positive.
past is a history of the last 30 frames: **+** indicates a message successfully transmitted or received, **-** indicates transmission or reception failure, **|** indicates a transmission backoff, and **.** indicates that there was no data.

```
`0`1c5c`<`1`1148324900`1148410279`66`0`0`0`0`1`1`1`1`1`-1`0``

`0`778c`<`1`1148324407`1148445545`68`0`0`0`0`0`0`0`0`0`0`0``

`1`5af5`<`1`1148324769`1148449996`85`0`383`0`0`0.211665`0.871543`0.1`4`7`4`
  -6`|||-|||||||-|||||||-||||||.-|.|||.|||-|`

`1`7f16`<`1`1148324685`1148369239`93`0`386`0`0`0.409218`0.831156`0.147622`8
   `14`1`-1`||-.||.+|.|||||||-|||-|||+|||||||+..|||`
```

Figure 35. The definition of the **_slot** table.

_load

The **_load** table holds information about the amount of communication between the node and its neighbors. It contains one record for every neighbor and a record that represents the total. The total is not necessarily the sum of the individual records since messages can be addressed to multiple neighbors. The definition of the **_load** table is shown in Figure 36.

```
create table _load (node x32, cm i32, om i32, im i32, cb i32, ob i32, ib
   i32, primary(node), private);
```

node is the short node id. The total record has **node** equal to zero.
cm is the number of messages queued for delivery to the node.
om is the number of messages sent to the node in the last second.
im is the number of messages received from the node in the last second.
cb is the number of bytes queued for delivery to the node.
ob is the number of bytes sent to the node in the last second.
ib is the number of bytes received from the node in the last second.

```
`a2d7`0`0`0`0`0`0`

`bf0a`2`0`1`125`0`60`

`7f16`0`0`0`0`0`0`

`ba47`0`0`0`0`0`0`

`de1`1`0`1`60`0`60`

`778c`6`0`1`405`0`60`

`1e58`13`0`1`991`0`60`

`6ae8`12`0`1`995`0`60`

`9ad4`0`0`0`0`0`0`

`0`57`0`1`3995`0`60`
```

Figure 36. The definition of the **_load** table.

_sas

The **_sas** table contains one record for each message that the sas protocol has queued for delivery to another node. It allows you to monitor the overhead communication required to effect the sas protocol. The definition of the **_sas** table is shown in Figure 37.

```
create table _sas (o x32, c x32, s x32, a x32, d x32, tq u32, ts u32, mn
   i32, priority i32, type i32, node x32, tic i32, why c, private);
```

o is the mask of neighbors for which the message was originally queued.
c is the mask of neighbors for which the message is currently queued.
s is is the mask of neighbors to which the message has been sent.
a is the mask of neighbors that have acknowledged the message.
d is is the mask of neighbors for which delivery has been cancelled.
tq is the time the message was queued for distribution.
ts is the time the message was first sent.
mn is the message number.
priority is the priority of the message. Lower numbers are higher priority.
type is the type of the messages. The sas protocol sends only two types of messages: type 0 is a node id mask, type 1 is a slot definition.
node is the id of the node that the message is about.
tic is the time slot that the message is about. It is valid only for slot definition messages.
why is the reason the message was queued.

```
`80`80`80`0`1701`1148474985`1148474988`295`1`0`ba47`100`SlotRepeat`

`88c`80`88c`80c`7300`1148475173`1148475173`316`1`0`a2d7`20`SlotRepeat`

`880`880`0`0`2100`1148474797`1148475359`353`2`1`a8b8`0`MaskNeighbor`

`80`80`0`0`100`1148474805`1148475359`355`2`1`e2b6`0`MaskNeighbor`

`8ee`8c0`7e`7e`7700`1148475142`1148475142`312`3`0`a2d7`20`ForwardSlot`

`7ff7`7fd5`22`22`0`1148475714`1148475714`420`3`0`7f16`2`ForwardSlot`

`880`800`0`0`0`1148475371`0`-1`48`0`afba`43`KeepAlive`
```

Figure 37. The definition of the **_sas** table.

_cache

The **_cache** table contains one record for each data message that the data cache has queued for delivery to another node. It allows you to monitor the communication required to share data records among the nodes. The definition of the **_cache** table is shown in Figure 38.

```
create table _cache (o x32, c x32, s x32, a x32, d x32, tq u32, ts u32, mn
   i32, priority i32, object c, data c, private);
```

o is the mask of neighbors for which the message was originally queued.
c is the mask of neighbors for which the message is currently queued.
s is is the mask of neighbors to which the message has been sent.
a is the mask of neighbors that have acknowledged the message.
d is is the mask of neighbors for which delivery has been cancelled.
tq is the time the message was queued for distribution.
ts is the time the message was first sent.
mn is the message number.
priority is the priority of the message. Lower numbers are higher priority.
object is the name of the table in which the record is stored.
data is the record key.

```
`7fff`4880`6e`6e`7f93`1148474056`1148474056`85`20`_flocal``afba`detection``

`7fff`4880`6e`6e`7f93`1148474056`1148474056`86`20`_flocal``afba`track``

`40`40`0`0`0`1148475586`1148475587`399`20`track``7d76`1148475259``
```

Figure 38. The definition of the **_cache** table.

_filter

The **_filter** table contains one record for each global query in the network. These queries are created with **scope global**. The global queries can be queued from any node in the network. This table allows you to determine which data records the node will distribute to satisfy global queries. The definition of the **_filter** table is shown in Figure 39.

```
create table _filter (node x32, object c, rule c, primary(node, object);
```

node is the short id of the node originating the global query.
object is the name of the queried table.
rule is the filter rule for the query. It is translated from the SQL syntax to a private shorthand.

```
`afba`location`.:1:.`

`afba`detection`.:1:.`

`afba`track`.:1:.`
```

Figure 39. The definition of the **_filter** table.

_flocal

The **_flocal** table contains one record for each global query in the network. These queries are created with **scope local**. The global queries can be queued from any node in the network. This table allows you to determine which data records the node will distribute to satisfy global queries. The definition of the **_flocal** table is shown in Figure 40.

```
create table _flocal (node x32, object c, rule c, primary(node, object);
```

node is the short id of the node originating the local query.
object is the name of the queried table.
rule is the filter rule for the query. It is translated from the SQL syntax to a private shorthand.

```
`de1`detection`.location.:40.298177 -74.085853:.2distance.:0.030:.2<=.`
`de1`track`.location.:40.298177 -74.085853:.2distance.:0.060:.2<=.`
`bf0a`detection`.location.:40.298150 -74.085737:.2distance.:0.030:.2<=.`
`bf0a`track`.location.:40.298150 -74.085737:.2distance.:0.060:.2<=.`
`afba`detection`.location.:40.298267 -74.085736:.2distance.:0.030:.2<=.`
`afba`track`.location.:40.298267 -74.085736:.2distance.:0.060:.2<=.`
`778c`detection`.location.:40.298166 -74.085958:.2distance.:0.030:.2<=.`
`778c`track`.location.:40.298166 -74.085958:.2distance.:0.060:.2<=.`
```

Figure 40. The definition of the **_flocal** table.

_route

The **_route** table contains information on how to reach a remote node. Each record specifies how a particular source node reaches a particular destination node. The sensor node does not normally maintain routing information for most of the nodes in the network. The node actively maintains routing table information only for those nodes that have issued global queries and for which it might need to generate data responses. Records are also maintained about routes that the node may have overheard from its neighbors. Compared to traditional routing processes, the sensor node's routing table is very sparse. The definition of the **_route** table is shown in Figure 41.

```
create table _route (source x32, destination x32, cost u8, nhop i32, hop c,
   at u32, primary(), private);
```

source is the short id of the source node. This field is usually the node itself or one of its neighbors.
destination is the short id of the destination.
cost is the cost to execute the route. Smaller cost values are better.
nhop is the number of alternative routes. The nodes maintain and use multiple, redundant routes to remote hosts.
hop is the list of alternative next hop routing choices.
at is the time at which this record was written.

```
`de1`afba`1`1`afba`1148398049`

`bf0a`afba`1`1`afba`1148398052`

`2390`afba`1`1`afba`1148398050`

`778c`afba`1`1`afba`1148399279`

`7fce`afba`1`1`afba`1148399129`

`8ab2`afba`1`1`afba`1148399620`

`7f16`afba`1`1`afba`1148398071`
```

Figure 41. The definition of the **_route** table.

2.3.6.3 Application Data Tables

Application data is available globally. The application data may be acquired from any node. If the data is not already resident on the node on which the query is performed, a network query can be performed to obtain the data and deliver it to you. Network queries are persistent, that is once a network query is performed data records automatically flow across the network to you until you cancel the query.

A network query is started by performing a `select` statement with the `scope local` or `scope global` qualifier. For example, the following command

```
query select and watch id, location, direction from location scope global;
```

performs a global network query that obtains all fields in the location record from all nodes in the network. Network queries create load on the network. They should be issued only for information that you really want or need. In normal operation, there are no active global queries—in order to develop tracks, the nodes exchange information only in small local neighborhoods.

The nodes develop and make available information about their status, about their location, about detections, and about tracks. This information is stored in the following tables:

info

The **info** table reports information about the configuration of a node. One record is available for each node. The node writes this record once when it begins execution. The definition of the **info** table is shown in Figure 42.

```
create table info (id x32, long c, sver c, fver c, start u32,
  primary(long));
```

id is the short node id.
long is the long node id. It is 64 bits long and is unique to each node.
sver is the version number of the software executing on the node.
fver is version number of the firmware executing on the node.
start is the time at which the node first began operation.

```
`674e`06639b007240e198`1.5.10`1.4.22`1148067545`

`1d70`09639b0072400f98`1.5.10`1.4.22`1148067265`

`738e`06639b007240c198`1.5.10`1.4.22`1148067415`

`c060`00639b007240558c`1.5.10`1.4.22`1148067268`
```

Figure 42. The definition of the **info** table.

status

The **status** table reports summary status information about a node. One record is available for each node. The node updates this record at the rate specified by the parameter **StatusInterval (300 s)**. The definition of the **status** table is shown in Figure 43. If a network query is performed on the **status** table, the resulting network load can easily surpass that required to accomplish the network's primary surveillance mission. High network load shortens network life and should generally be avoided.

```
create table status (id x32, battery i32, temperature i32, link i32, nbor
  c, start u32, rate i32, up i32, at u32, bin i32, bout i32, pin i32, pout
  i32, primary(id));
```

id is the node id. It is unique to each node and to each record.
battery is the current battery voltage in millivolts (mV).
temperature is the current node temperature in millidegree Celsius (mdC).
link is set to the number of available links or neighbors.
nbor contains a list of the neighboring nodes.
start is the time at which the node first began operation.
rate is the calibrated rate of the node clock. The nominal clock rate is 20,000,000 Hz.
up is the length of time that the node has been in operation in seconds (s).
at is the time of this report.
bin is the average incoming data rate over the last reporting interval in millibytes/second (mB/s).
bout is the average outgoing data rate over the last reporting interval in millibytes/second (mB/s).
pin is the average incoming data rate over the last reporting interval in millipackets/second (mp/s).
pout is the average outgoing data rate over the last reporting interval in millipackets/second (mp/s).

```
`8a61`2159`25656`1`5c97`1135312992`19999899`1510`1135314480`370`370`20`20`
```

Figure 43. The definition of the **status** table.

link

The link table reports summary information about a link. One record is available for each source and destination pair. For bidirectional links, therefore, there are two records, one written from each node's point of view. The definition of the link table is shown in Figure 44. If a network query is performed on the `link` table, the resulting network load can easily surpass that required to accomplish the network's primary surveillance mission. High network load shortens network life and should generally be avoided.

```
create table link (id x32, destination x32, at u32, active u8, quality u16,
  srx i32, stx i32, bin i32, bout i32, pin i32, pout i32,
  primary(id,destination));
```

id is the source node id. It is unique to each node and to each record.
destination is the destination node id. It is unique to each record.
at is the time of this report.
active is 1 if the link is being used and 0 if not.
quality is the measured quality of the link on a scale of 0 (bad) to 1000 (good).
srx is the average RSSI measurement made on the primary node.
stx is the average RSSI measurement made on the destination node.
bin is the average incoming data rate over the last reporting interval in millibytes/second (mB/s).
bout is the average outgoing data rate over the last reporting interval in millibytes/second (mB/s).
pin is the average incoming data rate over the last reporting interval in millipackets/second (mp/s).
pout is the average outgoing data rate over the last reporting interval in millipackets/second (mp/s).

```
`a2d7`3e91`1148081393`0`181`-79647`-77334`5223`3812`75`59`

`3e91`a2d7`1148081567`0`98`-78723`-80484`-315`0`0`0`

`a2d7`1d70`1148081393`1`57`-182000`-80000`3983`565`59`9`

`c060`1d70`1148081590`1`688`-64439`-65955`12850`13474`200`210`

`c060`3e91`1148081590`1`697`-72783`-70926`11454`11664`183`186`

`3e91`1c5c`1148081567`0`78`-78090`-80010`197`0`3`0`

`3e91`a8b8`1148081567`0`0`-80000`-81000`197`0`3`0`

`a2d7`5af5`1148081393`0`0`-61014`-59357`0`0`0`0`

`a2d7`c060`1148081393`0`19`-78000`-81000`0`0`0`0`

`a2d7`826c`1148081393`0`0`-81066`-78081`0`0`0`0`

`3e91`5af5`1148081567`1`404`-71021`-72741`7707`7884`118`115`

`3e91`826c`1148081567`0`0`-66238`-67810`197`0`3`0`
```

Figure 44. The definition of the `link` table.

location

The `location` table reports information about the location of the node. One record is available for each node. The node updates this record when it is moved. The definition of the `location` table is shown in Figure 45. If a network query is performed on the `location` table, the resulting network load can easily surpass that required to accomplish the network's primary surveillance mission. High network load shortens network life and should generally be avoided.

```
create table location (id x32, location geo, direction integer, at u32,
  primary(id));
```

id is the node id. It is unique to each node and to each record.
location is the location of the node in latitude and longitude.
direction is the direction in which the PIR sensor is pointing in degrees true (dT).
at is the time of this report.

`8a61`37.742020 -122.419558`34`1135313794`

Figure 45. The definition of the `location` table.

detection

The **detection** table reports information about a detection. One record is available for each detection. The nodes create a **detection** record whenever the PIR senses a target in the sensor field. The definition of the **detection** table is shown in Figure 46. Detection records are normally shared among neighbor nodes with a local query.

```
create table detection (id x32, channel i32, time u32, tus i32, location
  geo, course f32, speed f32, confidence f32, primary(id, channel, time),
  expire 300);
```

id is the node id. It is unique to each node and to each record.
channel is the sensor channel that made the detection. It is always 0 in the current implementation.
time is the time of this report in seconds.
tus is the time of this report in microseconds (us).
location is an estimate of the location of the target in latitude and longitude.
course is a crude estimate of the target course in degrees true (dT).
speed is a crude estimate of the target speed in meters per second (m/s).
confidence is a measure that the detection represents a real target. It ranges from 0 (low) to 1 (high).

```
`8a61`1`1135314755`912252`37.742020 -122.419540`0`0`0.998`
```

Figure 46. The definition of the **detection** record.

track

The **track** table reports information about tracks. One record is available for each track point. An entire track can be obtained by acquiring all records with the same value of **id** and connecting them together from the smallest value of **time** to the largest. The nodes update these records as the target moves through the sensor field. The definition of the **track** table is shown in Figure 45. Track records are normally shared among neighbor nodes with a local query.

```
create table track (id x32, time u32, location geo, course f32, speed f32,
  npoint i32, ndetection i32, duration i32, confidence f32, by x32,
  detection c, primary(id, time), expire 300);
```

id is the track id. It is a randomly assigned id created at the time of track initiation.
time is the time of this report.
location is the estimate of the current target location in latitude and longitude.
course is the estimate of the target course in degrees true (dT).
speed is the estimate of the target speed in meters per second (m/s).
npoint is the number of track points, up to and including this one.
ndetection is the number of detections used in this track point.
duration is the length of the track in seconds (s).
confidence is a measure of the confidence in the track point. It is between 0 (low) and 1 (high).
by is the node id of the node making this update.
detection is a list of detections used to make this track point.

```
`dad4`1148081079`37.510626 -122.493125`-48.068`1.082`1`3`0`0.83`1`1`ba47`
  de1-79 7fce-88 ba47-96`

`dad4`1148081096`37.510736 -122.493280`-48.068`1.082`2`3`17`0.83`1`1`ba47`
  de1-79 7fce-88 ba47-96`

`dad4`1148081113`37.510842 -122.493452`-50.602`1.109`3`1`34`1`1`1`
  778c`778c-113`

`dad4`1148081121`37.510891 -122.493493`-47.414`1.031`4`1`42`0.38`1`1`
  778c`a2d7-121`

`dad4`1148081131`37.510945 -122.493598`-49.628`1.054`2`1`52`1`1`1`5af5`
  5af5-131`

`dad4`1148081140`37.510971 -122.493626`-49.429`0.914`3`1`61`1`1`1`1c5c`
  1c5c-140`

`dad4`1148081148`37.511023 -122.493716`-49.996`0.976`3`2`69`0.48`1`1`5af5`
  1c5c-140 bf0a-148`

`dad4`1148081156`37.511079 -122.493747`-46.784`0.942`4`1`77`0.78`1`1`bf0a`
  9ad4-156`

`dad4`1148081165`37.511119 -122.493804`-44.026`0.893`5`1`86`1`1`1`ec4e`
  ec4e-165`

`dad4`1148081173`37.511163 -122.493876`-50.618`0.867`5`1`94`1`1`1`a2d7`
  a2d7-173`
```

Figure 47. The definition of the **track** table.

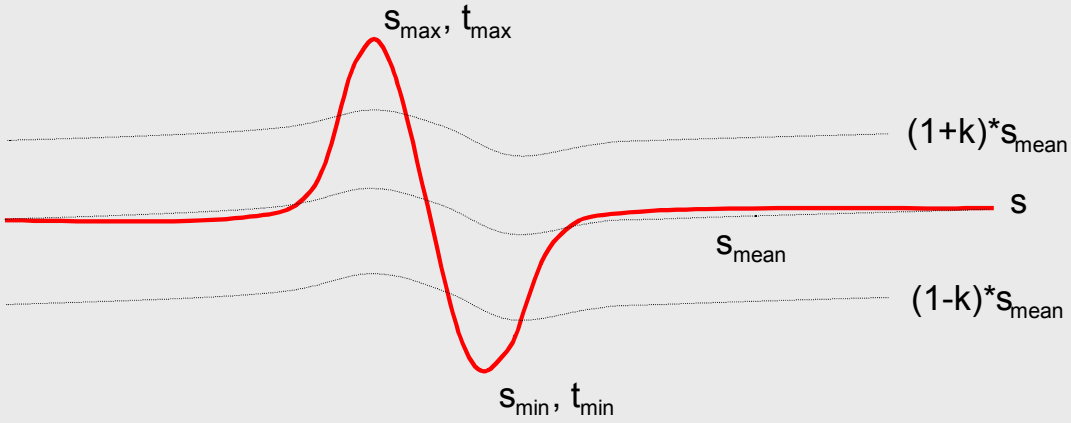

Figure 48. The PIR signal showing the characteristic shape of a detection.

3.4 Surveillance Application

The commands that control the software, firmware, and hardware used to control the surveillance application are described in this section.

3.4.1 Detector Test and Configuration

The sensor node uses a passive infrared sensor for detection of moving objects in the field. This sensor is configured with a sharply focused lens to provide a beam approximately ±10 degrees and a detection range of approximately 20 meters for personnel. Vehicles are detected at longer range.

Data from the passive infrared sensors is acquired at **`DetectionRate (100 Hz)`** and processed with a simple threshold detector looking for the pulse shape characteristic of a moving target as shown in Figure 48. The detection processing requires the signal level to rise to a value greater than **`DetectionThreshold (20 %)`** of the mean value and then fall the same amount within **`DetectionTimeWindow (2000 ms)`**. The pulse may also fall and then rise in a similar manner. After a detection is declared, the detector is inhibited from producing additional detections for **`DetectionInhibit (2000 ms)`**.

When a target is detected its location, course, and speed are roughly estimated from the pulse shape and a detection record is written to the data cache. The order of the pulse rise and fall determines the target direction perpendicular to the sensing beam, a measurement that is quite accurate. Distance from the sensor is estimated from the height of the pulse; higher pulses imply that the target is closer. Target speed is estimated from the time taken to complete the beam crossing. Neither distance from the sensor nor speed are reliable measurements; reliable estimates of target location, course, and speed are obtained from the tracker. The detections are communicated within a neighborhood around the detecting node and are used to drive the collaborative tracker on the detecting node and its neighbors.

The following commands allow you to control and test the node detector. This list may be obtained by typing **`detector`** as shown in Figure 49.

detector [on|off] -- acquire data continuously

The **`detector on`** command enables the detector. The node acquires and processes PIR data continuously at the specified rate. Detection processing is performed and if a target is detected, a **`detection`** record is written. The **`detector off`** command disables the detector.

```
detector
detector on -- acquire data continuously
detector off -- turn off detector
detector now -- acquire a single sample
detector power [on|off] -- control detector power
detector log [on|off] -- log debug messages
detector show [on|off] -- show all samples
```

Figure 49. The list of `detector` commands.

detector now -- acquire a single sample

The `detector now` command acquires and prints a single sample of PIR sensor data.

detector power [on|off] -- control detector power

The `detector power on` command forces the power for the PIR sensor to be turned on. Normally the power to the PIR sensor is turned on whenever the node tries to use the PIR sensor and is turned off when the sensor is not needed. The `detector power off` command cancels this forced operation.

detector log [on|off] -- log debug messages

The `detector log on` command enables the logging of information about the state of the detector receiver. The `detector log off` command disables the logging of this information.

detector show [on|off] -- show all samples

The detector log includes a strip chart of data samples. Normally only the sample values near a detection are printed as shown in Figure 50. When the `detector show on` command is issued, the detector log prints a continuous strip chart of the PIR sample data showing all samples. `The detector show off` command reverts to the normal state of printing only detections.

2.4.2 Tracker

The ultimate information product of the sensor network is target tracks. The tracks report the past movement of targets through the sensor field and predict the future movement of the target with an estimated current location and velocity vector. By exploiting the consistency of multiple detection in a neighborhood, the tracker confirms the presence of a target. Detections that do not correlate with other detections are discarded, reducing the false alarm rate associated with raw detections.

The tracker acquires track and detection data in a neighborhood around itself. Track data is acquired if the estimated location of the track is within `TrackFilterSize (40 m)` of the node. Detection data is acquired over half that distance. These tracks and detection form the basis for the extension of existing tracks and the creation of new tracks.

On a single node, the collaborative tracker employs standard tracking algorithms--a spatial and temporal data association window about the predicted track location and a linear fit to the prior track points and the new detections to produce an estimated target state vector. The tracker begins by creating a list of all tracks that are no older than `TrackTimeWindow (15 s)`. Then each detection is exaimned and compared to the active tracks. An estimate of the consistency of each detection with the existing tracks is made. Detections are associated with the track for which the fit is the best. Data association is accomplished spatially and temporally.

```
DETECTION: (51,4084)-(40,0)=(11,4084) at 1135290959.131696
  2033 2038 2038 2028 2021 2024 2029 2039 2044 2048       *          2034
  2052 2059 2060 2060 2059 2060 2058 2055 2052 2046        *         2056
  2044 2039 2038 2034 2031 2028 2033 2036 2037 2038       *          2035
  2043 2041 2036 2032 2030 2031 2032 2033 2034 2030       *          2034
  2031 2030 2028 2027 2025 2021 2023 2022 2026 2032       *          2026
  2035 2036 2037 2036 2037 2039 2043 2046 2044 2038       *          2039
  2037 2036 2038 2039 2043 2040 2040 2037 2034 2032       *          2037
  2030 2027 2027 2027 2030 2030 2033 2037 2042 2044       *          2032
  2045 2047 2047 2047 2046 2044 2043 2041 2042 2041       *          2044
  2042 2042 2044 2043 2047 2049 2052 2052 2052 2046       *          2046
  2040 2044 2040 2028 2024 2025 2032 2041 2048 2052       *          2037
  2058 2064 2066 2067 2065 2065 2067 2066 2065 2065        *         2064
  2071 2076 2083 2082 2080 2080 2081 2081 2085 2086        *         2080
  2086 2083 2083 2087 2094 2100 2104 2114 2125 2143        *         2101
  2164 2194 2250 2467 2896 3311 4057 4080 4059 4084            *     3156
  4059 4079 3311 2215 1333  735  367  161   56   10      *          1632
     0    7  128  368  680 1011 1330 1611 1847 2037   *               901
  2181 2283 2357 2407 2438 2447 2439 2429 2407 2381         *        2376
  2358 2335 2313 2291 2271 2251 2231 2212 2193 2176        *         2263
  2160 2147 2136 2125 2111 2095 2083 2068 2057 2043        *         2102
lat=37.456100 lon=-122.164000 orient=168.000000 cok=1
detected target at 37.455925 -122.163953 <- 37.456100 -122.164000 + 19.941
   at 78.000 31.616
16 put into detection
   (id,channel,time,tus,location,course,speed,confidence) values
   (b1ea,1,1135290959,131696,'37.455925 -122.163953',78.0,31.6,0.997);
```

Figure 50. An example of the detector log showing a detection.

The first measure is position consistency. The detection must be within **TrackPositionConsistency (10 m)** of the existing target track or of the predicted next location of the target. The temporal gap between the last track point and the detection must be less than **TrackExtensionTimeGap (20 s)**. If more than one existing track meets these specification the longest track or the track with the best fit is chosen. This is done with a composite score taking into account the above spatial and temporal separations and **TrackGoodLength (30 s)** and **TrackGoodNpoint (5)**.

After the association of all detections with their best track has been performed, the tracks are extended with a weighted linear fit. New detections are assigned a weight of 1. Old track points are weighted by decreasing amounts from the most recent point to the earliest track point. The weight of the most recent track point is **TrackPriorWeight (4)**. The weight decreases by 1 for each earlier track point. The resulting track point is distributed by creating a **track** record in the data cache. The velocity vector is given by the slope of the fitted line.

Unassociated detections may be used to create new tracks. Tracks are initiated when **TrackStart (3)** or more unused detections occur in the same data association window. If 3 or more detections are used to start a track, a linear fit is performed as for track extension. If only 2 detections are used for track initialization, the track point is placed at the midpoint between the 2 detections. In this case the velocity vector is estimated as zero.

Fitted tracks are subject to a consistency check. Track speed estimates must be less than **TrackSpeedMaximum (50 m/s)**. If the speed estimate is larger, the new track estimate is discarded.

The collaboration between multiple trackers is accomplished by sharing records through the data cache. Redundant updates to the same track are resolved automatically during network dissemination since **track** records are keyed by target id and time. Redundant track initiation (2 tracks simultaneously initiated by different nodes) is resolved by one tracker deleting its track in preference to the other.

Track estimates are propagated ahead of the predicted track, providing time to cue the nodes to turn on the sensor, start the detector and tracker software, and configure the communication system for low latency dissemination of critical data through the creation of a broadcast slot from the node.

2.4.3 Status Reporting

The nodes periodically report a summary of their status. This period is approximately **StatusInterval (300 s)**. The reporting interval is subject to some random variation so that a large number of records are not generated in unison on the network. The summary information is written into the **status** record.

At the same time the node reports on the status of all links to neighbor nodes. This information is written into the **link** table.

If a network query is performed on either the **status** or the **link** table, the resulting network load can easily surpass that required to accomplish the network's primary surveillance mission. High network load shortens network life and should generally be avoided.

2.5 Other Sensing

The commands that control the software, firmware, and hardware used to control the other sensing functions are described in this section.

2.5.1 Battery Meter Test

The sensor node includes a battery meter that can be used to monitor the condition of the battery. The battery voltage drops as battery energy is consumed.

The node is powered by 2 AA batteries. If rechargable NiMH batteries are used, the highest expected battery voltage is 2.6 V. The node is expected to operate until the battery voltage drops below 2.0 V. The duration of operation with voltages below 2.0 V is unpredictable. The node is not damaged if operated at lower voltages and the node behaves properly until it shuts down.

The following commands allow you to control and test the node battery meter. This list may be obtained by typing **battery** as shown in Figure 51.

battery now

When the **battery now** command is issued, the battery meter is read and the value displayed on the serial port. The raw sample value and the battery voltage are displayed as shown in Figure 52.

```
battery
battery now
battery on [interval, seconds between samples]
battery off
```

Figure 51. The **battery** commands.

```
battery now
2648 a58   2.134 v
```

Figure 52. The **battery now** display.

battery on [interval, seconds between samples]

The **battery on** command begins a sequence of readings of the battery meter at the specified interval spacing. Each reading is displayed as shown above.

battery off

The **battery off** command cancels the continuous reading and display of battery values.

2.5.2 Temperature Test

The sensor node includes a thermometer that can be used to obtain the temperature of the node. The thermometer is located inside the node between other electronic components. As such it responds to both the prevailing outside temperature and the heat generated by the electronics.

The following commands allow you to control and test the node thermometer. This list may be obtained by typing **temperature** as shown in Figure 53.

temperature now

When the **temperature now** command is issued, the thermometer is read and displayed on the serial port. The raw sample value, the temperature in degrees Celsius, and the temperature in degrees Fahrenheit are displayed as shown in Figure 54.

temperature on [interval, seconds between samples]

The **temperature on** command begins a sequence of readings of the thermometer at the specified interval spacing. Each reading is printed on the serial port as shown above.

temperature off

The **temperature off** command cancels the continuous reading and display of temperature values.

2.5.3 Sampler Test and Configuration

The sensor node includes an 8-channel analog to digital converter (ADC) that is used to measure certain aspects of the environment. The 8 channels of the ADC are connected as shown in Figure 55. The following commands allow you to control and test the node analog to digital converter. This list may be obtained by typing **sample** as shown in Figure 56.

```
temperature
temperature on [interval, seconds between samples]
temperature off
temperature now
```

Figure 53. The **temperature** commands.

```
temperature now
 546 222    2.572 C   36.630 F
```

Figure 54. The **temperature now** display,

sample [on|off] -- turn on/off the power to the adc

The **sample on** command forces the activation of power to the analog to digital converter. The sample off command cancels this forced activation. The power may not turn off immediately as it is automatically activated and deactivated by the node as required.

sample now [channel, 0-7] -- sample the specified adc channel

The **sample now** command acquires one sample from the specified channel. The sample value—in decimal, hexadecimal, and divided by 4096—is printed on the serial port as shown in Figure 57. For each of the node's regular sensors, specialized command exists to sample, convert, and display the sensor values in normal engineering units. For example, the **temperature now** and **battery now** commands.

sample log [on|off] -- log debug messages

The **sample log on** command enables logging of information about the data acquisition system. The **sample log off** command disables this logging.

2.6 Configuration and Control Parameters

The commands manipulate the configuration and control parameters are described in this section.

2.6.1 Parameter Configuration

The operation of the node is influenced by the value of a number of control parameters. These parameters influence all aspects of the node operation. The node is supplied with default values of all of the

Channel	Function
0	battery
1	magnetometer
2	magnetometer
3	magnetometer
4	pir
5	unused
6	unused
7	temperature

Figure 55. Correspondence between sensors and adc channels.

```
sample
sample [on|off] -- control power to the adc
sample now [channel, 0-7] -- sample the specified adc channel
sample log [on|off] -- log debug messages
```

Figure 56. The **sample** commands.

```
sample now 1
2323 493    0.567
```

Figure 57. The **sample now** display.

```
parameter
parameter read [minimum] [maximum]
parameter write [name] [value]
parameter different [minimum] [maximum] -- list parameters that differ from
   their original values
parameter original [minimum] [maximum] -- install original parameter values
parameter search -- list stored version numbers
parameter save -- saves current parameter values in flash memory
parameter load [version] -- load parameter values from flash memory
parameter erase [version] -- erase parameter values from flash memory
```

Figure 58. The **parameter** commands.

parameters that have been tested and are appropriate for normal operation. The values may be changed for special circumstances and may be saved in flash memory for reuse.

The parameter values may be viewed and manipulated with the parameter commands shown in Figure 58. In all of the commands, parameters may be referred to by their full name or by their number. Be aware that the names remain the same with new software versions, but that the numbers may change.

parameter read [minimum] [maximum]

The **parameter read** command lists the current values of the specified control variables. A minimum parameter and a maximum parameter may be specified to constrain the display. If no minimum or maximum is specified the entire list is displayed as shown in Figure 59.

The control parameters and their default values are shown below. A description of the use of these parameters is included in the relevant section of this guide.

parameter write [name] [value]

The **parameter write** command changes the value of the specified parameter. Please be careful when changing parameter values as the node does not check that the value makes sense.

parameter different [minimum] [maximum]

The **parameter different** command lists all parameters within the specified range that are different than their original, default values. If no range is specified, all parameters are shown. This command provides an easy and convenient way to determine if the node has been reconfigured.

parameter original [minimum] [maximum]

The **parameter original** command reinstalls the default value of the parameter into all parameters in the specified range. If no range is specified, all parameters are reset.

parameter search

The **parameter search** command lists the version numbers of the copies of the parameter values stored in flash memory. The node is capable of storing 2 parameter versions. When the node starts it automatically loads the highest version of stored parameters.

parameter save

The parameter save command stores the current parameter values in flash memory. If 2 or more versions of parameter values are already stored, the lowest version is overwritten.

```
parameter read
  0 NodeId=000085c2
  1 Loop=1
  2 WatchDog=1
  3 Gps=1
  4 Location=1
  5 TimeSync=1
  6 Modem=1
  7 Scheduler=1
  8 Sas=1
  9 Cache=1
 10 Detector=1
 11 Tracker=1
 12 Status=1
 13 GpsDelay=10 s
 14 ModemDelay=2 s
 15 SchedulerDelay=20 s
 16 DetectorDelay=300 s
 17 CommandWakeup=5000 ms
 18 ClockSleepUart=1
 19 ClockRate=20000000 Hz
 20 ClockSkew=0 Hz
 21 ClockRateNominal=20000000 Hz
 22 ClockRateMinimum=19800000 Hz
 23 ClockRateMaximum=20200000 Hz
 24 TimeSynchTechnique=1 [1=GPS, 2=OTA]
 25 TimeSynchAccuracy=500 us
 26 TimeSynchAgeMaximum=600 s
 27 TimeSynchMinimumInterval=5 s
 28 TimeSynchMaximumInterval=60 s
 29 TimeSynchLockout=2 s
 30 TimeSynchGpsReacquire=30 s
 31 TineSynchGpsMaximum=1800 s
 32 TimeSynchDriftRate=200 ns/s
 33 ModemTransmitPower=-5 dB
 34 ModemReceiveSensitivity=-80 dB
 35 ModemTuneTime=150 us
 36 BackoffMinimum=0.1
 37 BackoffDecrease=0.9
 38 BackoffIncrease=0.05
 39 CombMaximum=16
 40 CombCountMaximum=16
 41 ScrambleTime=1
 42 ScrambleFrequency=1
 43 ScrambleKey=1
 44 FrequencyBase=2
 45 FrequencyMany=50
 46 FrameGuardMinimum=1000 us
 47 FrameGuardMaximum=1000 us
 48 SlotLength=4000 us
 49 SlotGuardTime=1000 us
 50 SlotCarrierDetectTime=2000 us
 51 AckLength=2500 us
 52 AckSpacing=300 us
 53 AckGuardTime=10 us
 54 AckCarrierDetectTime=100 us
 55 AckDelay=300000 us
 56 TxVoltageMinimum=2100 mV
 57 NeighborExpected=5
 58 ListenInterval=10
 59 ListenValidTemporary=60 s
 60 ListenValid=86400 s
 61 ListenMany=1
 62 ListenManyMinimum=1
 63 ListenManyMaximum=5
```

Figure 59. List of control parameters.

```
 64 LoadIgnore=20 packet
 65 LoadRatio=10 packet/slot
 66 LoadInterval=60 s
 67 TransmitValid=60 s
 68 TransmitManyMaximum=1
 69 AnnounceProbabilityConstant=1
 70 AnnounceProbabilityFactor=0.8
 71 AnnounceProbabilityMinimum=0.01
 72 AnnounceProbabilityMaximum=1
 73 AnnounceTimeFactor=300
 74 ReallocateTime=60 s
 75 KeepAliveInterval=300 s
 76 NeighborDropInterval=900 s
 77 NonRetry=10
 78 NonInterval=10
 79 CriticalMaximum=300 s
 80 NodeQualityResponse=10 [1,99]
 81 NodeQualityMinimum=100 [0,1000]
 82 GpsLeapSecond=14 s
 83 LocationStartup=300 s
 84 LocationRecheckMinimum=300 s
 85 LocationRecheckMaximum=7200 s
 86 LocationNoFixMinimum=60 s
 87 LocationNoFixMaximum=600 s
 88 LocationAccuracyRequired=5 m
 89 LocationCountRequired=10
 90 LocationReportDistanceFactor=5 m
 91 LocationReportOrientationFactor=10 d
 92 LocationReportTimeFactor=3600 s
 93 LocationLatitude=-1000 dN
 94 LocationLongitude=-1000 dE
 95 LocationMagneticDeclination=0 [dT-dM]
 96 LocationOrientation=-1000 dT
 97 NorthPoleLatitude=79.74 dN
 98 NorthPoleLongitude=-71.78 dE
 99 CompassPulse=15
100 CompassWait=0 ms
101 CompassDelay=0 ms
102 CompassTurnOnDelay=100 ms
103 CompassInterval=200 ms
104 CompassUp=-1 [0,1,2]
105 CompassCenter[0]=0
106 CompassRange[0]=2048
107 CompassCenter[1]=0
108 CompassRange[1]=2048
109 CompassCenter[2]=0
110 CompassRange[2]=2048
111 DetectionThreshold=30 %
112 DetectionTimeWindow=2000 ms
113 DetectionInhibit=10000 ms
114 DetectionRate=100 Hz
115 DetectionReportThreshold=0.4 [0,1]
116 TrackFilterSize=60 m
117 TrackTimeWindow=60 s
118 TrackStartMinimum=3 detection
119 TrackPositionConsistency=10 m
120 TrackPriorWeight=4
121 TrackExtensionTimeGap=20 s
122 TrackGoodNpoint=5
123 TrackGoodLength=30 s
124 TrackSpeedMaximum=50 m/s
125 TrackSwitchDelay=1.5 s
126 StatusInterval=1800 s
```

Figure 59 (continued). List of control parameters.

parameter load [version]

The **parameter load** command restores the specified version of the parameter values from flash memory. If no version is specified, the highest stored version is used.

parameter erase

The **parameter erase** command erases the specified version stored in the flash memory. If no version, is specified the lowest stored version is erased. This command may be used to delete stored parameter values so that the node behaves in the default manner on subsequent incarnations.

2.7 Miscellaneous Functions

The commands that control additional miscellaneous node functions are described in this section.

2.7.1 Log Configuration

The sensor node can log a large volume of information about its operation. The information stream can be controlled to show certain classes of information and to suppress other classes with the log command.

name	content
task	information about the task scheduler
clock	information about the node clock
gps	information about the GPS receiver
sensor	information about the sensors
detector	information about the detector
modem	information about the modem
scheduler	information about the communication scheduler
sas	information about the sas protocol
cache	information about the data cache
cinout	data cache input and output messages
tsynch	information about the time synchronization function
schinout	input and output messages from the scheduler
sasmask	input and output messages from the sas protocol
location	information about the location determination function
link	information about the link test functions
tracker	information about the tracker
battery	information about the battery meter
compass	information about the compass
temperature	information about the thermometer
slot	information about slot quality
wake	summary of task scheduler wakeup
uncorrected	information about uncorrectable messages
cor	information about the receiver correlator

Figure 60. The list of log streams.

```
fec 100 2
100 iterations, 2 errors: encode=9975 us, decode= 13436 us
```

Figure 61. The `fec` display.

log [on|off] [stream name or mask]

The log command enables or disables the logging of operational information about the specified stream. Many of the log streams can also be controlled from commands located under their respective function headings. For example, the command `gps log on` is the same as the command `log on gps`. The available log streams are shown in Figure 60.

2.7.2 Forward Error Correction Test

Communication messages are protected by a Reed-Solomon(255,223) forward error correction (FEC) code. This code incorporates 32 8-bit code symbols and 223 8-bit data symbols in a 255 symbol package. The sensor node employs implied zero fill to compress the 255 byte packet if the true data length is shorter than 223 bytes.

fec [number of iterations] [number of errors]

The performance of the encoder and decoder can be tested with the `fec` command as shown in Figure 61.

2.7.3 Flash Memory Test

The node uses flash memory to store both program and data. Normally you should not alter the contents of the flash memory, as the possibility of overwriting program information is high. Other commands described in this guide should be used to store data in the flash memory (see `parameter save`).

The flash memory is organized into blocks as described in **Intel Advanced+ Boot Block Flash Memory (C3) Data Sheet** [4]. Individual bits of the flash memory may be cleared with the flash programming commands, but bits may only be set by erasing an entire block. This makes writing data to flash memory a cumbersome operation. In general, the block must be identified, unlocked, and erased. Then the individual words of the block may be programmed with the appropriate values. Afterwards, the block is locked to prevent inadvertent changes.

Program information is loaded into the node from a PC using special software and a special programming cable (known as the ByteBlaster II). Installation and use of the ByteBlaster is described in more detail in **ByteBlaster II Download Cable User Guide** [1].

The following commands allow you to control and test the node flash memory. This list may be obtained by typing `flash` as shown in Figure 62.

```
flash
flash read [minimum address, hex] [maximum address, hex]
flash write [address, hex] [value, hex]
flash program [address, hex] [value, hex]
flash erase [address, hex]
flash lock [address, hex]
flash unlock [address, hex]
flash [on|off]
```

Figure 62. The `flash` commands.

```
flash read 3000 3003
FW[3000]: d9da  55770 __
FW[3001]: 9416  37910 __
FW[3002]: 1f12   7954 __
FW[3003]: 2037   8247 7_
```

Figure 63. The **flash read** display.

flash read [minimum address, hex] [maximum address, hex]

The flash read command reads and displays 16-bit words of the flash memory. The memory contents are displayed as integers in hexadecimal and decimal and as text as shown in Figure 63.

flash write [address, hex] [value, hex]

The **flash write** command writes a 16-bit value to the flash memory controller. Note that the writes to the flash memory do not (in general) cause the flash memory to change. Rather most writes are commands to the flash memory controller. The flash memory controller must first be put into programming mode to change the actual contents of the memory. The **flash program** command does this for you.

flash program [address, hex] [value, hex]

The **flash program** command writes the specified value to the specified location in the flash memory. Note that the **flash program** command can only clear bits. That is, it will clear any bits that are zero in your specified value, but will leave unchanged all bits corresponding to ones in your specified value. If you want the memory to accurately reflect your specified values, you must first erase the entire flash memory block (see **flash erase**). You must also unlock the block before any programming can take place (see **flash unlock**).

flash erase [address, hex]

The **flash erase** command erases the entire flash memory block that includes the specified address. Flash memory is erased by setting all bits. After the memory is erased all 16-bit words hold the hexadecimal value ffff. Flash memory blocks are not all the same size.

flash lock [address, hex]

The **flash lock** command locks a block of flash memory. Locked blocks cannot be changed. All flash memory blocks are locked when the node is started.

flash unlock [address, hex]

The **flash unlock** command unlocks a block of flash memory. Blocks must be unlocked before they can be changed. All flash memory blocks are locked when the node is started. If you unlock and change a block, you should lock it again afterwards.

flash [on|off]

The **flash on** command supplies power to the flash memory. The **flash off** command disconnects the power for the flash memory. You should leave the power for the flash memory on because it consumes virtually no power when it is not actively used. The sensor node reads program and configuration data from the flash memory when it starts, when you execute a **parameter restore** or **parameter save** command, or when the loop detector is triggered.

```
register
register read [minimum register, hex] [maximum register, hex]
register write [register, hex] [value, hex]
```

Figure 64. The **register** commands.

2.7.4 Processor Register Test and Configuration

The processor registers are used and manipulated by the node software during normal operation. We do not recommend modifying any of the register values with the **register** commands. The processor registers are described in **Excalibur Devices Hardware Reference Manual** [3].

The following commands allow you to control the processor registers. This list may be obtained by typing **register** as shown in Figure 64.

register read [minimum register, hex] [maximum register, hex]

The **register read** command reads and prints the values of the specified registers.

register write [register, hex] [value, hex]

The **register write** command writes the specified value to the specified register. Writing a register value can cause the node to perform various operations. We do not recommend writing any register values.

2.7.5 Programmable Logic Device Test and Configuration

The processor consists of a general purpose computing core and a programmable logic device (pld). The pld registers support the communication between the general purpose computing core and the programmable logic. These registers are used by the node software and firmware in normal operation of the node. We do not recommend modifying any of these values with the **pld** commands.

The following commands allow you to control and test the pld registers. This list may be obtained by typing **pld** as shown in Figure 65.

pld read [minimum register, dec] [maximum register, dec] -- show readable registers

The **pld read** command reads and prints the values of the specified registers. Please note that these registers are specified with decimal values, rather than hex values as for most other node memory operations.

pld write [register, dec] [value, dec] -- write a register

The **pld write** command writes the specified value to the specified register. Writing a register value can cause the node to perform various operations. We do not recommend writing any register values. Please note that these registers are specified with decimal values, rather than hex values as for most other node memory operations.

```
pld
pld read [minimum register, dec] [maximum register, dec] -- show readable
   registers
pld write [register, dec] [value, dec] -- write a register
pld list [minimum register, dec] [maximum register, dec] -- show all
   registers
```

Figure 65.The **pld** commands.

```
pld list
         ClockOn   0: 00000000           0 w
     ClockSecond   4: 00000280         640 w
        ClockTic   8: 00ce211a    13508890 r
       ClockRate  12: 00000000           0 r
      ClockError  20: 00000000           0 r
        GpsPower  32: 00000000           0 w
         GpsRxOn  36: 00000000           0 w
         GpsTxOn  40: 00000000           0 w
      GpsRxIndex  44: 002c0002     2883586 r
      GpsTxIndex  48: 00000000           0 r
     SensorPower  64: 00000000           0 w
        SensorOn  68: 00000000           0 w
   SensorChannel  72: 00000000           0 w
    SensorSample  76: 00000000           0 r
SensorChannelOut  80: 00000000           0 r
      SensorDone  84: 00000000           0 r
   MagneticInput  96: 00000000           0 w
 MagneticOutput0 100: 0c500750   206571344 r
 MagneticOutput1 104: 07520b87   122817415 r
        PirPower 128: 00000000           0 w
           PirOn 132: 00000000           0 w
         PirRate 136: 00000000           0 w
        PirIndex 140: 00000000           0 r
         ModemOn 160: 00000000           0 w
         ModemPa 164: 00000000           0 w
        ModemLna 168: 00000000           0 w
    ModemRxIndex 172: 00000a0c        2572 r
    ModemTxIndex 176: 00000030          48 r
     ModemStatus 180: 00008101       33025 r
    ModemCurrent 184: 00008005       32773 r
             Led 192: 00000000           0 w
        PllPower 196: 00000000           0 w
     ClockSelect 200: 00000000           0 w
```

Figure 66. The list of pld control registers.

pld list [minimum register, dec] [maximum register, dec] -- show all registers

The **`pld list`** command displays all of the pld registers as shown in Figure 66. The registers are either readable or writable as indicated by the **`r`** or **`w`** in the last column. Registers that are not marked for either reading or writing are unused. Registers may be referred to by their name or by their address in all of the **`pld`** commands.

2.7.6 Switched Power Supply Test and Configuration

Certain node components run off a switched power supply. Normally the node software and firmware controls the state of this power supply and user intervention is not required. The power supply is turned on when any component requiring power from the supply is in use. The power supply is switched off, when no component requires power from it.

The following command allows you to directly control the state of the switched power supply.

power [on|off] -- control 3.3V switched supply

The **`power on`** command forces the switched power supply into the active state. The power supply remains active even if no node component requires it. The **`power off`** command cancels the forced activation of the supply. The supply may not turn off, if a node component is currently using it.

```
id
Node id: 01639b0072406b99 3255
Software: 1.4.14 05.12.06
Firmware: 1.4.8 05.12.02
```

Figure 67. The `id` display.

2.7.7 Serial Port Configuration

In order to conserve energy, the node does not read data from the serial port very often. The interval between successive read attempts becomes longer as the serial port remains unused. If you plan to do extensive testing, you may want to configure the node to be more responsive to your input.

The parameter `Commandwakeup (5000 ms)` allows you to set the maximum interval between serial port reads for this and future incarnations of the node. This value is saved in the flash memory by the parameter save command.

wakeup [on|off]

The `wakeup on` command allows you to force the node to read from the serial port frequently. The `wakeup off` command allows the node to revert to its normal state.

2.7.8 Identification

The long node id is permanently set in the node memory. It is 64 bits long and is unique to the node. The short node id is 16 bits long and is dynamically created by the node from the long id and several mathematical and logical operations. The short node id is used for identification of messages in the local neighborhood around the node. The long and short id are exchanged in the initial handshake between neighbors.

id

The id command displays some identifying information about the node. It displays the node id, in both long and short forms, and the software and firmware version numbers as shown in Figure 67.

3.7.9 Loop Detector

The loop detection timer protects the node from software faults. This timer expires after about 5 seconds. When the loop detection timer expires, a stack trace is performed. The stack trace is printed and saved in the flash memory. Then the node is allowed to continue executing. In most cases it will resume executing the problem loop until the watch dog timer expires. The `loop` commands are shown in Figure 68.

loop test

The `loop test` command exercises the loop detection timer. The node enters an infinite loop that terminates with the expiration of the loop detection timer.

```
loop
loop recover
loop test
```

Figure 68. The `loop` commands.

```
loop recover
SchedulerFunction=115 DataCount=14
REVERSE STACK TRACE AT 1135311407.278415
   62784() from      114
   62b68() from    627a0
   16668() from    62db0
   15b14() from    1667c
   4e918() from    15fd0
  550320() from    68d10
```

Figure 69. The display of the recovered stack trace.

loop recover

The saved stack trace may be recovered for analysis with the **loop recover** command. This command recovers the saved stack trace from the flash memory and displays it as shown in Figure 69. The numeric stack trace may be converted to a more useful form with the aid of the program symbol table.

2.7.10 Watch Dog Timer Test and Configuration

The watch dog timer protects the node from software faults. It is reset during task scheduling. If the timer counts down to zero, then the normal processing is interrupted and the node is reset.

The watch dog timer expires after about 15 seconds. When it expires the node is reset. All unsaved state information is lost and the node restarts as if from a cold start. If the loop timer has previously expired and saved a stack trace in flash memory, that stack trace remains and may be recovered for analysis.

watch

The **watch** command exercises the watch dog timer. The node enters an infinite loop which is terminated with the expiration of the watch dog timer. The node is reset if the test is successful. There is no recovery from a test of the watch dog timer. An example of the watch dog timer test is shown below. Note that the loop timer is triggered first in this test, then the watch dog timer is triggered and the node is reset.

reset

The **reset** command causes the node to shutdown and restart immediately by triggering the watch dog timer. There is no recovery from a **reset** command. When the node restarts it will read and use any parameter values that have been saved in the flash memory. Because the **reset** command has serious consequences, it must be issued twice within 10 seconds to be effective.

3.7.11 Node Operation

When the node starts, it prints an identification message and then pauses for 5 seconds to allow you to type commands. A restricted set of commands is available at this time. You may view and alter any configuration parameters, instruct the node to start execution, or reset the node. Any time you type any thing, the node will wait for another 60 seconds before starting automatically.

If you want it to start immediately, use the **go** command. When the node starts, it carefully starts the hardware, firmware, and software modules that are specified in the configuration parameters for automatic operation. Normally, all of the node modules are started automatically. These modules are controlled by the parameters listed in Figure 70.

When you issue the **go** command or if the automatic start procedure is triggered, the node finishes its startup sequence as shown in Figure 71.

go

The **go** command starts the standard sensor node processing. The **go** command is available only during the startup sequence. It shortens the normal timed wait for user input before beginning standard node processing.

WatchDog (1)	
Gps (1)	
Location (1)	will also start Gps
TimeSync (1)	will also start Gps
Modem (1)	
Scheduler (1)	requires Modem
Sas (1)	requires Scheduler
Cache (1)	requires Sas
Detector (1)	requires Cache for detection reporting
Tracker (1)	requires Cache
Status (1)	requires Cache

Figure 70. The node startup configuration parameters.

```
SENSOR NODE
Copyright (c) 1996-2005 Fantastic Data. All rights reserved.
Node id: 09639b007240c498 8a61
Software: 1.4.18 05.12.09
Firmware: 1.4.8 05.12.02
Node will start in 5 seconds.
Starting loop detector.
Starting watch dog timer.
Delayed gps start in 10 seconds.
Delayed modem start in 2 seconds.
Delayed detector start in 300 seconds.
Delayed scheduler start in 20 seconds.
Starting location analysis.
Starting time synchronization analysis.
Delayed gps start in 2 seconds.
Gps already started.
Starting scheduler.
Delayed sas start in 2 seconds.
Starting sas.
Delayed cache start in 2 seconds.
Starting cache.
Making data tables.
Starting status reporting.
Starting detector.
Starting tracker.
```

Figure 71. The normal node start up sequence.

3. Compass Calibration

The Fantastic Data sensor node includes a 3-axis magnetometer that is employed as a digital compass to determine the pointing direction of the detector. The compass must be calibrated before use. Compass calibration is performed in the factory prior to delivery to the customer. The compass can be recalibrated in the field using the built-in calibration routine.

Compass calibration consists of exposing the node to the extremes of the earth's magnetic field in all directions while the node is held in its intended vertical orientation. In other words, the node is slowly rotated about its intended vertical axis as the node acquires magnetometer data. By inspecting the acquired data, the built-in calibration routine selects the vertical axis and computes scaling and offset parameters for the other 2 axes. The compass is then ready for use.

To perform the compass calibration, complete the following steps:

1. Connect the node to a computer with the supplied serial cable. The cable plugs into the small port on the under side of the node.
2. Install batteries and allow the node to begin operation. It is not necessary to install the back cover during the calibration procedure.
3. Mount the node on the calibration turntable in the intended vertical orientation.
4. Begin the calibration data collection by typing the following commands:

```
compass calibration on
compass continuous on
```

5. Slowly rotate the node on the turntable. It should take about 2 minutes to complete a rotation.
6. Then end data collection by typing the following commands:

```
compass calibration off
compass continuous off
```

7. The node compass is calibrated. To check your calibration, you may inspect the value of the node compass with the command

```
compass now
```

8. You may view the node compass calibration values by inspecting the associated parameters:

```
parameter read CompassUp CompassRange[2]
118 CompassUp=-1 [0,1,2]
119 CompassCenter[0]=0
120 CompassRange[0]=2048
121 CompassCenter[1]=0
122 CompassRange[1]=2048
123 CompassCenter[2]=0
124 CompassRange[2]=2048
```

9. Save your calibration settings in the flash memory with the command

```
parameter save
```

If you do not save the calibration settings, they will be lost when the node restarts.

4. Information Extraction and Display Program

The information products of the sensor network—node locations, node status, detections, and tracks—can be viewed as text data records on a simple terminal emulator. However, that process is less than satisfactory since it does lead to a good understanding of the situation. The information products may also be automatically extracted, analyzed, and displayed by a companion suite of programs available for Linux systems.

This suite extracts the data to a computer running the Linux operating system. The data records developed on the sensor network are automatically extracted by issuing the appropriate data access commands and the results are stored in the Linux version of the Fantastic Data Cache. The display program is interfaced to the data cache to display the data in a meaningful manner.

This suite is available as source code for Linux systems. Instructions for installing and operating the suite are given in this section.

4.1 Installation instructions

1. Make a new directory where you want to put the source code. You can call it whatever you like.

   ```
   mkdir [source]
   ```

   Replace **[source]** with the name you chose for the source code directory.

2. Copy the source code from the CD to the [source] directory.

   ```
   cp -r /mnt/cdrom/* [source]
   chmod 0777 * */* */*/* */*/*/*
   ```

3. Decide where you want to put the data files and the executable programs. It's easiest if you make the directories bin and data as subdirectories of your home directory. But you can put them anywhere you want. If you put them elsewhere, pay careful attention to Step 6.

   Make both directories and the following subdirectories.

   ```
   mkdir [bin]
   mkdir [data]
   mkdir [data]/elevation
   mkdir [data]/images
   mkdir [data]/log
   mkdir [data]/overlays
   mkdir [data]/profiles
   mkdir [data]/symbols
   ```

   Replace **[bin]** and **[data]** with the names you chose.

4. Define symbols that point to the directories. You may want to edit the file x86.make to do this.

   ```
   export BIN=[bin]
   export DATA=[data]
   ```

5. Make the executable programs and data files.

   ```
   cd [source]
   make clean
   make
   ```

6. You must add the **[bin]** directory to your run time library search path with the commands:

   ```
   export LD_LIBRARY_PATH=[bin]
   ```

   You probably will also want to put the [bin] directory in your search path, so that the Linux shell will easily find the programs.

   ```
   export PATH=$PATH:[bin]
   ```

   If the [bin] and [data] directories are not subdirectories of your home directory, you must define a symbol that says where they are. This symbol is defined as the parent directory of the **[bin]** and **[data]** directories.

   ```
   export DDS=[parent directory]
   ```

   This must be done every time you execute the programs, so it is easiest to put the commands in a file such as .bashrc.

7. You are now ready to run the programs to extract data from a node and display it on the map.

4.2 Operating instructions

1. Start your network of sensor nodes. Connect one node to the computer with the special serial cable.
2. Start the data cache on the local system. It runs in the background.

   ```
   cache -none &
   ```

3. Start the data extractor

   ```
   extractor -file [data]/extractor.query
   ```

   All communication with the node will appear in the terminal window and will be logged to files in the **[data]/log** directory. Data corresponding to the queries in the file **extractor.query** will be automatically extracted from the node and inserted into the local data cache.

4. Start the local data cache query program in another terminal window

   ```
   process
   ```

   You may type SQL commands and get responses from the local data cache. Some useful commands are:

   ```
   delete from detection;     // deletes all detection to clear display
   delete from track;         // deletes all tracks to clear display
   select * from location;    // gets a list of node location records
   ```

   The Linux version of data cache is much pickier about SQL syntax than the sensor node. In particular, you must supply all of the punctuation including the closing semi-colon.

5. Start the map display program

   ```
   sitmap &
   ```

It will present a window and will automatically select a map for you. Since your installation script comes with 2 maps--one of the area around the Myer Center and the other of the area around the Fantastic Data World Headquarters--chances are that it will pick the wrong one. You can change the map with a menu available from the displayed map name in the lower right corner.

You will probably want to change a lot of other things about the appearance of the display. These changes will be remembered for subsequent invocations.

First, you probably want to actually see your data. To do that you have to turn on the overlays. Go to the top-left hand corner, the button with the three concentric squares leads to some menus. Pick

```
Overlay->Real->Node->Activate
Overlay->Real->Detection->Activate
Overlay->Real->Track->Activate
```

Your data should appear on the screen.

You may have to recenter the map and zoom in to see the data. You can click in the pan box to recenter the map. The test site is approximately in the center of the MC map. Or you can use the arrows in the bar at the bottom of the window or the commands in the menu to pan around.

If you want to turn off all of the control features, type Control ! in the window. Do it again, to turn them back on.

5. References

1. **ByteBlaster II Download Cable User Guide**. Version 1.1. December 2004. UG-BBII81204-1.1 P25-10324-00. Altera Corporation. San Jose, California. www.altera.com. ug_bbii.pdf.

2. **CC2400 Preliminary Data Sheet**. Revision 1.3. October 2004. Chipcon AS. Gaustadalleen, Norway. www.chipcon.com. CC2400_Data_Sheet_1_3.pdf.

3. **Excalibur Devices Hardware Reference Manual**. Version 3.1. November 2002. MNL-EPXA10HRM-3.1. Altera Corporation. San Jose, California. www.altera.com. mnl_arm_hardware_ref.pdf.

4. **Intel Advanced+ Boot Block Flash Memory (C3) Data Sheet**. October 2003. 290645-017. Intel Corporation. Santa Clara, California. www.intel.com. 29064517.pdf.

5. **NMEA 0183**. Version 3.01. National Marine Electronics Association. www.nmea.org.

6. **TIM-Lx GPS Modules System Integration Manual/Reference Design**. Revision C1. August 2005. GPS.G3-01001-C1. u-blox AG. Thalwil, Switzerland. www.u-blox.com. TIM-Lx_Sys_Int_Manual(GPS.G3-MS3-01001).pdf.