OpenAMIP™ Standard

Version 1.12

November 11, 2016

iDIRECT

VT iDirect® is a global leader in IP-based satellite communications providing technology and solutions that enable our partners worldwide to optimize their networks, differentiate their services and profitably expand their businesses. Our product portfolio, branded under the name iDirect, sets standards in performance and efficiency to deliver voice, video and data connectivity anywhere in the world. VT iDirect is the world's largest TDMA enterprise VSAT manufacturer and is the leader in key industries including mobility, military/government and cellular backhaul.

VT iDirect®
Company Web site: http://www.idirect.net ~ Main Phone: 703.648.8000
TAC Contact Information: Phone: 703.648.8151 ~ Email: tac@idirect.net ~ Web site: http://tac.idirect.net

iDirect Government™, created in 2007, is a wholly owned subsidiary of iDirect and was formed to better serve the U.S. government and defense communities.

iDirect Government™
Company Web site: http://www.idirectgov.com ~ Main Phone: 703.648.8118
TAC Contact Information: Phone: 703.648.8111 ~ Email: tac@idirectgov.com ~ Web site: http://tac.idirectgov.com

Document Name: OpenAMIP_Standard_RevC_11112016.pdf

Document Part Number: T0000682

Revision History

The following table shows all revisions for this document. To determine if this is the latest revision, check the Technical Assistance Center (TAC) Web site. Refer to *Getting Help* on page ix for TAC access information.

Revision	Date	Updates
A	08/04/2015	• First release of the OpenAMIP Standard (version 1.8) document in iDirect Technical Publications template. For changes from version 1.7 to 1.8, see *Version Changes* on page 17. • The time parameter of the w command is now made mandatory. See *Message Types* on page 5.
B	10/12/2016	Updated to OpenAMIP Standard version 1.9 to 1.12.
C	11/11/2016	Updated “w, W, H, and C” type message descriptions.

Contents

About

Purpose

This document describes the Open Antenna Modem Interface Protocol (OpenAMIP™) for satellite terminals. OpenAMIP is an ASCII message-based protocol for the interchange of information between an antenna controller and a satellite modem. OpenAMIP allows the modem to command the controller to seek a particular satellite. OpenAMIP also allows the modem and controller to exchange information necessary to initiate and maintain communications through the satellite.

OpenAMIP is designed to be extensible for vendor-specific enhancements.

Disclaimer

The Protocol was developed by iDirect.

The name "OpenAMIP" is a trademark™ of iDirect.

Permission to copy and distribute this document in unmodified form is hereby granted to all without restriction. Modified forms of this document may be distributed, but only if this "legal matters" section is retained intact and provided that any document that describes a modified form of the protocol clearly states that the protocol is modified.

To the extent that iDirect has rights to control the protocol itself, iDirect grants rights to implement the protocol to all, without restriction.

Use of the trademark "OpenAMIP" to describe an unmodified implementation of this protocol is unrestricted. Use the term "modified OpenAMIP" to describe a variant of this protocol, is also unrestricted; however the document containing the term "modified OpenAMIP" refers to this document.

While iDirect, Inc. strives to make the information in this document as accurate as possible, iDirect makes no claims, promises, or guarantees about the accuracy, completeness, or adequacy of the contents, and expressly disclaims liability for errors and omissions. No warranty of any kind, whether implied, expressed, or statutory, including but not limited to the warranties of non-infringement of third party rights, title, merchantability, or fitness for a particular purpose, is given with respect to the contents of this document.

iDirect, Inc. reserves the right to change or update this document at any time.

Certification

You may certify your compliance with the test suite yourself. If you do, you are free to use the trademark "OpenAMIP™" freely for any product that you have certified.

Your use of the OpenAMIP™ trademark authorizes any OpenAMIP™ implementer to validate your implementation and publish the results, referring to your product by company and product name, if the implementer finds your implementation to be non-compliant. A finding of non-compliance will not be published until thirty days after the OpenAMIP™ member notifies you of the finding. At your option, the implementer's published finding of non-compliance will include a reference to a statement in rebuttal by you.

Audience

The intended audience for this document is an engineering team responsible for integrating a satellite terminal.

Contents

This document contains the following major sections:

- *Introduction*

  This chapter gives an introduction about OpenAMIP.

- *Protocol Specification*

  This chapter describes the protocol specifications, message types, and syntax.…

- *Compatibility*

  This chapter describes the hardware and version compatibility.

- *Test Suite*

  This chapter displays the sample protocols.

- *Acronyms and Abbreviations*

  This list is meant to be generic within this document and may contain acronyms and abbreviations not found in this manual and some terms may not be defined based on industry standards.

- *Glossary*

  This list is meant to be generic within this document and may contain entries not found in this manual and some terms may not be defined based on industry standards.

Document Conventions

This section illustrates and describes the conventions used throughout this document.

Convention	Description	Example
`Command`	Used when the user is required to type a command at a command line prompt or in a console.	Type the command: `cd /etc/snmp/`

`Terminal Output`	Used when showing resulting output from a command that was entered at a command line or on a console.	`crc report all`
		`8350.3235 : DATA CRC [ 1]`
		`8350.3502 : DATA CRC [5818]`
		`8350.4382 : DATA CRC [ 20]`

Screen Reference	Used when referring to text that appears on the screen on a Graphical User Interface (GUI). Used when specifying names of commands, menus, folders, tabs, dialogs, list boxes, and options.	1. To add a remote to an inroute group, right-click the Inroute Group and select **Add Remote**. The **Remote** dialog box has a number of user-selectable tabs across the top. The **Information** tab is visible when the dialog box opens.
Hyperlink	Used to show all hyperlinked text within a document or external links such as web page URLs.	For instructions on adding a line card to the network tree, see *Adding a Line Card* on page 108.

WARNING: A warning highlights an essential operating or maintenance procedure, practice, condition, or statement which, if not strictly observed, could result in injury, death, or long term health hazards.

CAUTION: A caution highlights an essential operating or maintenance procedure, practice, condition, or statement which, if not strictly observed, could result in damage to, or destruction of, equipment or a condition that adversely affects system operation.

NOTE: A note is a statement or other notification that adds, emphasizes, or clarifies essential information of special importance or interest.

Getting Help

The iDirect Technical Assistance Center (TAC) and the iDirect Government Technical Assistance Center (TAC) are available to provide assistance 24 hours a day, 365 days a year. Software user guides, installation procedures, FAQs, and other documents that support iDirect and iDirect Government products are available on the respective TAC Web site:

- Access the iDirect TAC Web site at http://tac.idirect.net
- Access the iDirect Government TAC Web site at http://tac.idirectgov.com

The iDirect TAC may be contacted by telephone or email:

- Telephone: 703.648.8151

- E-mail: tac@idirect.net

The iDirect Government TAC may be contacted by telephone or email:

- Telephone: 703.648.8111
- Email: tac@idirectgov.com

iDirect and iDirect Government produce documentation that are technically accurate, easy to use, and helpful to our customers. Please assist us in improving this document by providing feedback. Send comments to:

- iDirect: techpubs@idirect.net
- iDirect Government: techpubs@idirectgov.com

For sales or product purchasing information contact iDirect Corporate Sales at the following telephone number or e-mail address:

- Telephone: 703.648.8000
- E-mail: sales@idirect.net

1 Introduction

This document describes the Open Antenna Modem Interface Protocol (OpenAMIP™) for satellite terminals. OpenAMIP is an ASCII message-based protocol to exchange information between an antenna controller and a satellite modem. OpenAMIP allows the modem to command the controller to seek a particular satellite. OpenAMIP also allows the modem and controller to exchange information necessary to initiate and maintain communications through the satellite.

OpenAMIP is not intended for any purpose except to permit a modem and a controller to perform synchronized automatic beam selection. It is not a status logging system or a diagnostic system. There is no explicit provision in OpenAMIP for security or validation. The controller and the modem may choose to use any of several security measures at lower protocol layers.

2 Protocol Specification

This chapter contains the following sections:

2.1 Introduction

OpenAMIP is intended to be simple and flexible. Communications are in the form of messages that are readable ASCII characters. A message consists of one or more space-separated variable-length fields. The command is terminated by a new line `<lf>` character or by the `<cr><lf>` sequence.

The first field is a message type, a single alphabetic character in the standard command set. Each type of message requires a specific number of parameters. The last parameter may optionally be separated from the new line by a comment that begins with a `#`. The `#` can be followed by a string containing any characters other than a new line.

The OpenAMIP protocol is a peer protocol: neither side is the master. The messages are sent through any of the several lower-level protocols, such as HTTP, TCP/IP over a LAN, UDP over a LAN, or using a high-speed serial connection.

2.2 Syntax

The OpenAMIP format specified here is in Backus-Naur form (BNF).

```
<msg>::=<msg_body><optional whitespace>'\n'
            | <msg_body><optional whitespace>'#'<comment_body>'\n'
<comment_body>::=<non-newline>
                          |<non-newline><comment_body>
<non_newline>::= {any printable character except '\n'}
```

```
<msg_body>::=<msg_type>
            |  <msg_type>  <param_list>
<param_list>::= <whitespace> <param>
                        | <param><param_list>
<param>::=  <binary>
                  |<float>
                  |<int>
                  |<string>
<binary>::= '1'
                 |'0'
<int>::= '-' <natural>
          | <natural>
<float::=<int>'.'<natural>
           | <int>
<natural>::= <digit>
               |   <digit><natural>
<digit>::=  '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
<string> ::=<string_char>
                |<string_char><string>
<string_char>::={any printable character except ' ' and '\n'}
<optional whitespace>::=NULL|<whitespace>
<whitespace>::=<whitespace_char>|<whitespace><whitespace_char>
<whitespace_char>::= ' '|'\t'|'\r'
```

2.3 Message Types

Table 2-1. Message Types

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
A	Alive interval. Antenna should send a status message at least this often. 0 means never repeat.	1	int interval, seconds	M
B	Beat frequency oscillator (local oscillator) frequencies; effective amount of down-conversion (Rx) or up-conversion (Tx).	2	float Rx LO frequency, MHz float Tx LO frequency, MHz	M
C	Carrier to Noise Ratio for Conical Scan. Reporting rate is configured by "c" message. Default "C" message rate is zero (if "c" message reporting rate parameter is not present, the "C" message is disabled). It is recommended to provide this message by UDP as a separate stream.	5	float CNR (SNR) in dB, measured on headers and pilots	M
			float CNR (SNR) in dB, measured on data	
			float time in seconds. This free-running counter may wrap around through zero periodically; it is recommended to use enough resolution for at least an hour between wrap events	
			int received carrier lock state. Values are in the range 0..7, where 0 means invalid, 1 means not locked, 7 means fully locked, and values in between are intermediate states of lock (intermediate state details are product-specific).	
			float composite power, dBm, measured at the IF input to the modem.	
E	Expected power. Maximum L-band Tx power to be expected at the antenna, in dBm.	1	float max power	M
F	Find the satellite. Antenna should now begin using the satellite specified by S, P, B, X, and H. This command overrides the N command.	0		M

Table 2-1. Message Types (continued)

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
H	Hunt frequency in MHz. Modem expects antenna to use this L-band hunt center frequency when commanded. Receive RF frequency can be determined through combination of RX LO (LNB) frequency in the "B" message and the center frequency in the "H" message.	2	float frequency, float bandwidth	M
I	ID of the modem type (optional)	2	string: modem manufacturer and string: modem model	M
K	Maximum and minimum sKew of the beam short axis to the geosynchronous arc, in degrees. Transmitter should be disabled when these limits are exceeded. Minimum skew defaults to zero if absent.	2	float max skew and float min skew	M
L	Lock status of receiver. The modem should send this message immediately when the status changes. The modem should send this message periodically at intervals specified by the antenna in the "a" message.	2	RX Lock State: binary 1 (locked) or 0 (unlocked)	M
	Antenna is free to transmit, or not. This command may be used by the antenna to remove power from the Tx amplifiers. **NOTE:** The Tx Enable parameter can be used to support a power calibration mode, in which the final power amplifier is disabled or terminated, but the preamplifier is still enabled and capable of measuring RF power at the preamp.		binary TX Enable: 1 (Tx on) or 0 (Tx off)	
N	Non-geosynchronous mode. The antenna should be aimed away from the geosynchronous arc. This is intended to support installation tests such as power measurements. **NOTE:** The N command is intentionally redundant with the L command; it is not intended to be the sole means of preventing interference during tests. This command overrides the F command, but should not cause the antenna to lose the parameters previously specified by S, P, B, X and H.	0		M
P	Polarization. Modem commands antenna to use these polarizations.	2	char Rx Polarization: L, R, V, or H and char Tx polarization: L, R, V, or H	M

Table 2-1. Message Types (continued)

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
S	Satellite longitude. Modem expects antenna to use this satellite when commanded.	3	float longitude (degrees)	M
	Maximum excursion in satellite's latitude (for inclined-orbit satellites)		float latitude variance (degrees)	
	Satellite's nominal polarization offset in degrees (for skewed satellites)		float polarization skew (degrees) From behind the dish, facing towards the satellite; clockwise is positive.	
T	Transmit frequency. Modem intends to transmit at this L-Band frequency and bandwidth	2	float Tx frequency, MHz and float Tx bandwidth, MHz	M
W	Where (location) Interval. Antenna should send w message immediately, and then repeat at least this often. 0 means "never repeat" ***NOTE:*** integer is a valid subset of the float type. Non-integer time is used only for specialized highly dynamic terminals, by agreement between modem and antenna vendor. The modem shall not request non-integer time unless the antenna is known to support it. Decimal point and digits after decimal may be omitted, unless modem and antenna both support high-rate reporting for highly dynamic terminals, and modem has requested rate > 1Hz.	1	int repeat interval, seconds. Should be float, if the modem requests antenna to transmit "w" at rates higher than 1 Hz	M
X	eXtra hunt parameters. This is a fixed string to be configured by the operator and sent as part of the lookup. The antenna vendor specifies the string. If the controller does not need this command, the modem does not need to send it, but the modem may send it anyway, in which case the controller will ignore it.	1	string	M
a	alive interval. Antenna requests to see an L message from the modem at least this often. 0 means "never repeat".	1	int repeat interval, seconds	A

Table 2-1. Message Types (continued)

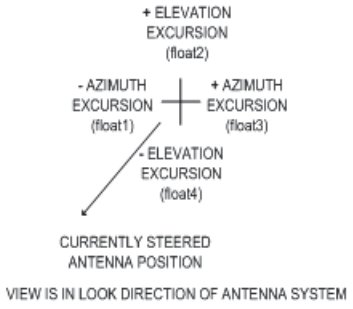
Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
c	conical scan setup (optional) Sent when conical scan performed. The four floating point values represent the times (UTC or GPS epochal) of beam steering excursions from the previously steered coordinates. Azimuth and elevation delta scan excursions are pre-determined by the antenna manufacturer and would be on the order of ±0.25°. The antenna may request periodic carrier-to-noise estimates ("C" message) by setting the fifth parameter. Default "C" message rate is zero (if "c" message reporting rate parameter is not present, the "C" message is disabled). ***NOTE:*** Some terminal implementations will use only the fifth parameter of this message; in that case, the first four parameters may be set to zero, or may optionally be set to describe the intended scan timing.	5	float1 -AZ: see drawing, float2 +EL: see drawing, float3 +AZ: see drawing, float4 -EL: see drawing, and int reporting rate in Hertz for "c" message  	A
i	ID of the antenna type	2	string: manufacturer and string: model	A
r	Reference frequency required for BUC and LNB.Frequency is in MHz.	2	int frequency MHz	A
	Reference Used by Rx (R), Tx (T) or both (B)		string Reference Used By: R, T, or B (e.g. "r 10 B")	

Table 2-1. Message Types (continued)

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
s	Status of the antenna. Antenna sends this immediately in response to the F command from the modem, or immediately whenever either of the two statuses changes, or periodically. The period is set by the A command from the modem. "Not functional" means that the antenna cannot currently operate and will never operate with this configuration. This can be temporary (for example, an illegal configuration) or permanent (for example, motor frozen).	4	binary Antenna Functional: 1 - antenna functional 0 - antenna not functional	A
	"Modem must not transmit" means that the antenna has detected a condition (loss of lock, blockage, cable unwrap, max skew exceeded) that does not require a reconfiguration, but that does require the modem to cease transmission.		Binary Modem May Transmit: 1 - Modem may transmit 0 - Modem must not transmit	
	The third parameter is the number of full sweeps the antenna has performed while searching for the satellite. It should be set to 0 upon receipt of an F command, and incremented when the antenna has performed a full sweep for the satellite. If omitted, this parameter is assumed to be 0. This parameter should be zero if an N command is more recent than an F command.		int Search Count	
	The fourth parameter should be set to 0 if an F command was sent more recently than an N command. If omitted, this parameter is assumed to be 0. **NOTE:** If the antenna cannot ensure it is ready for a transmitter test without regulatory violation, the third parameter should be set to 0.		binary Tx Disabled: antenna (1 - has, 0 - has not) successfully disabled transmission toward the geosynchronous arc (response to N command). If this parameter is "1" antenna is in a state to support installation tests such as power measurements; any power from the transmitter test is either terminated in a dummy load or otherwise prevented from interfering with satellites.	

Table 2-1. Message Types (continued)

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
w	where the platform is located. Antenna sends this to modem periodically. The period is set by the W command from the modem. If the location is not valid, the antenna may put 0 in the remaining parameters.	11	binary Location Valid: 1 - valid 0 - invalid	A
			float latitude (degrees) negative is south	
	The precision of the floating point numbers should reflect the precision of the location information. Typically six to eight digits is desirable for lat/lon values. The antenna should send a w immediately if its internal GPS status changes from "invalid" to "valid".		float longitude (degrees) negative is west of prime meridian	
	If the antenna does not know the time, the time parameter should be set to 0. The time parameter is mandatory if Doppler compensation is to be applied. ***NOTE:*** integer is a valid subset of the float type. Non-integer time is used only for specialized highly dynamic terminals, by agreement between modem and antenna vendor. The modem shall not request non-integer time unless the antenna is known to support it. Decimal point and digits after decimal may be omitted, unless modem and antenna both support high-rate reporting for highly dynamic terminals, and modem has requested rate > 1Hz. If provided, timestamp must correspond to the latitude/longitude values. If omitted, all following values must also be omitted, and Doppler compensation is not possible.		INT time (GPS seconds) time in seconds since the GPS epoch. Should be float, if the modem has requested a non-integer repeat interval less than 1 sec through "W" message	
	If the altitude parameter is not set, it is assumed to be zero.		float altitude (meters)	
			float heading referenced to true north (degrees)	
			float GPS computed speed (m/s)	
			float pitch angle (degrees). Positive is up, negative is down.	
			float roll angle (degrees). Positive is rolled to starboard, negative is rolled to port.	
			float yaw angle (degrees). Positive is inclined to starboard. Negative is inclined to port.	

Table 2-1. Message Types (continued)

Type	Description	No. of Parameter(s)	Name of the Parameters	Sender
			float skew angle (degrees). Positive is CW when facing satellite from ground. Negative is CCW when facing satellite.	

2.4 Physical Layer

2.4.1 TCP Interface

A modem and controller may communicate using TCP either ways. The method of discovering the IP address and TCP port is outside the scope of OpenAMIP. In the reference implementation, the antenna listens on a configured TCP port and accepts calls from a configured (range of) modem IP addresses. The modem initiates a TCP connection to a configured antenna IP address and TCP port.

Whenever the TCP connection is disconnected, the antenna sets its keep-alive timers to infinity. When a new TCP connection is established, the modem will send an ‘A’ to the antenna, and the antenna will send an 'a' to the modem. Typically each side will then set a disconnect timer to three times the requested interval. For example, the modem might send "A 3" and set its disconnect timer to 9 seconds. If at any time after that, the modem waits more than 9 seconds to receive an "s" message, the modem will break the TCP connection. It may then choose to periodically (or at random intervals) attempt to make a new TCP connection. Similarly, the antenna might send "a 2" and then break the connection if it must wait more than 6 seconds between received "L" messages.

Neither the antenna nor the modem is obliged to accept more than one TCP connection at a time, but this is not prohibited. In a system with two modems, one may be acting as a backup. In this arrangement, the antenna should only honor satellite selection requests from one modem.

TCP is a "stream-oriented" protocol: there is no particular mapping of an OpenAMIP message into an IP packet. A single packet may contain a fragment of a message, a complete message, or multiple messages. In the reference implementation, the modem sends an entire initial set of seven messages in a single POSIX "write" command immediately after opening the connection. On most POSIX systems, this will result in a single TCP/IP packet. The reference receiver implementation accumulates characters until a new line is found and then processes the result as an OpenAMIP message. Accumulation of the next message starts with the first character after the new line.

2.4.2 UDP Interface

Each message fits in a single UDP packet. A packet may contain more than one message, but any given message must be fully contained within one packet. The antenna has a configured IP address and well-known port, as does the modem. The initial state of the OpenAMIP interface is "idle" (that is, no keepalive) until the partner sends a keepalive timer. The interface reverts to the "idle" state if three keepalives are missed.

In Version 1.9, the modem may create a UDP stream for the "C" message, alongside the primary TCP connection for all other OpenAMIP messages. Because the "C" message will be sent at a relatively high rate (tens of Hertz), the UDP stream is more practical; it avoids TCP handshaking overhead. It is recommended that the modem and antenna use the same IP address and port for both TCP and UDP connections. Because TCP and UDP use separate address spaces, this does not cause any conflict.

2.4.3 Asynchronous Serial Interface

This is beyond the scope of OpenAMIP. However, SLIP can be used to establish an IP connection on the serial link. Alternatively, any packet-over-serial technique may be used. (Note that a checksum should be used here.)

2.5 Semantics

The protocol is primarily intended to convey state change information based on external events. The following notes are intended to provide functional guidance for various common events and message sequences. It is not intended to be a comprehensive list of messages nor a syntax dictionary.

To comply with regulatory constraints, the modem must disable its transmitter within 100ms when the antenna loses lock on a satellite, and must also disable the transmitter immediately when a blockage occurs. The antenna must minimize the interval between detecting a change in condition and sending the status message to the modem. Similarly, the antenna may choose to use the modem lock signal as part of its satellite search. The modem must also minimize the interval between detecting the condition and sending the message to the controller. Status changes should be reported within 10ms. However, since this will not be practical on a slow serial link, the links are deprecated.

Prior to any communication between the modem and the controller, the OpenAMIP state is unspecified. The timers are all set to infinite. The modem initiates communications by sending the commands needed to deliver the satellite parameters to the controller. It then sends an "F" message.

When the controller receives an "F" message, it must respond within 10 milliseconds with an "s" message. This is necessary to ensure regulatory compliance in the case when the modem needs to mute. The controller must also send a status every "keepalive" interval, and every time the status changes. When the controller responds to an "F" message, the "may transmit" status must reflect the status with respect to the newly-selected satellite parameters. This means that if the modem has just commanded the antenna to "Find" the satellite that it is already tracking and is already locked on, then the immediate status can be "may transmit". However, if the antenna is already tracking a satellite and is successfully locked to it, and the modem then sends new parameters and issues a new "Find" command, the controller must immediately send a status of "must not transmit" because it is not locked to the new satellite (it is locked to the old satellite). After the antenna locks to the new satellite, it will send a new status message indicating that the modem may transmit.

The modem should send an "L" message whenever the modem lock changes. It should also send the "locked" status every time its keepalive timer expires. Whenever the modem sends the "L" message for any reason, it restarts its keepalive timer.

When the modem issues a "W", the controller immediately responds with a "w". The controller responds thereafter every w seconds (zero seconds means never). If the controller sends a "w" to the modem which indicates that the location information is invalid, the controller should send a new "w" message immediately as soon as valid location information becomes available.

Latitude and longitude are reported in floating point decimal degrees. The range for latitude is -90.0 to 90.0, where -90.0 is the South Pole. The range for longitude is 360.0 to 360.0, where negative is west from the prime meridian and positive is east from the prime meridian. The overlap is intentional: the sender is free to use zero to 360 or -180 to 180 (or even -360 to 0 or a mixed system). The receiver must be able to handle the full -360 to 360. Leading zeros are optional for the sender, except that the number must have at least one digit before the decimal point. Trailing zeros are optional for the sender, except that the number must have at least one digit after the decimal. The receiver must be able to handle leading and trailing zeros correctly. If the fractional part is zero, the number may be specified as an integer (that is, without a decimal point). Note that the syntax does not permit the use of the "+" character.

The precision of the latitude and longitude is not specified by the OpenAMIP syntax; the number of digits after the decimal point is arbitrary. However, the sender should provide as much precision as is actually available. As a practical matter, OpenAMIP contemplates the ability to use this information for logging and transmission restrictions as mandated by regulatory authorities, so accuracy to about one kilometer is needed: this implies that latitudes and longitudes to a precision of one thousandth of a degree are needed.

If the modem issues a "P", "B", or "F" command that is incompatible with the antenna hardware, the antenna may either ignore the incompatible parts of the command or may set the "functional" status to "not functional".

The "K" message conveys the maximum skew of the short axis of a non-circular beam to the geosynchronous arc. If the antenna has a beam shape that is radially symmetric about the bore sight, this parameter may be ignored. Otherwise, the antenna must use the current skew as a factor in computing the "must not transmit" or "may transmit" status. When all other factors permit transmission, the antenna will immediately send a status message with a status of "must not transmit" when the angle transitions from below to above the maximum skew, and will immediately send a status message with a status of "may transmit" when the angle transitions from above to below the maximum skew. In contrast to some other messages, the "K" message takes effect immediately and the modem may send a new "K" message with a new max skew angle at any time. The "K" message also includes a minimum skew parameter, to support protection of non-geostationary satellites. The minimum skew parameter operates analogously to the maximum skew parameter; the antenna controller should send a status of "must not transmit" when the skew is less than this value.

When the antenna reports with an "s" message that the antenna is functional, it indicates that the antenna should currently be working. "Non-functional" means that the antenna is not currently in service. This does not include blockage, loss of lock, system initialization, loss of heading information, cable unwrap, or any condition that can correct itself without intervention. It does include detection of a fatal mechanical failure, or an operator command to the antenna controller from its front panel or other source, or an illegal configuration.

When the modem detects this status, it will not attempt to recover by, for example, switching to a different satellite or clearing and re-establishing the OpenAMIP connection. The modem waits until the antenna sends a "functional" message. The antenna provides a "may transmit" when it is locked on the satellite and ready to transmit. The antenna signals "must not

transmit" if there is any reason the modem should not transmit: blockage, loss of lock, cable unwrap, sea too rough, etc.

2.6 Examples

This section is intended to describe the purpose of each message. The formal syntax and semantics are described in later sections. Note that the messages here make use of the "comment" syntax. It is unlikely that operational implementations of the protocol will ever transmit messages with comments, but they are useful in descriptive documents such as this one and in test scripts. Typically, implementations of the receive side of the protocol will properly detect and ignore comments.

The modem must be able to convey all of the information needed by the controller to describe a satellite. This must be sufficient for the controller to identify the satellite and to command the controller to find the satellite.

2.6.1 Messages from Modem to Antenna Controller

"Keepalive" messages are sent to the modem regularly to ensure that communications connectivity with the controller are not lost.

```
A 10 # Alive: Antenna should resend status "s" every N seconds.

B 9750.0 12800.0  # "Beat Frequency": downconversion & upconversion
offsets: floating point in MHz.

E 0.5 # Expected power: Maximum L-band Tx power to be expected at the
antenna, in dBm.

F #F: Find. Use the recent S, P, B, X, and H parameters.

H 1123.321 0.256  # Hunt: floating point center frequency and bandwidth
in MHz.
```

The modem informs the controller when the modem has detected the downstream carrier:

```
I iDirect 5100 # ID: modem manufacturer and type strings.

K 45 15 # sKew: maximum and minimum skew.  The antenna controller must
disable transmission when outside these angles (in degrees).  This is
typically used with non-circular apertures.

L 1 1 # Lock status: Rx locked (1 is locked, 0 is unlocked), Tx OK (1
means antenna MAY transmit; 0 means antenna MUST NOT transmit).

N # Non-geosynchronous mode. No transmission. The antenna should be
placed in a state to aimed away from the geosynchronous arc. This is
intended to support installation tests such as power measurements.

P L R #Polarization: H, V, L or R for Rx and Tx, respectively.

S -20.1 1.0 3.5 # Satellite longitude:  All parameters are floating
point degrees, "-" is West. Wander in latitude is 1.0. Polarization
skew 3.5.

T 1450 4.5 # Transmit frequency:  The modem intends to transmit at this
L-Band frequency and bandwidth.
```

The modem requests periodic location information:

```
W 1 # Where: Antenna should send "w" location report every N seconds.

X nid=1234 # Xtra string: vendor-specific string for antenna
controller.
```

2.6.2 Messages from Antenna Controller to Modem

The controller provides status information to the modem such as, when it is locked onto the satellite, when it is functional and unblocked, how many attempts has it made to search for the satellite and (for installation support) when it is in a safe state for dummy transmission measurements. The controller sends an "s" message immediately after receiving an "F" message, and periodically at the interval defined by the "A" message:

```
a 60 # alive: modem should send keepalive messages every N seconds.

c 0.25 0.25 0.33 0.33 # conical scan setup: not supported by iDirect;
included as a placeholder for compatibility with other vendors'
systems.

i YoyoDyne 1234 # ID: antenna controller manufacturer and type strings

r 10 B # reference frequency required for BUC and LNB; not currently
supported by iDirect; included as a placeholder for compatibility with
other vendors' systems.

s 1 1 1 0 # s: four parameters: functional, OK-to-transmit, searched
once, not in transmitter test mode.
```

The antenna controller sends GPS information to the modem:

```
w 1 -10.123 20.235 123456789 10000 91.0 223.52 0.10 -0.51 91.0 # where:
location report. valid, lat, lon, time, altitude, heading, speed,
pitch, roll, yaw.
```

The "w" message parameters require more explanation:

- Valid (1) or invalid (0)
- Latitude in floating point degrees (South is negative)
- Longitude in floating point degrees (West is negative)
- GPS time in seconds; if the antenna does not have GPS time, set this to zero
- Altitude, heading, speed, pitch, roll, yaw are not physically required for system operation, but support logging for regulatory compliance and system performance management

3 Compatibility

This chapter contains the following sections:

3.1 Version Compatibility

New versions of the OpenAMIP protocol may be published. New versions will be strict supersets of older versions and may extend the protocol in only two ways:

- A new version may add new message types
- A new version may add new parameters to the end of an existing message type

Do not use any other syntactic extensions. Any extension to the semantics of the protocol must not affect the semantics of earlier versions. The intent of this specification is that any older implementation of the protocol can interoperate with any newer implementation without loss of any of the older functionality. A compliant implementation of OpenAMIP must ignore any unexpected message type that it receives, and must ignore any unexpected parameters at the end of a message. Furthermore, a compliant implementation must operate successfully if it receives a message with too few parameters. Parameters that are added to the protocol in version 1.5 or later will have default values that the receiver will use if a message does not provide the parameter.

New versions of the protocol are required to be backward-compatible with older versions. This is ensured by requiring that the meanings of parameters never change from version to version. New parameters may be added to a message, and new messages may be added. The receiver is required to ignore extra parameters and unknown messages; this allows an older receiver version to work with a newer sender. The receiver is required to operate properly when it receives a message that does not have enough parameters; this allows a newer receiver version to work with an older sender (the older version will not implement functionality that requires the newer version), but the older version will continue to provide its functionality when operating with a partner that is using a newer version.

3.1.1 Version Changes

This section provides a history of the changes between each successive pair of versions of the OpenAMIP Standard. Since the standard only allows for new commands, or new parameters to

existing commands, but never a removal of an existing command or parameter, each subsection will list those commands and parameters which are new in that version.

For the full definition of these new commands and parameters, see *Semantics* on page 12.

Table 3-1. Version Changes

Version Number	What's changed?
1.7	c - new message to support conical scan setup. K - added one new parameter: minimum skew parameter for protection of non-geostationary satellites. N - new message to support transmitted power measurements. r - new message for reference frequency selection. s - added two new parameters: for search count and "N" command confirmation. w - added six new parameters: for altitude, heading, speed, pitch, roll, and yaw.
1.8	Reformatted for clarity require - added note that the command allows transmission to be disabled for transmitter calibration.
1.9	Added "C" message to report CNR (SNR) at high speed. Added parameter to existing "c" message to enable control of the "C" message. Time parameter of the "w" message has been changed to the float type, and the repeat interval parameter of the "W" message has also been changed to the float type.
1.10	Added "lock status" parameter to "C" message.
1.11	Added wideband signal strength to "C" message.
1.12	Changed wideband signal strength to composite power in "C". Corrected number of parameters in "C" and added "skew" parameter to "w" message. Removed mandatory requirement for non-zero timestamp in "w" message introduced in 1.8, to maintain backward compatibility. Updated semantics section for "w" message. Updated “H” message.

3.2 Modified OpenAMIP

Any antenna or modem manufacturer can extend the protocol by creating an extended type field. The extended type field consists of the manufacturer's name (with no spaces) followed by a colon, followed by a type (with no spaces). If a modem or antenna controller receives a message of unknown type, the modem or antenna controller will ignore the message. If the

messages are optional for operation of the equipment, then the protocol still qualifies as "unmodified" OpenAMIP. If the messages must be used for a particular antenna or modem, then the resulting implementation must be called "modified OpenAMIP".

Examples:

```
Yoyodyne:NID 1132  # additional search parameter
iDirect:stow 1 # command specified by iDirect
```

3.3 Hardware Compatibility

OpenAMIP is intended for a typical installation with a specific modem and a specific antenna are installed and configured to work together. The protocol does not make provision for auto-discovery or parameter negotiation. These are installation issues and the protocol was developed to focus on operations. It is the responsibility of the installer to assure that the parameters are compatible. Essentially all incompatibilities will cause loss of service and the need for intervention, so the mechanisms needed for auto-negotiation have no practical benefit. The obvious examples of incompatibilities occur in the "P", "H", and "B" commands. An antenna that is mechanically configured for LHCP and that has no polarization switch hardware will not operate correctly for RHCP or linear polarization. Similarly, an antenna with a mechanical polarizer will not be able to select Tx polarization independently from Rx polarization. Similarly, an antenna whose downconversion offset frequency ("LNB local oscillator") is fixed cannot implement a B command to change to another frequency, and more generally an antenna with a selectable downconversion frequency can only change to one of a small set of downconversion frequencies. Finally, an antenna whose tracking receiver supports a specific set of (one or more) bandwidths cannot select an arbitrary hunt bandwidth. It is the responsibility of the installer to ensure that the modem does not send parameters that the antenna does not support. For the hunt bandwidth, the antenna may choose to operate with a different hunt bandwidth. Do not operate the antenna for other unsupported "P", "B", and "H" parameters. When the antenna does not have a controllable down conversion frequency, the antenna may choose to ignore the "B" command. The modem may choose to not send the B command.

4 Test Suite

This chapter contains the following sections:

iDirect provides reference implementations in C. No representations are made that these are suitable for use in any product. Semantics may be validated by executing a script that emulates a controller or a modem. The scripts are written in POSIX-compliant C. Code for the test suite was developed from the reference implementation. The source code for the reference implementations and the test scripts is copyrighted by iDirect but is licensed at no cost for use for any purpose.

4.1 Modem Module Reference Design

The modem implements the protocol as follows: The modem reads the antenna's IP address and TCP port number from a configuration file. The modem attempts to connect to the antenna through TCP: if the connection fails, the modem attempts to re-establish it. Whenever the modem succeeds in connecting to the antenna, it sends a set of setup commands. These commands are sent "back-to-back" with no intervening commands and without waiting for responses: the commands are "S", "H", "P", "B", "X", "A", "F", "W", and "L". The modem then waits for messages from the antenna. The modem sends an "L" whenever its lock state changes. If the modem receives an "a", it will send an L periodically. If the modem does not receive an "s" or a "w" at the expected periodic intervals (based on its "A" and "W" requests), it clears the TCP connection and attempts to re-establish it, and the cycle repeats. If the modem decides to switch to a different satellite, it sends the setup sequence again.

4.2 OpenAMIP_sim.c

```
/*
  "Reference implementation" of the Antenna controller's
OpenAMIP(tm) protocol processing code.
   ---- begin notice----
   Copyright (c) 2007, 2008 iDirect technologies iDirect hereby
licences anyone to use this code, modified or unmodified, for any
```

```
   purpose, providing that  this notice is retained in the source
code.

   The use of the trademarked name OpenAMIP(tm) is restricted and
may not be used except under the terms of a separate licence: see
www.OpenAMIP.org for details.

   In particular, you are prohibited from using the name OpenAMIP
to describe a protocol that does not comply with the standard.
Therefore, if you modify the code so that the result is not an
implementation of the protocol, you must remove any output
messages that refer to the name.

   ---- end notice---

   This program is a trivial implementation for the Antenna-
controller end of the OpenAMIP protocol. It receives and parses
input from a modem and it responds properly. The program makes
almost no attempt to simulate a real antenna controller.

   The only such "simulation" is the response to an "F" message.
If none of the satellite values have changed since the prior "F"
message, then the "locked" status is true. If any of the satellite
values have changed, the "locked" status is false and remains
false for 20 seconds. Note that we respond to an "F" message with
an "s" response that reflects the new "locked" status, not the old
locked status.

   The program is "forgiving", in that it ignores any message type
that it cannot understand. This is as specified by the protocol
definition document.

   Optionally, the program can log unknown messages to the
standard output, and/or send a comment message back to the modem
when it receives a unknown message.

   The program should compile and run on any POSIX-compliant
system, but has been tested only on Linux (specifically RHEL) and
Cygwin under Windows.
 */
#include <sys/select.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
```

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#define FALSE  0
#define TRUE   1

static char * version="2008-02-14";

// satellite location parameters from the modem:
static float  freq=0.0,hunt_bw=0.0;
static char   pol_rx=0,pol_tx=0;
static double sat_lon=0.0,sat_io_lat=0.0,sat_skew=0.0;
static double rx_lo=0.0,tx_lo=0.0;
static int    modem_state,tx_ctl;

static int changed=TRUE;  //internal variable to track whether or
not we changed a satellite param.

// status variables to report to the modem.
static int locked=FALSE;
static int functional=TRUE;
static int blocked=FALSE;

// arrays for the timer event system.
#define FAR_FUTURE ((time_t)0x7FFF0000)
enum {SWING,GPS,STAT,MAXTIMER};
static int intervals[MAXTIMER];
static int timers[MAXTIMER];

//commandline parameters:
static int    verbose=TRUE;
static int    tcp_port=5005;
static int    my_ip=INADDR_ANY;
static double lat,lon;
```

```
//Handle a message received from the modem. A "Message" is a
sequence of
//  characters ending in a newline.
static void process_message(char *buff,int now)
{
  // define the message keywords. The current keywords are all
single character, but
  // the syntax permits a variable-length word, so use an
enumerated word list to
  // simplify any later expansion of the protocol.
  static char *keywords[]=
       {"S", "H","P",    "F",     "W",      "A",       "L", "B",
"X",  0};
  enum
{SAT,HUNT,POL,EXECUTE,LOC_TIME,STAT_TIME,MODEM_STAT,BAND,XTEND,MAX
};


  int len;
  int i,np;
  double tmpfla,tmpflb,tmpflc;
  char tmpchra, tmpchrb;


  len=strcspn(buff," \n#\r\t");  // get the length of the message
keyword.
  if(len==0)
    return;                     // no keyword--ignore the message:
it's a comment
  for(i=0;keywords[i]!=0;i++)
    if(  (len==strlen(keywords[i]))
         &&(0==strncmp(keywords[i],buff,len))
      )
      break;
  switch (i)
  {case SAT:
   tmpfla=tmpflb=tmpflc=0.0;         //preload defaults;
     np=sscanf(buff+len,"%lf %lf %lf" ,
               &tmpfla,&tmpflb,&tmpflc);  //get new satellite
lon, io_lat, and skew

if((tmpfla!=sat_lon)||(tmpflb!=sat_io_lat)||(tmpflc!=sat_skew))
```

```
       changed=TRUE;                          //set "changed" if
different
     sat_lon=tmpfla;
     sat_io_lat=tmpflb;
     sat_skew=tmpflc;
     break;
   case HUNT:
     np=sscanf(buff+len,"%lf %lf",&tmpfla,&tmpflb);  //get new
hunt frequency
     if((tmpfla!=freq)||(tmpflb!=hunt_bw))
       changed=TRUE;                            //set "changed" if
different
     freq=tmpfla;
     hunt_bw=tmpflb;
     break;
   case POL:
     tmpchra='H',tmpchrb='V';
     np=sscanf(buff+len,"%c %c",&tmpchra,&tmpchrb);     //get new
pol.
     if((tmpchra!=pol_rx)||(tmpchrb!=pol_tx));
       changed=TRUE;                            //set "changed" if
different
     pol_rx=tmpchra;
     pol_tx=tmpchrb;
     break;
   case BAND:
     np=sscanf(buff+len,"%lf %lf",&tmpfla,&tmpflb);  //get new
hunt frequency
     if((tmpfla!=rx_lo)||(tmpflb!=tx_lo))
       changed=TRUE;                            //set "changed" if
different
     rx_lo=tmpfla;
     tx_lo=tmpflb;
     break;
   case XTEND:
    break;
   case EXECUTE:
     if(changed)                              // move to new sat if
it really is new.
```

```
     { timers[SWING]=now+20;               //  tell the modem
after we lock.
       locked=0;                           // we are not locked on
the new sat.
     }
     changed=FALSE;
     timers[STAT]=now;                     // post an event to
send a status message immediately.
     break;
   case LOC_TIME:
     timers[GPS]=now;                          //post an event to
send a GPS message immediately
     np=sscanf(buff+len,"%d",&intervals[GPS]); //save the new GPS
reporting interval.
     break;
   case STAT_TIME:
     timers[STAT]=now;                         // post an event to
send a status message immediately.
     np=sscanf(buff+len,"%d",&intervals[STAT]); //save the new
status reporting interval
     break;
   case MODEM_STAT:
     np=sscanf(buff+len,"%d %d",&modem_state,&tx_ctl); //save the
new modem state and tx ctl
     break;
   case MAX:                                    //ignore unknown
messages
     break;
  }
}

static void send_msg(char *buff,int sock)
{ write(sock, buff, strlen(buff));            // send message
  if(verbose)
    printf("--> %s",buff);
}

/* We are about to go back to sleep. See if we need to do anything
first,
   and then decide when we should wake back up.
```

```
   We get the current time and check against each timer. If any
timer has expired, we process that timer, which may set another
timer. After expired timers are processed, we find the next timer
that will expire and compute and return the interval from now
until then.
*/
static time_t process_timers(int sock,time_t now)
{
  int    i;
  time_t next=now+3600;  //preload a long sleep interval.
  int    avail=1;
  char buff[100];

  if(timers[GPS]<now)
  { sprintf(buff,"w %d %3.3f %3.3f %d\n",avail,lat,lon,(unsigned
int)now);
    send_msg(buff,sock);
    timers[GPS]=now+intervals[GPS];
  }
  if(timers[STAT]<now)
  { sprintf(buff,"s %d %d\n",functional,locked&!blocked);
    send_msg(buff,sock);
    timers[STAT]=now+intervals[STAT];
  }
  if(timers[SWING]<now)
  { locked=1;
    timers[STAT]=now;
    timers[SWING]=FAR_FUTURE;
  }
  for (i=0;i<MAXTIMER;i++)
    if(timers[i]<next)
       next=timers[i];
  return (next-now);
}

/* event loop for connected state. Entered when a TCP connection
is established, exits when the connection dies.
   sleep until data arrives or a timer expires. If data arrives,
the routine accumulates it
```

```
   and delivers it one line at a time to the message processor.
Timer expiries go to the
   timer processor, which computes the next sleep interval.
 */
static void handle_events(int sock)
{
  static char readbuf[1000];
  static char *scan_pt=readbuf;
  static char *read_pt=readbuf;
  char *eol;
  fd_set read_fds;
  struct timeval now;
  struct timeval tv;
  int retval;
  int i;
  int len;

  gettimeofday(&now,0);
  for(i=0;i<MAXTIMER;i++)
  { intervals[i]=3600;
    timers[i]=FAR_FUTURE;
  }
  while(TRUE)
  { FD_ZERO(&read_fds);
    FD_SET(sock,&read_fds);
    tv.tv_sec=process_timers(sock,now.tv_sec);
    tv.tv_usec=0;
    retval=select(sock+1,&read_fds,NULL,NULL,&tv);  //sleep here
    if(retval==-1)
    { perror("select()");
      exit (1);
    }
    gettimeofday(&now,0);
    if(retval==1)
    { if(0>=(len = read(sock,read_pt, readbuf+999-read_pt)))
        return;    //connection is closed
      read_pt+=len;
      *read_pt=0;
```

```
      while(0!=(eol=index(scan_pt,'\n')))   //is there a line in
the buffer?
      { *eol=0;
        if(verbose)
          printf("<-- %s\n",scan_pt);
        *eol='\n';
        process_message(scan_pt,now.tv_sec);
        scan_pt=eol+1;
        if(scan_pt>=read_pt)
        { read_pt=scan_pt=readbuf;
          *read_pt=0;
        }
      }
    }
  }
}


#define err(s) printf("error: %s\n",s)
static int process_args(int argc, char **argv)
{
  char *parm;
  char c;

  *(argv++);
  while (--argc)
  { parm= *(argv++);
    if (parm[0]=='-')
    { while( (c=*++parm) )
        switch(c)
        { case 'p':
            tcp_port=atoi(*(argv++));
            if(parm[1]!='\0')
            { err("\'p\' must be last");
              return(FALSE);
            }
            if(!argc--)
            { err("missing port number after \'p\'");
```

```
                return(FALSE);
              }
              break;
            case 'i':
              if(!argc--)
              { err("missing ip address after \'i\'");
                return(FALSE);
              }
              if(-1==(my_ip=inet_addr(*(argv++))))
              { err("ip address is invalid");
                return(FALSE);
              }
              break;
            case 'v':
              verbose=TRUE;
              break;
            case 'l':
              argc-=2;
              if(argc<=0)
              { err("\'l\' must be followed by two parameters");
                return(FALSE);
              }
              lat=atof(*(argv++));
              lon=atof(*(argv++));
              break;
            default:
              err("unknown flag");
              //fall through
            case 'h':
              printf("OpenAMIP(TM) Antenna controller simulator
version %s\n"
                     "Usage: OpenAMIP_sim [options...] \n"
                     " [options...] are any combination of:\n"
                     "   -i <my_ip>        --ip address on which to
listen.\n"
                     "   -p <port>         --listen on this port\n"
                     "   -l <lat> <lon>    --antenna's lat and
lon\n"
                     "   -v                --verbose\n"
```

```
                    "   -h                --print this message\n",
                    version
                   );
            return(FALSE);
          }
       }
  }
  return(TRUE);
}

/* main program. Contains the listen/accept loop.
 */
int main(int argc, char* argv[])
{
  int sock;         // handle to connected socket
  int server_sock;  // handle to listen socket
  struct sockaddr_in
    address;          // Internet socket address stuct
  socklen_t addr_size  =sizeof(address);
#define SOCKET_ERROR       -1
#define QUEUE_SIZE          5

  process_args(argc,argv);
  printf("Starting TCP server\n");
  server_sock = socket(AF_INET, SOCK_STREAM, 0);  // make a socket
  if(server_sock == SOCKET_ERROR)
  { printf("ERROR: Could not make a socket\n");
    return 1;
  }
  // fill address struct
  address.sin_addr.s_addr = my_ip;
  address.sin_port = htons(tcp_port);
  address.sin_family = AF_INET;

  if(SOCKET_ERROR==bind(server_sock, (struct sockaddr*)&address,
addr_size) )
  { printf("ERROR: Could not bind socket\n");
    return 1;
```

```
  }
  getsockname(server_sock, (struct sockaddr *) &address,
&addr_size);
  // establish listen queue
  if(SOCKET_ERROR==listen(server_sock, QUEUE_SIZE))
  { printf("ERROR: Could not listen\n");
    return 1;
  }
  // we are now a TCP server. listen for a connection. If it
closes,
  // listen for a connection...
  for(;;)
  { while(SOCKET_ERROR==(sock = accept(server_sock, (struct
sockaddr*)&address, &addr_size)))
      printf(" Accept failed with %s\n",strerror(errno));
    if (verbose)
       printf("---- Opened socket\n");
    // now go handle all the events. The routine will not return
here until the call is disconnected
    handle_events(sock);
    // close socket
    if(verbose)
      printf("==== Closing Socket\n");
    if(SOCKET_ERROR==close(sock))
    { printf("ERROR: Could not close socket\n");
      return 1;
    }

  }
  return 0;
}
?
```

4.3 OpenAMIP_modem.c

```
/*
  "Reference implementation" of the modem's OpenAMIP(tm) protocol
processing code.
   ---- begin notice----
   Copyright (c) 2007, 2008 iDirect technologies
```

```
   iDirect hereby licences anyone to use this code, modified or
unmodified, for any purpose, providing that  this notice is
retained in the source code.


   The use of the trademarked name OpenAMIP(tm) is restricted and
may not be used except under the terms of a separate licence: see
www.OpenAMIP.org for details.

   In particular, you are prohibited from using the name OpenAMIP
to describe a protocol that does not comply with the standard.
Therefore, if you modify the code so that the result is not an
implementation of the protocol, you must remove any output
messages that refer to the name.

   ---- end notice---


   This program is a trivial implementation for the modem end of
the OpenAMIP protocol. It is intended as a simple example for aid
in implementing the protocol in a modem and as a simple tester
when developing the protocol for an antenna. It makes a TCP
connection to the  antenna controller and then sends a set of
messages to select a satellite and solicit periodic status and GPS
messages. It receives and prints all messages from the antenna
controller.


   The program is "forgiving", in that it ignores message types
that it cannot understand. This is as specified by the protocol
definition document.


   Optionally, the program can log unknown messages to the
standard output, and/or send a comment message back to the antenna
controller when it receives a unknown message.


   The program should compile and run on any POSIX-compliant
system, but has been tested only on Linux (specifically RHEL.)
 */


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <sys/select.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

#define FALSE 0
#define TRUE 1

static char * version="2008-02-14";

// satellite location parameters from the modem:
static float  freq_center=1000.0, freq_bandwidth=1.0;
static char   pol_rx='V',pol_tx='H';
static double sat_lon=123.5, sat_io_lat=0.0, sat_pol_skew=0.0;
static double rx_lo=10750.0,tx_lo=14000.0;
static int    modem_state=FALSE;

// status from the antenna
static int locked=FALSE;
static int functional=FALSE;
static int blocked=FALSE;

// arrays for the timer event system.
#define FAR_FUTURE ((time_t)0x7FFF0000)     //64K seconds before
rollover, in 2038
enum {LOCK,          //timer before sending our next "L" to
antenna.
      GPS,           //we expect a "w" from the antenna before
this expires
      STAT,          //we expect a "s" fron the antenna before
this expires
      MAXTIMER
    };
static int intervals[MAXTIMER];      //timer reload values
static int timers[MAXTIMER];         //next expiry time for each
timer
```

```
//commandline parameters:
static int    verbose=TRUE;                //if set, print more
stuff
static int    antenna_tcp_port=5005;       //tcp port for the
antenna
static char  *antenna_ip="127.0.0.1";      //antenna's IP address
static int    antenna_alive_interval = 30; //keepalive timer
static int    loc_interval=600;

static double lat,lon;

//Handle a message received from the antenna.
static void process_message(char *buff,int now)
{
  // define the message keywords. The current keywords are all
single character, but
  // the syntax permits a variable-length word, so use an
enumerated word list to
  // simplify any later expansion of the protocol.
  enum                    {STAT,LOC,LOCK_TIME,MAX};
  static char *keywords[]={"s" ,"w","a"      ,0};

  int len;
  int i,np;

  len=strcspn(buff," \n#\r\t");           // get the length of the
message keyword.
  if(len==0)
    return;                               // no keyword--ignore
the message: it's a comment
  for(i=0;keywords[i]!=0;i++)
    if(  (len==strlen(keywords[i]))
         &&(0==strncmp(keywords[i],buff,len))
      )
      break;
  switch (i)
  {case STAT:
     timers[STAT]=now+intervals[STAT];    //reset the timeout for
"alive"
     break;
```

```
   case LOC:
     timers[GPS]=now+intervals[GPS];      //reset the timeout for
"expected GPS"
     break;
   case LOCK_TIME:
     np=sscanf(buff+len,"%d",&intervals[LOCK]);
     timers[LOCK]=now;
     break;
   case MAX:                              //ignore unknown
messages
     break;
  }
}

static void send_msg(char *buff,int sock)
{ int len=strlen(buff);

  if(len!=write(sock, buff, len))          // send message
     perror("Write to socket failed");
  if(verbose)
    printf("--> %s",buff);
}

/* We are about to go back to sleep. See if we need to do anything
first,
   and then decide when we should wake back up.

   We get the current time and check against each timer. If any
timer has
   expired, we process that timer, which may set another timer.
After expired
   timers are processed, we find the next timer that will expire
and compute
   and return the interval from now until then.
*/
static time_t process_timers(int sock,time_t now)
{
  int    i;
  time_t next=FAR_FUTURE;  //preload a long sleep interval.
```

```
  char buff[100];

  if(timers[GPS]<now)
  { printf("Location timeout!\n");
    timers[GPS]=now+intervals[GPS];
  }
  if(timers[STAT]<now)
  { printf("Keepalive timeout!\n");
    timers[STAT]=now+intervals[STAT];
  }
  if(timers[LOCK]<now)
  { sprintf(buff,"L %d\n",modem_state);
    timers[LOCK]=now+intervals[LOCK];
  }
  for (i=0;i<MAXTIMER;i++)
    if(timers[i]<next)
       next=timers[i];
  return (next-now);
}

/* processing for connected state. Entered when a TCP connection
is established, exits
   when the connection dies.
   Start by initializing the timers, then send the satellite find
sequence, then enter
   the event loop.
   Event loop:sleep until data arrives or a timer expires. If data
arrives, the routine accumulates it
   and delivers it one line at a time to the message processor.
Timer expiries go to the
   timer processor, which computes the next sleep interval.
 */
static void handle_events(int sock)
{
  static char buff[1000];
  static char *scan_pt=buff;
  static char *read_pt=buff;
  char *eol;
  fd_set read_fds;
```

```
struct timeval now_st;
struct timeval tv;
int retval;
int i;
int len;
time_t now;

gettimeofday(&now_st,0);
now=now_st.tv_sec;
for(i=0;i<MAXTIMER;i++)
{ intervals[i]=3600;
  timers[i]=FAR_FUTURE;
}
sprintf(buff,"S %3.2f %1.2f %1.2f\n"
             "H %5.3f %5.3f\n"
             "P %c %c\n"
             "B %5.3f %2.3f\n"
             "F\n"
             "A 10\n"
             "W 300\n",
        sat_lon, sat_io_lat,sat_pol_skew,
        freq_center,freq_bandwidth,
        pol_rx,pol_tx,
        rx_lo,tx_lo
       );
send_msg(buff,sock);
timers[GPS]=now+600+1;
intervals[GPS]=601;
timers[STAT]=now+21;
intervals[STAT]=20;
while(TRUE)
{ FD_ZERO(&read_fds);
  FD_SET(sock,&read_fds);
  tv.tv_sec=process_timers(sock,now);
  tv.tv_usec=0;
  retval=select(sock+1,&read_fds,NULL,NULL,&tv);  //sleep here
  if(retval==-1)
  { perror("select()");
```

```
      exit (1);
    }
    gettimeofday(&now_st,0);
    now=now_st.tv_sec;
    if(retval==1)
    {  if(0>=(len = read(sock,read_pt, buff+999-read_pt)))
         return;         //connection is closed
       read_pt+=len;
       *read_pt=0;
       while(0!=(eol=index(scan_pt,'\n')))   //is there a line in
the buffer?
       { *eol=0;
         if(verbose)
           printf("<-- %s\n",scan_pt);
         *eol='\n';
         process_message(scan_pt,now);
         scan_pt=eol+1;
         if(scan_pt>=read_pt)
   { read_pt=scan_pt=buff;
           *buff=0;
         }
       }
    }
  }
}

#define err(s) printf("error: %s\n",s)
static int process_args(int argc, char **argv)
{
  char *parm;
  char c;

  *(argv++);
  while (--argc)
    {
      parm= *(argv++);
      if (parm[0]=='-')
      { while( (c=*++parm) )
```

```
            switch(c)
            {
           case 'p':
             antenna_tcp_port=atoi(*(argv++));
              if(parm[1]!='\0')
              {  err("\'p\' must be last");
                 return(FALSE);
              }
              if(!argc--)
              { err("missing port number after \'p\'");
                return(FALSE);
              }
             break;
           case 'i':
              if(!argc--)
              { err("missing ip address after \'i\'");
                return(FALSE);
              }
              antenna_ip= *argv++;
            break;
            case 'v':
              verbose=TRUE;
              break;
           default:
               err("unknown flag");
               //fall through
           case 'h':
             printf("OpenAMIP(TM) modem simulator version %s\n"
                   "Usage: OpenAMIP_modem [options...] \n"
                   " [options...] are any combination of:\n"
                   "   -i <antenns_ip>   --ip address of antenna
controller.\n"
                   "   -p <port>         --tcp port nbr of antenna
controller\n"
                   "   -v                --verbose\n"
    "   -h                --print this message\n",
                   version
                  );
```

```
            return(FALSE);
          }
      }
  }
  return(TRUE);
}

/*
Main loop. get commandline params, open a TCP connection to the
antenna controller,
issue a satellite location sequence, ask for periodic status and
locations, and wait.
If the controller asks for periodic keepalive, then issue them.
After one minute,
shift to the next satellite and continue.
*/

int main(int argc, char **argv)
{
  int s=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);  //get a TCP
socket structure
  struct hostent *hp;
  struct sockaddr_in hostaddr_in;

  process_args(argc,argv);
  memset(&hostaddr_in,0,sizeof(struct sockaddr_in));
  hp = gethostbyname (antenna_ip);

  hostaddr_in.sin_family=AF_INET;
  hostaddr_in.sin_addr.s_addr=((struct in_addr *)(hp->h_addr))-
>s_addr;
  hostaddr_in.sin_port= htons(antenna_tcp_port);

  while(TRUE)
  { if( connect(s,(struct sockaddr *) &hostaddr_in,
sizeof(hostaddr_in)))
    { perror("TCP connect failed");
      exit(1);
    }
```

```
      handle_events(s);
      // close socket
      if(verbose)
        printf("==== Closing Socket\n");
      if(0!=close(s))
      { perror("Could not close socket\n");
        return 1;
      }
    }
    return(0);
}
```

iDirect
13861 Sunrise Valley Drive, Suite 300
Herndon, VA 20171-6126
+1 703.648.8000
+1 866.345.0983
www.idirect.net
Advancing a Connected World