SC Application Program Interface

Version 1.2

04 February 2003

ORBCOMM L. L. C.

21700 Atlantic Boulevard
Dulles, Virginia 20166, U.S.A.

Table of Contents

1. Introduction

To facilitate the development of application software using the ORBCOMM network, ORBCOMM has developed a collection of functions called the Subscriber Communicator Application Program Interface (SC API). These functions largely abstract the details of the ORBCOMM network and the SC Serial Interface Specification from the application developer. The SC API provides C language functions that fabricate the necessary byte streams from the SC Serial Interface Specification to be sent to the SC. In addition, the SC API simplifies the retrieval of serial packets sent to the application by the SC. The SC API allows the application developer to send Reports, Messages and Globalgrams; to access and specify the value of internal parameters; and to receive Messages, Globalgrams and User Commands sent to the SC. The SC API functions are organized into several source files in a modular fashion. The application developer need only include those SC API source files implementing the particular functionality required by the application.

2. Subscriber Communicator Application Program Interface (SC API)

2.1 Host Hardware Requirements

In order to host the SC API software, the target hardware must include:

- a hardware timer capable of generating periodic interrupts; and
- an interrupt-driven serial port.

2.2 API Architecture

Figure 1 on the next page depicts the architecture of the SC API. Communication with the SC is initiated when the application software makes a call to one of the API functions listed in Section 2.5 of this document. The API functions construct an API_PKT structure (defined in scapi.h) containing the appropriate command code and the application data, if any, then call psSendPktToSc(). This function re-formats the API_PKT structure into a packet from the SC Serial Interface Spec, calculates and installs the packet's Fletcher checksum, then calls vdXmitSerPkt() to transfer the packet to the SC via the serial port.

The SC will respond to packets sent by the API functions with either a PARAMETER RESPONSE packet (if the API function generated a GET or SET PARAMETER request), a STATUS packet (in response to a psAPIReqStatus() call), or a LINK LEVEL ACK packet (in response to all other packets from the API functions). These packets must be routed back to the psSendPktToSc() function so that the SC's response packet can be returned to the API function called by the application. The API function must analyze the packet from the SC in order to return to the application the appropriate result code and possibly a parameter value.

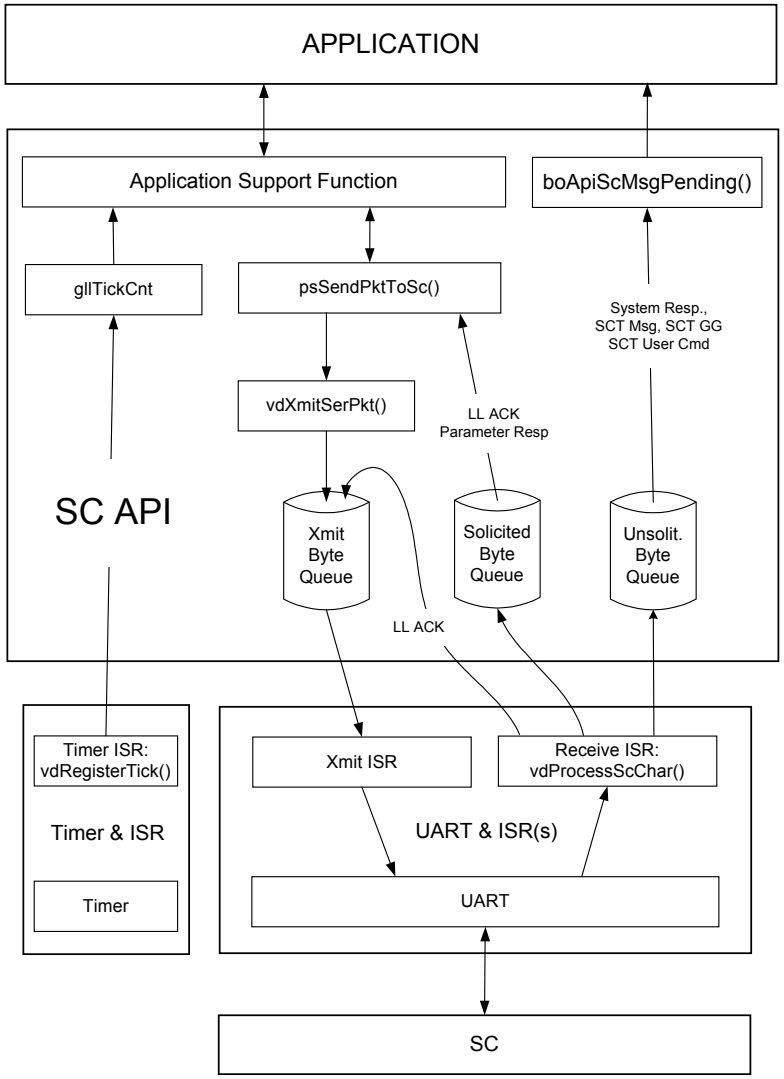

Figure 1 - SC API Architecture

In addition to packets that are generated in response to API function calls, the API must accept packets that arrive spontaneously from the ORBCOMM Message Switch (e.g., SYSTEM ANNOUNCEMENTS, SC-TERMINATED MESSAGES, etc.) or the Satellite (SC-TERMINATED GLOBALGRAMS). The inability to anticipate completely the arrival of data from the SC implies that the API's interface to the serial data stream from the SC must be interrupt-driven. Called from within the host's received character interrupt service routine, the vdProcessScChar() function analyzes the incoming packet's bytes in order to decide into which of the two SC Byte Queues to insert them, based on the packet type field byte. The vdProcessScChar() function also parses the incoming packets on the fly to determine their length. Once it has received the expected number of bytes in the packet, the vdProcessScChar() function increments a packet count for the appropriate SC Byte Queue to indicate the presence of a complete packet. Finally, the vdProcessScChar() function sends a LINK LEVEL ACK packet back to the SC in response to all packets received from the SC (except LINK LEVEL ACK packets themselves) as required by the SC's protocol.

Since they represent responses to API function calls, the vdProcessScChar() function routes bytes from PARAMETER RESPONSE, STATUS, or LINK LEVEL ACK packets into the SC Solicited Response Byte Queue. The psSendPktToSc() function will retrieve and analyze these packets to determine the appropriate return code for the API function. The SC Unsolicited Byte Queue is the destination for bytes from all other packets. The application software must call the SC API function boAPIScMsgPending() regularly to check for and to extract these packets.

In addition to the packet generation and processing functionality, the SC API implements a timeout mechanism to insure that it does not "hang" should communications with the SC be lost. This functionality is implemented using a software timer within the SC API driven by periodic interrupts ("ticks") from the host hardware. To allow the timer to be updated, the function vdRegisterTick() must be called from within the host hardware's periodic interrupt service routine. To insure low overhead, the vdRegisterTick() function merely increments an internal variable that is used by the API functions to track the SC's response.

2.3 Interfacing the SC API to Your Host Hardware

In order to use the SC API, the application developer must write the following software routines:

- an timer-driven interrupt service routine (ISR) that periodically calls the function vdRegisterTick();
- an ISR to extract characters from the XmitByteQueue and transfer them to the SC; and
- an ISR to receive serial characters from the SC. This ISR should pass the characters to the vdProcessScChar() function.

The included file dosbsp.c provides examples of these functions as required to run the SC API on a DOS platform.

2.4 Integrating the SC API with Your Code

1. Identify the SC API modules containing the functionality required by your application. Update your software development environment to compile them and link them with your application code. Generally these will include internal.c, global.c, util.c in addition to modules containing SC API functions called directly by your code.

2. In the file scapi.h, set the values of the constants SC_UNSOL_BYTE_Q_SIZE, SC_SOL_RESP_BYTE_Q_SIZE, and XMIT_BYTE_Q_SIZE based on the amount of memory available on your target hardware and on the size of the ORBCOMM SC-Originated and SC-Terminated Messages used in your application. These constants specify the size of the three queues within the SC API.

3. Make the following changes to your application code:

   - Call the boAPIInit() function during initialization.
   - Install the timer and serial port interrupts during initialization.

- Initialize the serial port hardware to 4800 baud, eight data bits, no parity bits, and one stop bit (or whatever setting is used by your SC's serial port).
- Frequently call boAPIScMsgPending() to check for Serial Specification packets sent by the SC. If you receive a packet from the SC, call the free() function to release the memory allocated for the packet's .pbyMsgBody element after you're done processing the packet.
- Add calls to SC API functions as required.

The included file scdemo.c shows these steps.

2.5 The Functions That Make Up the SC API

2.5.1 BOOL boAPIInit()

DESCRIPTION: Initializes the API; must be called during application software initialization.

PARAMETERS: none

RETURNS: BOOL - TRUE if the function successfully initialized the API, FALSE otherwise.

2.5.2 float flAPIGetVersion()

DESCRIPTION: Returns the API version.

PARAMETERS: none

RETURNS: API version (float)

2.5.3 void vdRegisterTick()

DESCRIPTION: Notifies the API of the expiration of the host's tick timer by incrementing the internal tick count variable. Called from within the target hardware's tick interrupt service routine. The API software uses the internal tick count variable as a timing reference for implementation of an SC response timeout.

PARAMETERS: none

RETURNS: none

2.5.4 void vdAPISetRepsonseTimeout(long lTimeoutTicks)

DESCRIPTION: Sets the timeout value, *in ticks*, for the API function calls. The duration of a tick is determined by the period of the interrupt that calls the

vdRegisterTick() function. To insure low overhead, the vdRegisterTick() function merely increments an internal variable that is used as a timing reference by the API functions to track the SC's response. The value of the internal tick count can be read by the application using the lAPIGetTickCnt() function described below.

PARAMETERS: long lTimeoutTicks - value, *in ticks*, to be used as a timeout for API function calls.

RETURNS: none

2.5.5 long lAPIGetTickCnt()

DESCRIPTION: Returns the value of the API's internal tick count variable. The internal tick count variable is reset to 0 when the microprocessor is reset (i.e., is powered up). The duration of a tick is determined by the period of the interrupt that calls the vdRegisterTick() function. The internal tick count variable is used as a timing reference by the API software to implement an SC response timeout. This function is useful for implementing timeouts.

PARAMETERS: none

RETURNS: current tick count (long)

2.5.6 PKT_STATUS psAPISetScParam(BYTE byParmIndex, BYTE byValueByteCnt, BYTE abyValue[])

NOTE: This function is useful only with SCs that are fully compliant with the Revision F SC Serial Interface Specification.

DESCRIPTION: Sets the SC internal configuration parameter specified by byParmIndex to the value specified by the abyValue[] bytes. See Appendix A in the SC Serial Interface Specification for a list and description of the configuration parameters. Scapi.h provides a list of equates useful for accessing these parameters. The abyValue bytes are interpreted as described below.

PARAMETERS: BYTE byParmIndex - index of parameter to be changed, see equates in scapi.h

BYTE byValueByteCnt - number of value bytes (abyValue[]) valid numbers are one through four

BYTE abyValue[] – one to four hexadecimal bytes that define the new value of the parameter specified by byParmIndex. abyValue[0] is the least significant byte. Each subsequent value byte is multiplied by increasing powers of 256:

parm value = abyValue[0] * 1 +

abyValue[1] * 256 +
abyValue[2] * 65536 +
abyValue[3] * 16777216

RETURNS: PKT_STATUS - indicates success or failure of the parameter value modification attempt (see scapi.h).

2.5.7 PKT_STATUS psAPIGetScParam(BYTE byParmIndex, long *plRetValue)

NOTE: This function is useful only with SCs that are fully compliant with the Revision F SC Serial Interface Specification.

DESCRIPTION: Returns the value of the SC's internal configuration parameter specified by byParmIndex in the variable pointed to by plRetValue. See Appendix A in the SC Serial Interface Specification for a list and description of the configuration parameters. Scapi.h provides a list of equates useful for accessing these parameters. NOTE: Several of the SC internal parameters cannot fit within a long variable. For access to these parameters use the psAPIGetScParamRaw() function. If this function is invoked to read an SC parameter that will not fit within a long, it will return the PKT_REJ_INV_PARM error code.

PARAMETERS: BYTE byParmIndex - index of parameter whose value is to be returned, see the SC Serial Interface Spec (Appendix A) and equates in scapi.h

long *plRetValue - pointer to a long variable into which the specified parameter's value will be loaded if retrieval is successful. If there is an error on the retrieval attempt, this parameter will not be modified.

RETURNS: PKT_STATUS - indicates success or failure of the parameter value retrieval attempt (see scapi.h).

2.5.8 PKT_STATUS psAPIGetScParamRaw(BYTE byParmIndex, BYTE **ppabyRetParms)

NOTE: This function is useful only with SCs that are fully compliant with the Revision F SC Serial Interface Specification.

DESCRIPTION: Returns an array of bytes from which the value of the SC's internal configuration parameter specified by byParmIndex can be calculated. See Appendix A in the SC Serial Interface Specification for a list and description

of the configuration parameters. Scapi.h provides a list of equates useful for accessing these parameters. While this function will work with any of the SC's internal parameters, it is most useful when accessing those parameters whose values will not fit into a long variable (e.g., downlink channel list). The first byte in the returned array indicates the number of subsequent parameter value bytes in the array (*ppabyRetParms[0] = byte 7 of the PARAMETER RESPONSE packet).

PARAMETERS: BYTE byParmIndex - index of parameter whose value bytes are to be returned, see the SC Serial Interface Spec (Appendix A) and equates in scapi.h

BYTE *ppabyRetValue - pointer to a variable that will point to an array of bytes from which the specified parameter's value may be calculated if retrieval is successful. If there is an error on the retrieval attempt, this parameter will not be modified.

RETURNS: PKT_STATUS - indicates success or failure of the parameter value retrieval attempt (see scapi.h).

2.5.9 PKT_STATUS psAPIInitiateSelfTests()

DESCRIPTION: This function instructs the SC to initiate internal health tests and set its internal *st_diag_code* parameter accordingly.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the self test request attempt (see scapi.h).

2.5.10 PKT_STATUS psAPIPerformLocalLoopbackTest()

DESCRIPTION: This function instructs the SC to execute a local loopback test and to set its internal *st_diag_code* parameter accordingly.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the loopback test request attempt (see scapi.h).

2.5.11 PKT_STATUS psAPIPerformGccLoopbackTest(BYTE byGwyId)

DESCRIPTION: This function instructs the SC to perform a loopback test through the designated Gateway and to set its internal *st_diag_code* parameter accordingly.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the loopback test request attempt (see scapi.h).

2.5.12 PKT_STATUS psAPIRequestOrbEle()

DESCRIPTION: This function directs the SC to send orbital elements data to the host as an SC-Terminated Message.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the orbital elements request attempt (see scapi.h).

2.5.13 PKT_STATUS psAPISendCfgCmd(CFG_CMD_DEF *pccCfg)

DESCRIPTION: This function sends a CONFIGURATION COMMAND to the SC to set the configuration parameters.

PARAMETERS: CFG_CMD_DEF *pccCfg – pointer to the CFG_CMD_DEF structure that is "filled in" by the application to specify the parameters desired to be set. These parameters include – the PIN code, the desired destination gateway, the default polled byte, default ack level, default report O/R, default message O/R, default priority, default message body type, default service type and the Gateway search mode.

RETURNS: PKT_STATUS - indicates success or failure of the attempt to send a Configuration command to the SC (see scapi.h).

2.5.14 PKT_STATUS psAPISendPosDetCmd(BYTE byTypeCode)

DESCRIPTION: This function sends a POSITION DETERMINATION command to the SC.

PARAMETERS: BYTE byTypeCode – the type code desired to be sent to the SC.

RETURNS: PKT_STATUS - indicates success or failure of the attempt to send a POSITION DETERMINATION command to the SC (see scapi.h).

2.5.15 PKT_STATUS psAPIGoToNextDownlink()

DESCRIPTION: This function directs the SC to search for a satellite on the next channel in the downlink channel list.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the go to next downlink channel request attempt (see scapi.h).

2.5.16 PKT_STATUS psAPIReqStatus()

DESCRIPTION: This function directs the SC to return a STATUS packet.

PARAMETERS: none

RETURNS: PKT_STATUS – indicates success or failure of the attempt to send the status request (see scapi.h).

2.5.17 PKT_STATUS psAPIRegisterWithGateway(BYTE byGwyId)

DESCRIPTION: Requests registration with the specified Gateway. The SC will ultimately respond with a SYSTEM RESPONSE packet providing the results of the registration request.

PARAMETERS: BYTE byGwyId - the ID of the Gateway from which registration is requested.

RETURNS: PKT_STATUS - indicates success or failure of the SC registration request packet transmission attempt (see scapi.h).

2.5.18 BOOL boAPIScMsgPending(API_PKT *papRetApiPkt)

DESCRIPTION: Indicates whether of not the API's Unsolicited Byte Queue contains a packet from the SC. If so, the function returns TRUE and loads the structure pointed to by papRetApiPkt with the packet bytes extracted from the queue.

PARAMETERS: API_PKT *papRetApiPkt - pointer to an API_PKT structure that will be loaded from the UNSOL_BYTE_Q if it contains at least one packet's worth of bytes.

RETURNS: BOOL - TRUE if the UNSOL_BYTE_Q contains at least one packet's worth of bytes, FALSE otherwise.

2.5.19 PKT_STATUS psAPIClrScoMsgQueue()

DESCRIPTION: The function instructs the SC to clear its SC-Originated message queue.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the clear queue packet transmission attempt (see scapi.h).

2.5.20 PKT_STATUS psAPIReqGlobalGrams()

DESCRIPTION: Execution of this function causes the SC to request the Satellite to transmit all stored GlobalGrams addressed to the SC.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the GlobalGram request attempt (see scapi.h).

2.5.21 PKT_STATUS psAPIReqOrInd(BYTE byGwyId)

DESCRIPTION: Execution of this function causes the SC to request the Satellite to transmit all O/R indicator addresses to the SC.

PARAMETERS: BYTE byGwyId - the ID of the Gateway from which the messages are to be requested.

RETURNS: PKT_STATUS - indicates success or failure of the OR indicator request packet transmission attempt (see scapi.h).

2.5.22 PKT_STATUS psAPIReqScoDataStatus(BOOL boIsMhaRefNum, BYTE byMsgRefNum, BYTE byGwyId)

DESCRIPTION: This function requests the status of the SC-Originated Message/Report/GlobalGram identified by byMsgRefNum from the Gateway specified by byGwyId. The SC will respond with a SYSTEM RESPONSE packet with the status field set appropriately.

PARAMETERS: BOOL boIsMhaRefNum - indicates whether the specified MsgRefNum is an MHA reference number (if TRUE) or a Gateway reference number (if FALSE).

BYTE byMsgRefNum - reference number of the SC-Originated message/report/GlobalGram whose status is requested.

BYTE byGwyId - if boIsMhaRefNum is FALSE, this parameter represents the desired Gateway from which the message status is requested. Otherwise this parameter is ignored.

RETURNS: PKT_STATUS - indicates success or failure of the message status request packet transmission attempt (see scapi.h).

2.5.23 PKT_STATUS psAPIReqDataFromGcc(BOOL bo150BytesOrLess, BYTE byGwyId)

DESCRIPTION: Requests queued SC-Terminated Messages/User Commands from the Satellite originating from the specified Gateway.

PARAMETERS: BOOL bo150BytesOrLess - indicates whether or not the request pertains only to messages/commands of 150 bytes or less.

BYTE byGwyId - the ID of the Gateway from which the messages are to be requested.

RETURNS: PKT_STATUS - indicates success or failure of the data request packet transmission attempt (see scapi.h).

2.5.24 PKT_STATUS psAPIReqSubjectList(BYTE byGwyId)

DESCRIPTION: Requests a list of subjects of Messages queued in the Satellite that originated from the specified Gateway.

PARAMETERS: BYTE byGwyId - the ID of the Gateway from which the subject list is to be requested.

RETURNS: PKT_STATUS - indicates success or failure of the subject list request packet transmission attempt (see scapi.h).

2.5.25 PKT_STATUS psAPIReqMsg(BYTE byMsgIndex, BYTE byGwyId)

DESCRIPTION: Referring to the list of message subjects retrieved by a previous call to psAPIReqSubjectList(), this function instructs the Gateway specified by byGwyId to transmit the message with the byMsgIndex'th subject.

PARAMETERS: BYTE byMsgIndex - index of message to be retrieved, from previous call to psAPIReqSubjectList().

BYTE byGwyId - the ID of the Gateway from which the message is requested.

RETURNS: PKT_STATUS - indicates the success or failure of the message request attempt (see scapi.h).

PKT_STATUS psAPIDeleteMsg(BYTE byMsgIndex, BYTE byGwyId)

DESCRIPTION: Instructs the Gateway identified by byGwyId to delete the message whose subject appeared as the byMsgIndex'th item in the list supplied by the most recent call to psAPIReqSubjectList().

PARAMETERS: BYTE byMsgIndex - index of message to be deleted, from previous call to psAPIReqSubjectList().

BYTE byGwyId - the ID of the Gateway from which the message is requested.

RETURNS: PKT_STATUS - indicates success or failure of the message deletion request attempt (see scapi.h).

2.5.26 PKT_STATUS psAPIClearActiveMsg()

DESCRIPTION: The SC should delete the SC-ORIGINATED or SC-TERMINATED MESSAGE currently being transferred between the SC and the Gateway.

PARAMETERS: none

RETURNS: PKT_STATUS - indicates success or failure of the message clearance attempt (see scapi.h).

2.5.27 PKT_STATUS psAPIDequeueMsg(BYTE byMhaRefNum)

DESCRIPTION: Instructs the SC to remove the message identified by the MHA reference number byMhaRefNum from its SC-Originated queue.

PARAMETERS: BYTE byMhaRefNum - MHA reference number of the message to be removed from the SC-Originated message queue.

RETURNS: PKT_STATUS - indicates success or failure of the message removal request attempt (see scapi.h).

2.5.28 PKT_STATUS psAPISendPosReport(BYTE pbyRetMHaRefNum, long lLatCode, long lLonCode)

DESCRIPTION: When executed, this function causes the SC to fabricate a POSITION REPORT packet based on the location data passed as parameters, and transmit this packet to the Gateway identified by the SC's *desired_gwy_id* internal parameter.

PARAMETERS: BYTE *pbyRetMhaRefNum - pointer to a BYTE variable into which an MHA reference number will be returned if the POSITION REPORT is sent to the SC correctly.

long lLatCode – a representation of the SC's latitude as returned by a POSITION STATUS packet.

long lLonCode – a representation of the SC's longitude as returned by a POSITION STATUS packet.

RETURNS: PKT_STATUS - indicates success or failure of the position report transmission attempt (see scapi.h).

2.5.29 PKT_STATUS psAPISendExplicitMsg(EXPLICIT_MSG_DEF *pmdMsgDef, BYTE *pabyUserData, BYTE *pbyRetMhaRefNum)

DESCRIPTION: Transfers pmdMsgDef->usMsgBodyLen bytes of application data pointed to by pbyUserData to the Satellite. This functions allows (requires) the message parameters to be specified explicitly (in the EXPLICIT_MSG_DEF structure passed as a parameter - see scapi.h).

PARAMETERS: EXPLICIT_MSG_DEF *pmdMsgDef - pointer to an EXPLICIT_MSG_DEF structure that is "filled-in" by the application to specify the message parameters. These parameters include: the ID of the Gateway to which the Message should be sent, whether or not the Message should be held in the SC until the SC receives a POLL command, the ACK level, priority, message body type, recipient quantity, and a parameter that indicates whether or not the Message contains a subject. Based on these parameters, (and the user data) the function generates an SC-ORIGINATED MESSAGE packet and transfers it to the SC.

BYTE *pbyUserData - pointer to the first byte of the application data whose length is specified by the pmdMsgDef->usMsgBodyLen parameter. The calling function is responsible for releasing this buffer after psAPISendExplicitMsg() returns, if required.

BYTE *pbyRetMhaRefNum - pointer to an MHA reference number that will be returned if the Message is sent to the SC correctly.

RETURNS: PKT_STATUS - indicates success or failure of the message transmission attempt (see scapi.h).

2.5.30 PKT_STATUS psAPISendExplicitGg(EXPLICIT_GG_DEF *pgdGgDef, BYTE *pbyUserData, BYTE *pbyRetMhaRefNum)

DESCRIPTION: Transfers pgdGgDef->usMsgBodyLen bytes of application data pointed to by pbyUserData to the Satellite as a GlobalGram. This functions allows (requires) the message parameters to be specified explicitly (in the EXPLICIT_GG_DEF structure passed as a parameter see scapi.h).

PARAMETERS: EXPLICIT_GG_DEF *pgdGgDef - pointer to an EXPLICIT_GG_DEF structure that is "filled-in" by the application to specify the message

parameters. These parameters include: the ID of the Gateway to which the Message should be sent and the O/R indicator. Based on these parameters, (and the user data) the function generates an SC-ORIGINATED GLOBALGRAM packet and transfers it to the SC.

BYTE *pbyUserData - pointer to the first byte of the application data whose length is specified by the pmdMsgDef->usMsgBodyLen parameter. The calling function is responsible for releasing this buffer after psAPISendExplicitGg() returns, if required.

BYTE *pbyRetMhaRefNum - pointer to an MHA reference number that will be returned if the Message is sent to the SC correctly.

RETURNS: PKT_STATUS - indicates success or failure of the message transmission attempt (see scapi.h).

2.5.31 PKT_STATUS psAPISendExpRep(EXPLICIT_REP_DEF *prdRepDef, BYTE *pbyUserData, BYTE *pbyRetMhaRefNum)

DESCRIPTION: Transfers exactly six bytes of application data pointed to by pbyUserData to the satellite as a Report. This function allows (requires) the report parameters to be specified explicitly (in the EXPLICIT_REP_DEF structure passed as a parameter - see scapi.h)

PARAMETERS: EXPLICIT_REP_DEF *prdRepDef – pointer to an EXPLICIT_REP_DEF structure that is "filled in" by the application to specify the report parameters. These parameters include: the ID of the Gateway to which the message should be sent, whether or not the report should be held in SC until the SC receives the POLL COMMAND, the service type and the O/R indicator.

BYTE *pbyUserData – pointer to the first of six bytes of application data. The calling function is responsible for releasing this buffer after psAPISendExplicitRep() returns, if required.

BYTE *pbyRetMhaRefNum – pointer to an MHA reference number that will be returned if the message is sent to the SC directly.

RETURNS: PKT_STATUS – indicates success or failure of the message transmission attempt (see scapi.h).

3. Frequently Asked Questions

3.1 How can I tell if a satellite is in view?

First, use the psAPIReqStatus() function to request a STATUS packet. Then call the boAPIScMsgPending() function until it returns a STATUS packet from the SC. Byte 8 of the STATUS packet indicates whether or not a Satellite is in view.

3.2 To which Gateway(s) is the satellite connected?

Use the psAPIReqStatus() function to request a STATUS packet. Call the boAPIScMsgPending() function until it returns a STATUS packet from the SC. Byte 9 of the STATUS packet indicates how many Gateways, if any, are connected to the Satellite. The bytes immediately after byte 9 indicate which Gateways are connected. See the STATUS packet description in the SC Serial Interface Specification.

3.3 How do I check to see if my message has been transmitted?

Use the psAPIReqStatus() function to request a STATUS packet. Call the boAPIScMsgPending() function until it returns a STATUS packet from the SC. If the *queued_ib_msgs* is zero, all of the SC-Originated Messages loaded by your application have been transmitted.

3.4 How can I determine the SC's location?

First call the psAPISendPosDetCmd() function, passing to it a *byTypeCode* parameter of 0 or 1 to insure the GPS hardware is running. Then call the psAPISendPosDetCmd() function again, this time with a *byTypeCode* value of 0 to request a POSITION STATUS packet from the SC. Call the boAPIScMsgPending() function until it returns a POSITION STATUS packet from the SC. The SC's location is specified by bytes 10-15 of the POSITION STATUS packet.

3.5 Do I have to generate a *LINK LEVEL ACK* packet in response to packets I received from the SC with the *boAPIScMsgPending()* function?

No, the SC API internal functions automatically generate a LINK LEVEL ACK packet in response to packets from the SC.

3.6 Do I need to test the checksum of packets received with the *boAPIScMsgPending()* function?

No, the boAPIScMsgPending() function returns only packets with valid Fletcher checksums. Invalid packets received from the SC are discarded.

3.7 Do packets received with the boAPIScMsgPending() function require any special handling?

Yes, you must call the free() function on the message body portion of the API packet (the .pbyMsgBody element of the API_PKT structure) to avoid a memory leak.

4. Software Naming Convention

Names of variables and functions within the SC API were derived using the so-called Hungarian naming convention. In this scheme, variables and functions names are given a prefix in lower case letters that indicates the type of the variable or the function's return code. The first letter of each segment of the main body of the variable name or function name is capitalized and the segments are concatenated together without the use of the underscore character. For example, iByteCounter would be a variable of type int and vdHwInit() would be a void function. Some programmers feel that the use of lower case characters in the prefix and mixed-case characters in the variable/function name results in C code that looks as if it were written in Hungarian, hence the name of this convention. The reduction in programming errors afforded by the ability to determine the type of the variable or function by inspection, however, makes the adoption of this naming convention worthwhile. To assist application developers in integrating the SC API functions, a dictionary of the variable/function return codes prefixes is presented on the next page.

Prefix	Variable/Function Return Code Type
a	array of (e.g., abyBuffer is an array of bytes)
ap	API_PKT structure as defined in scapi.h
bo	BOOL – either TRUE or FALSE, see scapi.h
bq	SC_BYTE_Q structure as defined in scapi.h
by	BYTE – unsigned char, defined in scapi.h
fl	float
gd	EXPLICIT_GG_DEF structure
rd	EXPLICIT_REP_DEF structure
cc	CFG_CMD_DEF structure
gl	a global variable
i	integer
l	long
md	EXPLICIT_MSG_DEF structure from scapi.h
p	pointer
ps	PKT_STATUS enum (defined in scapi.h)
qi	BYTE_Q_ID enum, defined in scapi.h
sp	SER_PKT structure as defined in scapi.h
sz	NULL-terminated string
us	UNSIGNED SHORT (defined in scapi.h)
vd	void
w	WORD (16-bits, an unsigned short)