SyncSockets Interface for ALC Protocol

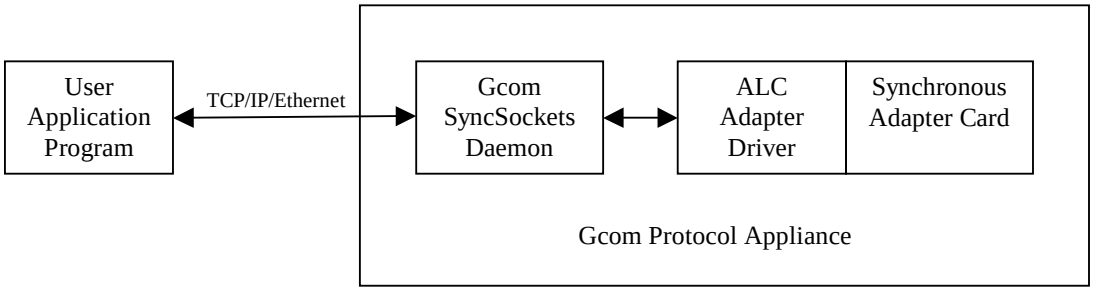
The purpose of this document is to specify the specific message formats to be used for passing ALC data to a user application via the Gcom SyncSockets interface. It is expected that the reader is already familiar with SyncSockets concepts. More information on SyncSockets can be found at the following locations.

SyncSockets Tutorial http://www.gcom.com/home/documents/syncsockets_tutorial/syncsockets_tutorial.pdf
SyncSockets User Guide http://www.gcom.com/home/documents/syncsockets/ssug.pdf

In addition we assume that the reader is familiar with the message formats and interchanges described in the Sita P.1024B specification. This is Sita document number PZ.7130.1 Third Edition Rev. 0.

Reference Architecture

The following diagram illustrates the software modules that are involved in this process.



The synchronous adapter card and its driver is responsible for transmitting and receiving the basic message blocks of the ALC protocol. No protocol interpretation is performed by the driver or the adapter.

The Gcom SyncSockets Daemon (SSD) provides the SyncSockets interface between the Adapter Driver and Adapter Card and the User Application Program. The SSD operates the adapter port using SyncSocket connections of type RAW. The data field content of the SS_OP_DATA message is passed as-is to the adapter driver, and vice versa. See below for more details concerning message formats.

The User Application Program (UAPP) manages multiple SyncSockets connections to the SSD, one per physical port. For the terminal mode of operation, it is likely that the User Application Program is, in turn, forwarding data to terminal emulators (TE) at remote locations.

The SyncSockets Interface

The SyncSockets interface between the UAPP and the SSD has the following characteristics.

1. Each SyncSockets connection corresponds with a single ALC physical port.
2. Traffic for multiple ALC terminals is multiplexed over a single SyncSocket connection.
3. Traffic for an individual terminal session is identified by way of the IA and TA characters within the messages. The demultiplexing of sessions is a function of the UAPP.
4. The configuration file that is processed by the SSD is created using the Gcom Management Console (GMC). This defines the RAW mode connection and specifies the physical port to which it is to be attached.
5. The data field content of the SS_OP_DATA message consists of the entire ALC message. The CCC character may optionally be included in the message data field.

Connection Management

By establishing a SyncSocket connection to the GPA for a particular physical port the UAPP causes modem signals to be asserted for that port (configurable) and the Adapter Driver to activate the adapter port. This places the adapter port in a state in which it is ready to send and receive ALC messages.

When the UAPP disconnects the SyncSocket connection, either by sending SS_OP_DISC_REQ or by closing the SyncSocket TCP connection, the configured modem signals are dropped and the adapter port is disabled.

Data Field Format in SS_OP_DATA

While operating in the data state the UAPP sends and receives SS_OP_DATA SyncSocket messages to and from the GPA in order to access the physical ALC port(s).

The detailed format of the data field of the SS_OP_DATA is as follows. The notation "[CCC]" means that the CCC is present or absent in the SS_OP_DATA message according to configuration options described later in this document.

1) Output messages
   a) GA IA EOMc [CCC]
   b) IA RS EOMc [CCC]
   c) IA TA CMD LA Text EOM [CCC]
   d) IA TA C1 C2 EOMc [CCC]
2) Input messages
   a) GA NIA
   b) IA TA Text EOM [CCC]

c) IA TA Text EOM [CCC] GA NIA
d) IA TA Text EOM [CCC] IA TA Text EOM [CCC] … GA NIA

The characters in the data field consist of 6-bit BCD right justified within each 8-bit byte with the two high order bits of each byte set to zero. The adapter driver calculates the CCC value when transmitting messages. Any CCC value included in the SS_OP_DATA from the UAPP is ignored and therefore need not be the correct value. If the CCC is present in SS_OP_DATA sent to the UAPP it is the one received on the comm line.

Received messages with a CCC that does not check are sent to the UAPP as SS_OP_SPECIAL with the SyncSockets var field set to the value of SS_SOP_BAD_DATA.

Terminal Mode and Host Mode

When operating in terminal mode the GPA will receive Output Messages and send Input Messages. When operating in host mode it will receive Input Messages and send Output Messages.

Output Messages include polling messages formatted as in 1.a above. The UAPP must decide how to respond to received polling messages when operating in terminal mode. It must decide when to issue polling messages when operating in host mode.

Input Messages include text message responses (2.b) and a special message that tells the host to proceed with its polling cycle (2.a). Input messages can be concatenated together in the same SS_OP_DATA that is sent to the GPA. This is illustrated by message forms 2.c and 2.d.

When the GPA receives Input Messages (operating in host mode) it never concatenates multiple messages into the data field of an SS_OP_DATA message. Thus Input Messages that the UAPP receives from the GPA will always be of the form 2.a or 2.b and never of the form 2.c or 2.d.

Terminal Mapping

Relative to a particular physical ALC port an individual terminal and/or attached device is identified by the IA and TA fields of the messages exchanged with the host via that port. These characters are preserved in the exchange between the UAPP and the GPA.

It is beyond the scope of this document to speculate as to how the UAPP will map these characters into discrete Terminal Emulator sessions or how the UAPP will manage such mappings among multiple TE sessions and multiple physical UTS ports.

Configuration Using the GMC

You can use the Gcom Management Console to configure the ALC protocol. If you download and install the test program archive, alctest.tar, then within that archive you will find a file named alctest/configs/alc-config.tar. You need to copy this file to some location that is accessible via the browser that you use to connect to the GMC.

Use the GMC's upload feature under Management → Archives to upload the alc-config.tar file into the GPA, then install it via the GMC. At that point there should be visible protocol stack names for ALC-BTOB-1 and ALC-BTOB-2. Under the Management → Adapters function assign these to ports 1 and 2 of the adapter.

There will also be an SSD configuration named SSD_Raw. Under the Management → SSDs function assign this configuration file to the SSD.

You can then start all ports to cause the GPA to run these protocol configurations. At that point you are ready to run the alctest program.

ALC SyncSocket Configuration

This section is oriented towards using ASCII configuration files to configure ALC ports. If you are using the GMC do not follow this procedure.

The configuration of the ALC ports in the SSD configuration file proceeds in three steps.

First there needs to be a module definition for the RAW mode module itself. This configuration element resides in the section of the file in which the modules are defined. The number *n* is a small integer that is used to number the module definitions consecutively.

```
module.n
        name            = "libssd_rf.so"
        id              = "RAW"
```

Second there need to be connection definitions that give the connection names to be used. The following example is for two ALC connections that are to be individually configured in the third step of the process. Again, the numbers *x* and *y* are used to number the connection definitions consecutively.

```
connection.x:
        type            = "RAW"
        name            = "line_1"

connection.y:
        type            = "RAW"
        name            = "line_2"
```

Third, there need to be specific definitions for the named connections. The following illustrates this with examples.

```
line_1:
        logopts         = 0
        logopts         = 0x7d
        logopts         = 0xff
        logopts         = 0x01          # write to log file
        logfile         = "/usr/spool/gcom/line_1.log"
        logsize         = 8000
        bad_data_option = 1             # get frames w/CRC errors
        line_number     = 1
        max_message_size= 5000

line_2:
        logopts         = 0
        logopts         = 0x7d
        logopts         = 0xff
        logopts         = 0x01          # write to log file
        logfile         = "/usr/spool/gcom/line_2.log"
        logsize         = 8000
        bad_data_option = 1             # get frames w/CRC errors
        line_number     = 2
        max_message_size= 5000
```

The individual parameters are shown in the following table.

Parameter Name	Type	Description	
line_number	Number	PPA parameter used in the CDI Attach Request This value represents the physical line number over which the Raw Frame connection operates. It begins with 1.	
max_message_size	Number	Maximum message size in bytes	
logfile	String	Name of the logfile to use	
logsize	Number	Maximum size of the logfile in kilobytes	
logopts	Number	Logging options, usually specified in hexadecimal as the logical OR of the following values. The logging options are applied to each connection individually. The value specified for the first connection is also used as the global default value. **Note**: If this parameter is not set, the value defaults to 0, which means no error reporting at all. See the Gcom CDI API Guide for more detail.	
		0x0001	Write log entries to log file.
		0x0004	Log received protocol messages.
		0x0008	Log transmitted protocol messages.
		0x0010	Log UNIX error returns.
		0x0020	Log signal handling.
		0x0040	Log received data.
		0x0080	Log transmitted data.
		0x0100	Log discarded messages.
		0x0200	Verbose logging

encryption_mode	Number	Value	Description
		0 (default)	No encryption
		2	DES CBC (Cipher Block Chaining)
		3	DES CFB (Cipher Feedback)
		4	DES OFB (Output Feedback)
encryption_key		A quoted string containing one, two, or three encryption passwords Each password is separated by an ASCII ':'. If a password must contain a ':', it is represented as "\:" in the string. Default is "". Example: "password 1:second password" The number of passwords determines the number of iterations of DES encryption applied to the data stream. For example: Three passwords result in triple-DES encryption.	

For the encryption parameters note that the encryption applies to the SyncSocket connection between the SSD and the UAPP.

ALC Driver Configuration

This section is oriented towards using ASCII configuration files to configure ALC ports. If you are using the GMC do not follow this procedure.

The synchronous driver in the GPA is configured via the GMC to perform the 6-bit synchronous protocol of ALC. There are no protocol modules built on top of the driver, thus the designation "Raw Mode." This configuration is illustrated in the following ASCII configuration file excerpt.

```
cd_min_sdu      = 0
cd_max_sdu      = 5000
cd_output_style = 0             # no data acks
ix25mode        = 14            # ALC
maxfrmsz        = 2100
maxfrmsz        = 460
ix25_chip_type  = 13            # HD64570
ix25wrapflg     = 00            # no loopback
ix25_clks       = 2             # TxC output, RxC from TxC
ix25_clks       = 00            # TxC, RxC both inputs
ix25_clks       = 1             # TxC output, RxC input
seq_end_dly     = 8             # nr of pad chars
seq_end_dly     = 1             # nr of pad chars
mdm_msk         = 0x03          # DTR and RTS
baudrate        = 9600
wd_timer        = 5000          # watchdog
lbfrcqsz        = 20
```

```
frxcqsz          = 40
ix25_ll_optns    = 0x08          # large trace buffer
ix25_ll_optns    = 0x18          # bit invert, large trace buffer
ix25_ll_optns    = 0x28          # include CCC, large trace buffer
stop             = 1
```

The following table summarizes a few of the key parameters.

Parameter	Description	
`cd_output_style`	The value 0 inhibits acknowledgments from the driver for efficiency reasons.	
`ix25mode`	The value 14 designates ALC protocol.	
`ix25_clks`	The value 1 is appropriate for back to back testing. Change to 0 to connect to an external mode.	
`seq_end_dly`	The number of idle PAD characters sent at the end of a message.	
`ix25_ll_optns`	Bitwise OR of individual options as follows.	
	0x08	Use a large trace buffer for characters sent and received. Always set this to 0x08.
	0x10	When set, invert the bits of the data prior to transmitting and after receiving.
	0x20	When set the data field of the SS_OP_DATA will include the CCC character.
`stop`	Set to 1 to prevent any attempt to add additional protocol stacks on top of the driver.	

ALC Test Program

Upon request to Gcom you may obtain a copy of a program that can be used to test ALC ports in both host and terminal mode. The program demonstrates the capability to send and receive all forms of ALC messages.

The program is released in source code form in a TAR archive. The name of the archive is `alctest.tar`. When the file is untarred it creates a subdirectory named “alctest”. The contents of the archive are as follows.

File Name	Description
alctest/alctest.c	Source for the ALC test program.
alctest/Makefile	Makefile to build the program alctest. The program is built in the alctest directory.
alctest/mk_tar	Shell script to build this tar archive.
alctest/configs/alc-config.tar	An archive file that can be uploaded and installed via the Gcom Management Console. It contains two protocol stacks: ALC-BTOB-1 (for port 1) and ALC-BTOB-2 (for port 2).

	It also contains a configuration file for Gcom_ssd named SSD_Raw.
alctest/configs/configs.btob	Driver level configuration for ALC. Must be modified to change the "include CCC" option if so desired.
alctest/configs/ssd.cfg	SyncSockets Daemon configuration file. Contains connection definitions for two raw mode connections.
alctest/configs/die	Shell script to kill Gcom_monitor and Gcom_ssd. Must be run on the GPA.
alctest/configs/start.btob	Shell script to start Gcom_monitor and Gcom_ssd with the above configuration files. Must be run on the GPA.

The program can be run from any computer that has network access to a GPA 2G running ALC protocol. The configuration of the ports has ports 1 and 2 configured for ALC. For testing purposes the two ports must be cabled back to back using the standard Gcom back to back (orange) cable.

The program is run using the following command line.

`alctest –h` *hostname* `[-C]`

The optional –C argument tells the test program that it is to insert a dummy CCC character into the test data that is sent to the GPA and to expect there to be a CCC character present in data received from the GPA. In order for this to function correctly the setting of this option must be coordinated with the ix25_ll_optns 0x20 bit (see above).

The output of the program looks like the following. At the end of the test you must type a Return to cause the program to exit. This allows you to check statistics and trace information while the test program keeps the ports open.

```
# ./alctest –h debugtst1
ALC Test version 1.2 08/05/06
Opening connection to debugtst1:8000
Opening connection to debugtst1:8000
Attempting SS connection to debugtst1:line_1
Connected to debugtst1:line_1
Attempting SS connection to debugtst1:line_2
Connected to debugtst1:line_2

Start test
Write message 3 bytes
 Read message 3 bytes
Write message 3 bytes
 Read message 3 bytes
Write message 8 bytes
 Read message 8 bytes
Write message 5 bytes
```

```
 Read message 5 bytes
Write message 2 bytes
 Read message 2 bytes
Write message 6 bytes
 Read message 6 bytes
Write message 6 bytes
 Read message 6 bytes
Write message 6 bytes
 Read message 6 bytes
Write message 6 bytes
 Read message 6 bytes
Write message 8 bytes
 Read message 6 bytes
 Read message 2 bytes
Write message 14 bytes
 Read message 6 bytes
 Read message 6 bytes
 Read message 2 bytes
Hit return to exit
```