CGC Software Manual BI-0501
CAN Gateway Controller

Version 2.1 January 1998

Documentation History

Date	Version	Change/Description
97/07/10	0.1	Preliminary Specification
97/09/19	1.0	First Release
97/12/15	2.0	Protocol Changes
98/01/05	2.1	Minor Clarifications

Copyright

Disclaimer

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Purpose

This manual describes the usage and technical specification of the CAN Gateway Controller protocol (abbreviated CGC). Both the setup and basic usage procedures as well as a specification of the protocol is described in this manual. To ease the application of the CGC protocol on a host, a special library is provided which hides most of the details of the protocol.

The CAN Gateway Controller protocol allows users connected to a BI-0501 through a TCP/IP style network to communicate with devices connected to any of the BI-0501's CAN ports. It is a simple request/response type protocol which maintains very little state between requests. Although the CGC server for example does require a user to login before giving access to CAN ports, it is not possible to have multiple outstanding requests which need to be replied to in a particular order. With the CGC protocol users can run CAN based applications on a host computer which communicate with devices connected to any of the CAN ports on a BI-0501.

1.2 Audience

This manual is mainly intended for those who are involved with the installation and use of the BI-0501 CGC server.

1.3 Conventions

To distinguish between various types of information this manual uses a few typographical conventions:

Helvetica: The Helvetica font is used for normal body text;
`Courier`: The courier font is used to denote literal values as in:
'the value of the error code will be b'

Meta-syntactic items such as fields in packets are denoted by enclosing them in < and > brackets.

1.4 Overview

Chapter 1 Introduction
This chapter.

Chapter 2 CAN Gateway Controller
An overview of the CAN Gateway Controller function and usage is given in this chapter. It gives an overview of the CGC protocol and describes the basic installation steps required

to install the CGC in a TCP/IP based network. The chapter concludes with a walk through of the usage of both the CGC protocol and the host library.

Chapter 3 CAN Gateway Controller Protocol
This chapter gives the specification of the CAN Gateway Controller protocol as it appears “on the wire”.

Chapter 4 CGC Host Library
To ease the implementation of the CAN Gateway Controller protocol on the host side a library is available which does most of the packet building and parsing. This removes the need for an application programmer to know the details of each packet flowing between the host and the BI-0501.

Chapter 2

CAN Gateway Controller

2.1 Introduction

In this chapter an overview of the CAN Gateway Controller protocol is given, followed by the installation procedure of a BI-0501 equipped with the CGC protocol.

2.2 Overview

The CAN Gateway Controller is both a protocol definition and an standard application on the BI-0501. The protocol describes the sequence and format of messages between a user's host (the client) and the CGC application (or CGC server) on the BI-0501. The protocol operates on top of TCP/IP (specifically TCP) allowing hosts (the clients) to send and receive CAN style messages to and from devices connected to any of the BI-0501's CAN ports.

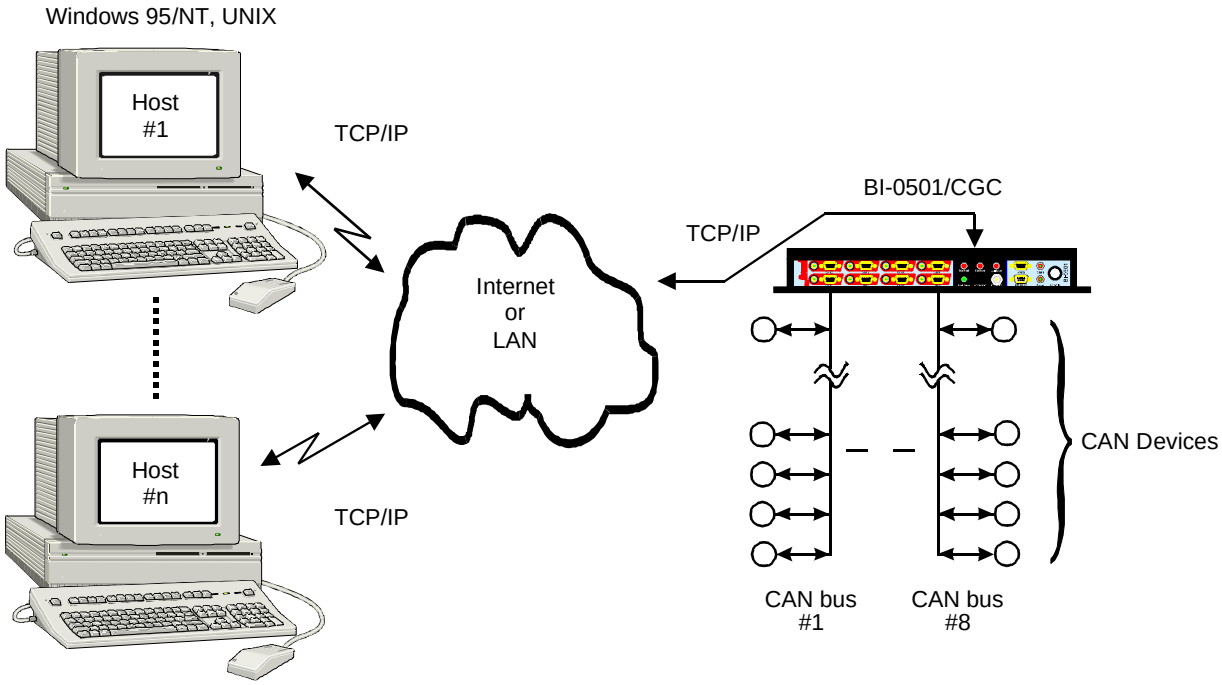


Figure 2-1 BI-0501/CGC Network Configuration

Two TCP connections are maintained by the CGC protocol: one for the command protocol and one for relaying CAN messages between the BI-0501 and the user's host application. The command protocol is a simple request/response protocol. The user sends a single command to the CGC server on the BI-0501

which is executed by the CGC server and replied on. It is never possible to have multiple outstanding requests and responses are never sent out of order or delayed. CAN messages are relayed on a separate TCP connection between the BI-0501 and the host application. This separate TCP connection is initiated by the CGC server after executing the USER-LOGIN command. As part of the response, the CGC server sends back a port number to which the user's host application can connect in order to send and receive CAN messages. Before it is possible to send and receive CAN messages over this separate connection it is necessary for the host application to allocate CAN ports on the BI-0501.

The CGC protocol supports multiple users. Before access is granted to one or more CAN ports, a user must request allocation of those ports. This allows users to access a mix of CAN ports on the BI-0501 and also prevents users interfering with each other.

2.3 Setup

Very little is required to setup the BI-0501 with the CGC server. The CGC server is contained in Flash memory and will start-up when power is applied to the BI-0501. It will then listen to connection requests from either it's Ethernet port or serial port. The serial port uses the SLIP protocol.

The factory default IP address for the BI-0501's Ethernet port is 10.0.0.1 and that for the serial port is 10.0.0.2. These addresses are stored in nonvolatile memory and at initial power-up the CGC server will listen for connection requests to these addresses. The addresses should probably be changed to match the network addresses where the BI-0501 is installed. The new addresses can be configured using the SYS-CONFIG packet which also changes the addresses in nonvolatile memory.

To do so one should login to the CGC server using the USER-LOGIN packet (or the `cgc_u_login()` function in the CGC Host library) with name 'root' (see below) and subsequently issue a SYS-CONFIG command. The CGC server will then store the new IP addresses in nonvolatile memory and reset itself. After the CGC server has reset itself it will listen to connection requests using the newly configured IP addresses.

The TCP port on which the CGC server on the BI-0501 listens for connection requests is 28001. This cannot be changed.

The default settings for the serial port are: 9600bps, 8 data bits, no parity, 1 stop bit.

For certain operations on the BI-0501 special privileges are needed. These operations include changing the IP addresses for the Ethernet and serial ports and forcibly resetting the CGC server. To gain these privileges one must be logged in under a certain name. The factory default name is `root' and is stored in nonvolatile memory, but can be changed with the SYS-CONFIG packet in the same manner as described above for changing the BI-0501's IP addresses. The new name is written to nonvolatile memory and will take effect after the next reset of the CGC server.

The default bit rate on a CAN port on the BI-0501 after allocation is 125kbps. The bit rate can be changed with the PORT-SETSPEED request. The following bit rates can be selected:

Table 2-1 CAN bus bit rate and bus length

Bit Rate (bps)	Bus Length (m)
1M	25
800k	50
500k	100
250k	250
125k	500
50k	1000
20k	2500
10k	5000

Note that the bus length are recommended maxima. For bus lengths greater than 1000m it may be necessary to employ repeater devices.

2.4 Usage

To use the CGC server one can either hand craft packets and send them as described in chapter 3 or use the CGC host library functions described in the chapter 4.

The protocol is mostly stateless meaning that the transmission of a packet does not depend on a particular state of the CGC server. However, for certain types of packets to perform a useful function it is necessary that certain other packets have been sent previously. For example: before a port can be allocated by a user, the user must have logged in using the USER-LOGIN packet.

The normal sequence of events when using a CAN port on the BI-0501 is:

- login with the USER-LOGIN packet;
- connect to the port number on the BI-0501 mentioned in the response to the USER-LOGIN packet;
- allocate one or more ports with the PORT-ADD packet;
- possibly configure watchdog messages and filters with the WATCHDOG-ADD and FILTER-ADD packets;
- send and receive CAN messages;
- logout with the USER-LOGOUT packet when done.

Control packets can be sent regardless if the sender is logged in with the CGC server. They are always responded to unless the sender does not have sufficient privileges.

2.5 Security

The CGC server uses a cooperative security model. Before given access to CAN ports on the BI-0501 users must register themselves with the CGC server. To change certain essential parameters on the CGC server one must be registered with the CGC server as the so called `root' user. This name can be changed on the CGC server to avoid others from inadvertently changing those parameters.

There are many different types of security measures that could be taken to protect the CGC server and the BI-0501 from malicious tampering in a production environment. The exact nature of those measures depend strongly on the facilities available on the (LAN) network to which the BI-0501 is connected. Advanced security measures could include such things as firewalls, packet encryption and strong authentication techniques.

Chapter 3

Protocol Description

3.1 Introduction

In this chapter the CAN Gateway Controller protocol is described in detail. The CGC protocol allows one or more host computers (typically workstations) to interact with any of the CAN channels on the BI-0501.

3.2 Connection Management

The CGC server on the BI0501 uses up to three TCP/IP connections with the user's host application: one is used for the command/response protocol, one is used for exchanging CAN messages and one is used to notify the user's host application in the event of a power failure on the BI-0501. The first two connections are TCP connections, while the latter is a UDP connection.

The CGC server listens on TCP port 28001 for connection requests. When the user's host application connects to this port an initial TCP connection for the command/response protocol is created. This connection is also called the "command connection".

The user's host application can then send commands to the CGC server which are executed and replied to. To send and receive CAN messages however, the user's host application must log in with the CGC server, using the USER-LOGIN packet. A side effect of this USER-LOGIN packet is that the CGC server allocates a second TCP port on which it will listen for a connection request from the user's host application.

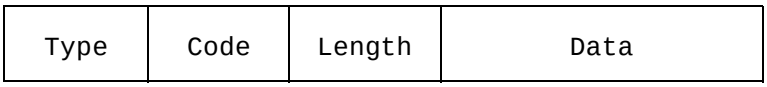
This second TCP port is used for relaying CAN messages between the CGC server and the user's host application. The TCP port number for this second connection is sent to the user's host application as part of the reply to the USER-LOGIN packet. After the user's host application connects to this second port, no CAN messages will be sent by the CGC server until CAN ports are actually allocated using the PORT-ADD request. This second TCP connection for relaying CAN messages is also called the "data connection".

When the user's host application sends a USER-LOGOUT packet the CGC server closes the data connection, but not the command connection. If the user's host application closes the command connection, the CGC server performs all internal actions associated with the USER-LOGOUT, which includes closing its end of the command connection and the data connection.

The third connection between the user's host application and the CGC server is used to notify the user's host application of unexpected power failures of the BI-0501. This connection uses the UDP protocol and does not need to exist permanently (UDP is a so called connectionless protocol). The user's host application notifies the CGC server of a UDP port on the host on which it wants to receive a power failure notification with the SYS-POWERDOWN packet.

3.3 Packet Format

Packets are basically encoded in a fixed field format, i.e. in a packet of a particular type the same field is always located at the same position in the packet. The first two fields in every packet are a Type field and a Code field. These two indicate the type (request or response) and the operation code of the packet. Below is a representation of a packet:



The `Type` field indicates whether the packet is a request to perform an operation or a response to a previous request. There are three types of packets:

- request Request packets initiate an action by the receiver;
- solicited Response packets that are sent (as the name implies) in response to request packets;
- unsolicited Unsolicited response packets are sent without a previous request.

The difference between the first two types is that request packets must be answered with a solicited response packet. Request packets may be sent by either a host or by the CGC. Request packets can also be interpreted as commands, while solicited response packets give the result of command execution. Currently the CGC does not send request packets.

Unsolicited response packets are sent without a previous explicit request. They can be sent when an event happens or a state changes on either the host or the CGC. Currently unsolicited response packets are sent by the host when it wants the CGC to transmit one or more CAN messages and by the CGC when it receives messages from the CAN ports on the BI-0501.

The `Code` field indicates the specific request or response depending whether the `Type` field indicates a request or response packet.

The `Data` field is of variable length and depends on the `Code` and `Length` field.

The basic unit of packets is a byte. The `Length` field defines the total number of bytes specified in the packet. The `Length` field itself is 4 bytes long and defined as an unsigned integer in network byte order. The `Type` and `Code` fields are each one byte long from which follows that the minimum packet length is six bytes. In the packet descriptions in chapter 3 the `Length` field is mentioned but not further described as it has the same format and function for each packet.

3.4 Field Types

As mentioned in the previous paragraph, packets use a fixed position for fields. This means that the position of a field in a packet defines its type as opposed to tag based field encoding schemes. In this paragraph the definition of the various field types and their encoding used in the packet descriptions in the following paragraphs.

Three basic data types are used:

`u_char`	an unsigned 8 bit integer;
`u_short`	an unsigned 16 bit integer;
`u_long`	an unsigned 32 bit integer.

For the u_short and u_long types the bytes within a value are encoded in so called *network byte order*. This means that the least significant byte is at the highest index position if the value is viewed as a sequence of u_char values.

Most systems implementing TCP/IP have special functions or macros to convert values of u_short or u_long type to and from their native format. They are usually known under the names htonl() (which stands for *h*ost *to n*etwork *l*ong), htons(), ntohl() and ntohs(). The htonl() and ntohl() convert u_long values from and to host byte order and htons() and ntohs() convert u_short values.

When an sequence of values is contained in a field, an array notation is used:

```
u_char[8]
```

means a sequence of 8 unsigned integers (usually characters).

Strings are written as an array of u_char values. All strings have a maximum length and strings shorter that the maximum array length should be padded with NULL characters (i.e. \0).

Fields within a packet are written using angle-brackets (< and >) as delimiters, as in:

```
<port> <errcode>
```

meaning that the field named port is followed by a field name errcode. Note that the whitespace between the <port> and <errcode> fields is for typographical reasons only. Within a packet fields are packed.

In packet descriptions such as:

```
R N <length> <timestamp0>
```

the letters R and N are the Type and Code field and are in ASCII code. In fact all letters are written as literal ASCII characters. The <length> field is defined as an u_long size field which holds the length of the complete packet including the Type and Code fields and the <length> field itself. Therefore the <length> field should at least have the value 6.

Not all packets have a fixed number of fields, sometimes a list of items may be given in a request or returned in a response. When a packet has a variable number of fields, the notation:

```
<nrofitems> { <item> }
```

is used. The <nrofitems> field is the count of the number of <item> fields that follow. The braces ({ and }) means that zero or more <item> fields may follow the <nrofitems> field. Note that the field <nrofitems> may have a value of zero, in which case no <item> fields follow it.

A CAN message is encoded as u_char[20] in the packet descriptions. The actual layout is:

```
u_long       <timestamp>
u_long       <ident>
u_short      <reserved>
u_char       <rtr>
u_char       <dlc>
u_char[8]    <data>
```

The <timestamp> field is the time when the message was received on a CAN port by the BI-0501 counted in microseconds since the CGC server was started. For CAN messages to be sent by the BI-0501 the <timestamp> field is ignored. The <ident> field is the identifier of the CAN message. The <rtr> field indicates if the RTR bit is set in the message. It has a non-zero value in that case. The <dlc> field is the CAN data length code and finally the <data> field contains the data of the CAN message.

No checking is done on CAN message to be sent on a CAN port, the <dlc> field is assumed to have a value between 0 and 8 for example . It is the responsibility of the user's application that syntactically correct CAN messages are generated.

3.5 Request Types

This paragraph lists each request packet with its associated request code. It is intended for reference purposes only. Requests are indicated with a printable ASCII letter.

A	PORT-ADD
B	PORT-SETSPEED
C	CAN-SEND
D	PORT-DELETE
E	CAN-ERROR
F	FILTER-STATUS
H	WATCHDOG-ADD
I	FILTER-ADD
J	SYS-CONFIGURE
K	SYS-SERIAL
L	USER-LOGIN
M	PORT-GETSPEED
N	SYS-NOP
O	USER-LOGOUT
P	PORT-STATUS
Q	FILTER-DELETE
R	SYS-RESET
S	PORT-STATISTICS
U	USER-STATUS
V	SYS-INFO
W	WATCHDOG-STATUS
X	WATCHDOG-DELETE
Y	CAN-RECEIVE
Z	SYS-ERROR
!	SYS-POWERDOWN

3.6 Privileges

For certain operations the user needs special privileges, the so called root privileges. The factory default configuration assigns these privileges to the user who logs in with the name 'root'. The SYS-CONFIG packet can be used to change this name. In the packet descriptions however it is assumed that the name has not been changed from the factory default 'root'. If the name has been changed then the correct name should be read instead.

3.7 Error Codes

When a request results in an error, an error code is returned. An error code is a single printable ASCII character with no further special meaning. The following error codes are defined:

a	No error; operation completed successfully.
b	Malformed packet.
c	Not owner; attempt to perform an operation for which the user does not have sufficient privileges.
d	No such user; user is not known to the CGC.
e	User list is full.
f	User already logged in.
g	No such port; requested CAN port does not exist.
h	Port never allocated; request for an operation on a CAN port that was never allocated by the user.
j	No such watchdog message; requested watchdog message does not exist.
k	Invalid watchdog condition.
l	Too many watchdog messages; watchdog message list is full.
m	No such filter; requested filter message does not exist.
n	Invalid filter condition.
o	Too many filters; filter list is full.
p	CAN controller overflow.
r	CAN controller is in bus-off state.

3.7.1 ERROR: Malformed packet detected

Request

N/A

Response

```
S Z <length> <errcode> { <packetcontents> }
```

Fields

```
u_long          <length>
u_char          <errcode>
u_char[32]      <packetcontents>
```

Description

If a packet is received which cannot be interpreted an ERROR packet is returned. This error packet always has the `<errcode>` field to b and copies the first 32 bytes from the original packet into the `<packetcontents>` field. If the original packet is less than 32 bytes long then the remaining locations in `<packetcontents>` are filled with hexadecimal 0.

3.8 Power-down Handling

When the BI-0501 experiences a power failure it notifies the CGC server which in turn attempts to notify the currently connected users. This notification is done by sending a special message using the UDP protocol. Simultaneously the CGC server will send a single predefined CAN message out over the CAN ports allocated to the user.

The UDP port to which the power down notification is sent must have been previously registered with the CGC server. Likewise the CAN messages to be sent on the CAN ports allocated to user must have been defined previously. There are no defaults.

The name of this packet is: SYS-POWERDOWN.

Request

```
R ! <length> <udpport> <nrofports> { <port> <message> }
```

Response

```
S ! <length> <err0> <nrofports> { <port> <err1> }
U ! <length> <code>
```

Fields

```
u_long          <length>
u_short         <udpport>
u_long          <nrofports>
u_long          <port>
u_char[20]      <message>
u_char          <err0>
u_char          <err1>
u_long          <code>
```

Description

The SYS-POWERDOWN command defines the UDP port number on the user's host to which a power down message should be sent in case the CGC server detects an unexpected power failure.

The <udpport> is the port number (in network byte order) to which the power down message should be sent. Only ports in the range 1024 to 65535 are accepted, UDP port numbers outside this range are rejected with an error notification. If the UDP port number is rejected because of a range error, no CAN messages will be registered.

The <nrofports> field defines the number of ports for which to set the power down CAN message. It is followed by a (possibly empty) list of CAN ports on which to set the message. The <port> field defines the CAN port and the <message> defines the CAN message.

In the response packet the <err0> field indicates if the UDP port number was accepted or not. If it was accepted the <err0> field will be set to a. It is followed by the same <nrofports> field from the request packet and enumerates for each port if the CAN message has been accepted or not.

When a power failure is detected by the CGC server it sends a unsolicited power failure message to the previously mentioned UDP port. The <code> field is an unsigned long integer in network byte order and will have the value 1.

3.9 User Management

Before any of the CAN ports available through the CGC can be used, a user must identify him or herself with the CGC. This is done with a USER-LOGIN packet. After login the user can allocate any number of CAN ports and send and receive messages through them. When the user is done, sending the USER-LOGOUT packet will automatically release all CGC resources allocated to that user. No user can login more than once. Any user can request a list of users through the USER-STATUS command.

A user is identified by a name and this name must be unique within the CGC although it does not have a predefined list of possible user names. One user name is special however: the name ‘root’. A user who is logged in with this name has special privileges in that the ‘root’ user can forcibly log out any other user. The user ‘root’ can also remove CAN ports allocated to other users to make them available to other users.

User names are at most eight characters long and may contain any printable ASCII character.

3.9.1 USER-LOGIN: Login a user to the CGC

Request

```
R L <length> <user>
```

Response

```
S L <length> <errcode> <dataport>
```

Fields

```
u_long          <length>
u_char[8]       <user>
u_char          <errcode>
u_short         <dataport>
```

Description

The USER-LOGIN command registers a user with the CGC. A user cannot login more than one time. A user with the name ‘root’ is special in that it is the only user allowed to issue the SYS-RESET command.

The returned <dataport> field (in network byte order) is the number of a TCP port on which the CGC listens for connection requests. After connecting to this port the CGC will send CAN messages it received on CAN ports allocated by the user’s host application. CAN messages that should be transmitted by the CGC on one of the BI-0501’s CAN ports, should be sent to this TCP port. In effect two TCP connections are maintained by the CGC for each logged in user: a command channel which is used for all command related activity and a data channel which is used to relay CAN messages between the user’s host and the CGC.

The CGC will not send CAN messages as long as the user’s host application does not connect to the <dataport>, regardless if CAN ports on the BI-0501 were allocated by the user’s host application or not.

After initial login no resources such as ports, filters and the like are allocated to the user. These should be allocated using the appropriate commands. The USER-LOGIN command may fail if the user is already logged in or if the maximum number of users is exceeded. If the data port cannot be allocated by the CGC the <dataport> field will have the value 0.

3.9.2 USER-LOGOUT: Logout a user to the CGC

Request

```
R O <length> <user>
```

Response

```
S O <length> <errcode>
```

Fields

```
u_long          <length>
u_char[8]       <user>
u_char          <errcode>
```

Description

The USER-LOGOUT command removes the user from the CGC. All resources allocated to the user are deallocated and made available for allocation by other users. The <user> field is either empty (i.e. filled with eight NULLs) or the name of a user to logout. If the <user> field is empty then the user who is requesting the logout is logged out. Only the user logged in as 'root' may logout other users.

A side effect of the USER-LOGOUT command is the closing of the data connection between the user's host application and the CGC if one was allocated.

3.9.3 USER-STATUS: Request a list of logged in users

Request

```
R U <length>
```

Response

```
S U <length> <nrofusers> { <user> <ipaddress> <port> <protocol> }
```

Fields

```
u_long          <length>
u_long          <nrofusers>
u_char[8]       <user>
u_long          <ipaddress>
u_short         <port>
u_long          <protocol>
```

Description

The USER-STATUS command requests a list of logged in users. The <nrofusers> field indicates how many users are logged in. It is followed by a (possibly empty) list of users, where the <user> field indicates the user who is logged in, the <ipaddress> field is the IP address of the host from which the user is logged in, the <port> field is the port on the host from which the user is logged in and the <protocol> field names the protocol used to communicate with that user. The <protocol> number is the number of the Internet protocol according to the most recent Internet 'Assigned Numbers' RFC. Currently the TCP protocol is 6 and the UDP protocol is 17.

3.10 Control Packets

Several control packet are available to test and control the CGC as a whole. With the SYS-NOP packet one can test if the CGC is up and what the round trip time is, the SYS-RESET command restarts the CGC, deallocating all allocated resources. The SYS-INFO and SYS-CONFIGURE can be used to query basic BI-0501 configuration and configure some parameters.

3.10.1 SYS-NOP: No operation (a.k.a. ping)

Request

```
R N <length> <timestamp0>
```

Response

```
S N <length> <timestamp0> <timestamp1>
```

Fields

```
u_long          <length>
u_long          <timestamp0>
u_long          <timestamp1>
```

Description

The SYS-NOP command can be used to ascertain the correct operation of the CGC. The `<timestamp0>` field is a long word field in unspecified byte order of unspecified contents. Commonly one would put the current date and time in seconds since some base date (epoch).

The response packet places this timestamp unchanged in the first long word field. The `<timestamp1>` field is also a long word field containing the current time as known by the CGC in network byte order. This time value is the number of seconds elapsed since January 1st 1988.

3.10.2 SYS-RESET: Reset the CGC to initial state

Request

```
R R <length>
```

Response

```
S R <length> <errcode>
```

Fields

```
u_long          <length>
u_char          <errcode>
```

Description

The SYS-RESET command resets the CGC to its initial state and undoes all effects from commands such as USER-LOGIN, PORT-ADD, PORT-SETPHYS, WATCHDOG-ADD and FILTER-ADD. It will effectively restart the CGC server.

The SYS-RESET command can only be executed by a user currently logged in into the CGC with the name ‘root’. Note that as an effect of the SYS-RESET command the user ‘root’ will be logged out.

3.10.3 SYS-INFO: Request information about the CGC

Request

```
R V <length>
```

Response

```
S V <length> <timestamp> <cgcversion> <serialnr> <prodname>
<os> <cpuid> <temp> <eaddr> <e-ipaddr> <s-ipaddr>
<nrofmemblocks> { <memtype> <baseaddr> <length> }
<nrofcancntl> { <controller> <fifodepth> }
```

Fields

```
u_long          <length>
u_long          <timestamp>
u_char[32]      <cgcversion>
u_char[8]       <serialnr>
u_char[32]      <prodname>
u_char[32]      <os>
u_long          <cpuid>
u_long          <temp>
u_char[6]       <eaddr>
u_long          <e-ipaddr>
u_long          <s-ipaddr>
u_long          <nrofmemblocks>
u_long          <memtype>
u_long          <baseaddr>
u_long          <length>
u_long          <nrofcancntl>
u_long          <controller>
u_long          <fifodepth>
```

Description

The SYS-INFO command requests a configuration report from the BI-0501. The configuration report includes the CGC software version, information about the system hardware and software and the memory and CAN controllers on the BI-0501. For purposes of clarity the contents of the response is separated in several groups.

CGC State Group

`<time>` The time field is the current system time in UTC represented as the number of seconds since January 1, 1988.

`<cgcversion>` The `<cgcversion>` field is the version number of the CGC. It is a string with printable characters of unspecified contents.

BI-0501 System

`<serialnr>` This field is a string with printable characters of unspecified contents defining the serial number of the BI-0501.

`<prodname>` The `<prodname>` field is a string with printable characters of unspecified contents. It defines the product name and will usually be `BI-0501`.

<os> This field defines the name and version of the operating system on the BI-0501. It is again a string of unspecified contents.

<cpuid> The <cpuid> field defines the CPU type controlling the BI-0501. Currently it can have the following values:
0 Unknown
1 M68EN360
2 M68040

<temp> The <temp> field is the current board level temperature value as measured by the temperature sensor on the BI-0501. The temperature is measured in tenth of degrees Celsius. The initial temperature measurement can take up to 1 second to complete. If the initial measurement has not been completed the <temp> field will be set to all 1s.

Network Group

<eaddr> The <eaddr> field defines the Ethernet MAC address of BI-0501. It consists of six bytes in the order as is common for the notation of Ethernet addresses.

<e-ipaddr> The <e-ipaddr> field defines the BI-0501's IP address on the Ethernet port. The format is the same as that of all IP addresses in IP packets.

<s-ipaddr> The <s-ipaddr> field defines the BI-0501's IP address on the RS-232 port. The format is the same as that of all IP addresses in IP packets.

Memory Group

<nrofmemblocks>This field defines how many memory blocks are reported.

<memtype> The <memtype> field defines the type of memory. Currently it can have the following values:
0 Unknown
1 SRAM
2 DRAM
3 FLASH
4 DPRAM

The value DPRAM is specific to the M68EN360 on the BI-0501. It is a special area of on-chip memory of the M68EN360.

<baseaddr> The <baseaddr> field defines the start address of the memory block.

<length> The <length> field defines the length of the memory block in bytes.

CAN Group

<nrofcancntl>This field defines how many CAN controllers are installed.

<controller> The <controller> field defines the type of controller, transceiver and if the BI-0501 draws power from CAN bus. It can have the following values:

	Controller	Transceiver	Bus power
0	Unknown	Unknown	Unknown

1	82C200	82C251	No
2	82C200	82C252	No
3	SJA1000	82C251	No
4	SJA1000	82C252	No

If the transceiver draws power from the CAN bus, `1000` is added to the controller field in the previous list.

`<fifodepth>` The `<fifodepth>` field defines the number of bytes of FIFO memory associated with each CAN controller. If a CAN controller does not have a FIFO, then this number will be 0.

3.10.4 SYS-CONFIGURE: Change system configuration

Request

```
R J <length> <time> <e-ipaddr> <s-ipaddr> <rootname>
```

Response

```
S J <length> <time> <e-ipaddr> <s-ipaddr> <rootname>
```

Fields

```
u_long          <length>
u_long          <time>
u_long          <e-ipaddr>
u_long          <s-ipaddr>
u_char[8]       <rootname>
```

Description

The SYS-CONFIGURE command can be used to change a few essential parameters on the BI-0501. The `<time>` field defines the time to set the BI-0501's real-time clock to. It is specified as the number of seconds since January 1, 1988. The field `<e-ipaddr>` is the IP address to set for the BI-0501's Ethernet port. Likewise, the `<s-ipaddr>` field is the IP address to set for the BI-0501's RS-232 port. The `<e-ipaddr>` and `<s-ipaddr>` fields are specified in the same manner as addresses are specified in IP.

The `<rootname>` field defines the name of the user who should have so called root privileges. A user logging in into the CGC Server under this name is able to change the parameters mentioned above as well as resetting the CGC server and forcibly logging out other users. The `<rootname>` field is a character string of at most eight characters. The new name will be written in nonvolatile memory if not empty.

The change will only take effect after resetting the CGC server, changing the root name in itself will not reset the CGC server. If the user who issued the SYS-CONFIG request packet is not the root user, the `<time>`, `<e-ipaddr>` and `<s-ipaddr>` fields will be set in the response packet, but the `<rootname>` field will be empty (i.e. filled with eight NULLs).

Only a user who is logged in as 'root' can change these parameters. After changing either the BI-0501's clock and/or the IP addresses, and sending the response, the CGC application will reset itself without sending a reset response packet.

If a parameter does not need to be changed, then it should be given the value 0 in the request packet. The response packet contains the new actual values of the system's time and IP address. Note that some address values are illegal such as a local loopback address

(127.*.*.*) and so called broadcast addresses (255 in any byte position). If an attempt is made to set any such an illegal address, the response will contain the original address and the address is not changed. No error indication is given.

3.10.5 SYS-SERIAL: Configure the RS-232 port

Request

```
R K <length> <speed> <databits> <parity> <stopbits>
```

Response

```
R K <length> <speed> <databits> <parity> <stopbits>
```

Fields

```
u_long          <length>
u_long          <speed>
u_long          <databits>
u_long          <parity>
u_long          <stopbits>
```

Description
The SYS-SERIAL command sets the speed, number of databits, parity and number of stopbits to configure the for the serial port on the BI-0501.

The `<speed>` field is the baud rate to use on the serial port. It may have the following value: 75, 110, 135, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, 38400 or 57600. Any value outside this range is rejected. The `<databits>` field must either have the value 7 or 8 and defines the number of databits in a word. The `<parity>` must be one of: `N` for no parity, `O` for odd parity, `E` for even parity, `M` for mark parity or `S` for space parity. The `<stopbits>` field finally must either have the value `1` for 1 stopbit, 2 for 1.5 stopbits or 3 for 2 stopbits.

If any of the fields does not have the correct value the request is ignored and the current settings are returned in the reply packet. If the values are correct the serial port is reconfigured and the new values are stored in nonvolatile memory to be used as power-on defaults. Note that if the serial port is in use no further use will be possible until the system the BI-0501 is connected to has changed its serial port settings too.

Only the user currently logged in as `root' may change the serial port configuration. For all other users the current settings are returned.

3.11 CAN Port Packets

Before a CAN port can be used, a user must allocate it. Once allocated, the port can be used to transmit and receive CAN messages. A port can only be allocated by a single user although a user who is logged in as ‘root’ can forcibly deallocate any allocated CAN port.

The PORT-STATUS command reports which CAN ports are in use by which users. Combining this with the response from a USER-STATUS command, one can quickly find the host that is using a specific port. The speed at which a CAN port should be operated is specified with the PORT-SETSPEED command and is reported by the PORT-GETSPEED command.

Ports are numbered starting at 0 for CAN port 1 up to and including 7 for CAN port 8. Port numbers outside this ranged are flagged with an error code.

3.11.1 PORT-ADD: Allocate one or more CAN ports to a user

Request

```
R A <length> <nrofports> { <port> }
```

Response

```
S A <length> <nrofports> { <port> <errcode> }
```

Fields

```
u_long          <length>
u_long          <nrofports>
u_long          <port>
u_char          <errcode>
```

Description

The PORT-ADD command allocates one or more ports to a user. The request lists the number of ports to add followed by an enumeration of those ports. The response indicates the same number of ports and for each port mentioned in the request an error code in the same order as in the request.

Adding ports may fail if a user is not logged in to the CGC, a port is already allocated to another user or if the requested port does not exist.

3.11.2 PORT-DELETE: Remove one or more ports from a user

Request

```
R D <length> <nrofports> { <port> }
```

Response

```
S D <length> <nrofports> { <port> <errcode> }
```

Fields

```
u_long          <length>
u_long          <nrofports>
u_long          <port>
u_char          <errcode>
```

Description

The PORT-DELETE command removes one or more ports from the list of ports allocated to user. The `<nrofports>` field indicates how many ports should be deleted and is followed by an enumeration of those ports. The response indicates for each deallocated port an error code in the same order as in the request packet.

For each port that is deallocated, the list of watchdog messages and filters is removed too. The CAN controller associated with the port is reset to its initial state (i.e. no more messages are received or transmitted and the actual CAN controller associated with the port is set in the reset state).

Port deallocation may fail if the user is not logged in to the CGC, a port is not allocated to the requesting user or if a port does not exits. A user who is logged in as ‘root’ may deallocate any port regardless if the port was previously allocated to that user.

3.11.3 PORT-STATUS: Report CAN port allocation status

Request

```
R P <length> <nrofports> { <port> }
```

Response

```
S P <length> <nrofports> { <port> <errcode> <speed> <user> }
```

Fields

```
u_long          <length>
u_long          <nrofports>
u_long          <port>
u_char          <errcode>
u_long          <speed>
u_char[8]       <user>
```

Description

The PORT-STATUS command reports the allocation status for a list of ports. The <nrofports> field indicates from how many ports the status is requested and is followed by an enumeration of those ports. The reply indicates this same number of ports followed by an enumeration indicating the status for each port.

The <port> field in the response names the port, the <errcode> field tells if there is an error associated with the port, and the <user> field names the user to whom the port is allocated. If a port is not allocated the <user> field is empty (i.e. filled with eight NULLs).

The <speed> field contains the bit rate at which the CAN port operates.

3.11.4 PORT-GETSPEED: Get CAN port bit rate

Request

```
R M <length> <nrofports> { <port> }
```

Response

```
S M <length> <nrofports> { <port> <errcode> <speed> }
```

Fields

```
u_long          <length>
u_long          <nrofports>
u_long          <port>
u_char          <errcode>
u_long          <speed>
```

Description

The PORT-GETSPEED command reports the current bit rate at which a port operates. The <nrofports> fields indicates of how many ports the speed is requested and it is followed by an enumeration of those ports. The reply indicates this same number of ports followed by an enumeration indicating the speed of each port.

The <errcode> field details if there is an error associated with the port while the <speed> field indicates the bit rate the port operates at.

3.11.5 PORT-SETSPEED: Set CAN port bit rate

Request

```
R B <length> <nrofports> { <port> <speed> }
```

Response

```
S B <length> <nrofports> { <port> <errcode> <speed> }
```

Fields

```
u_long          <length>
u_long          <nrofports>
u_long          <port>
u_char          <errcode>
u_long          <speed>
```

Description

The PORT-SETSPEED command sets the bit rate at which the CAN port operates. The <nrofports> field indicates how many ports should be changed. It is followed by an enumeration of the port number and the speed. The <speed> field is the bit rate the port should operate at.

The <errcode> field in the response details if there is an error associated with the port and the <speed> field has the actual bit rate of the port. If the named port does not exist the value of the <speed> field is meaningless.

3.11.6 PORT-STATISTICS: Get CAN port statistics

Request

```
R B <length> <port>
```

Response

```
S B <length> <port>
<txmsg> <txbyte> <txerr>
<rxmsg> <rxbyte> <rxerr>
<overrun> <erract> <errpasv> <busoff> <rxfull> <txfull>
```

Fields

```
u_long          <length>
u_long          <port>
u_long          <txmsg>
u_long          <txbyte>
u_long          <txerr>
u_long          <rxmsg>
u_long          <rxbyte>
u_long          <rxerr>
u_long          <overruns>
u_long          <erract>
u_long          <errpasv>
u_long          <busoff>
```

u_long <rxfull>
u_long <txfull>

Description

The PORT-STATISTICS command returns the values of several counters the CAN driver on the BI-0501 maintains. The <port> field names the CAN port for which the statistics should be returned. If the named port does not exist, all counters are set to 0. The other fields have the following meaning:

<txmsg> The number of CAN messages transmitted;
<txbyte> The number of data bytes transmitted;
<txerr> The number of transmit errors encountered;
<rxmsg> The number of CAN messages received;
<rxbyte> The number of data bytes received;
<rxerr> The number of receive errors encountered;
<overrun> The number of times an overrun error was signalled by the CAN controller;
<erract> The number of times the CAN controller entered the error active state;
<errpasv> The number of times the CAN controller entered the error passive state;
<busoff> The number of times the CAN controller entered the bus off state;
<rxfull> The number of times a CAN message was left in the FIFO because the driver's receive buffer was full;
<txfull> The number of times a CAN message was not accepted for transmission because the driver's transmit buffer was full.

Not all counters are reliable: because not all CAN controllers indicate transmission and reception errors exactly, the <txerr> and <rxerr> fields may be zero. Also, some CAN controllers do not indicate when they make the transition from the error active to the error passive state exactly. Therefore the <errpasv> and <erract> fields are only rough indicators of the number of times the error passive and error active modes have been entered by the CAN controller.

3.12 Watchdog Packets

Sometimes it is desirable to have the CGC to send CAN messages on a CAN port at regular intervals. This may be periodic or if no packets have been received on a CAN port for some time. Both functions can be implemented using so called watchdog messages.

Watchdog messages are identified by a message number. This is a number ranging from 0 to 7 that defines the position of the watchdog message in a list of watchdog messages for a port. Higher numbers mean lower priority.

A special type of watchdog message is the so called “hardware watchdog message”. This type of message is sent on the port whenever the BI-0501's hardware watchdog exception handler is invoked. Messages that are sent on a port because of hardware watchdog invocation are *not* matched against filter rules for the port. Note that the number of hardware watchdog messages should be kept to a minimum as the total system state may not be certain and not enough time might be left to transmit all hardware watchdog messages.

By using filter rules it is possible to have a watchdog message relayed to the user's host if one is sent on a CAN port.

There is a limit of eight watchdog messages for each port.

3.12.1 WATCHDOG-ADD: Add watchdog message to a port

Request

```
R H <length> <port> <messagenr>
<condition> <timeout> <message>
```

Response

```
S H <length> <port> <errcode0> <messagenr>
<condition> <timeout> <errcode1> <message>
```

Fields

```
u_long          <length>
u_long          <port>
u_char          <errcode0>
u_long          <messagenr>
u_long          <condition>
u_long          <timeout>
u_char          <errcode1>
u_char[20]      <message>
```

Description

The WATCHDOG-ADD command adds a watchdog message to the port named in the <port> field.

The field <messagenr> is a unique number identifying the watchdog number. The <condition> field indicates under which condition the message should be sent. If the <condition> field has the value I then the message is sent whenever the port has been idle (i.e. no messages transmitted nor received) for the number of microseconds in the <timeout> field. If the <condition> field has the value T then the message is sent every <timeout> microseconds.

To send multiple messages in response to a single watchdog trigger, one should set the <condition> and <timeout> fields of these messages to the same value. Their priority numbers should be sequential. Then, when the watchdog is triggered, the CGC server sends the messages in priority order all at once. This feature can be used to implement CAN application protocols such as CANopen. The message with the highest priority in the group would be a so called SYNC message and could be followed by one or more COMMAND messages that should be transmitted within the synchronisation window.

The value H in the <condition> field is special in that it means that the message is a so called “hardware watchdog message”. The <messagenr> field is a unique number identifying the priority of the message. Lower numbers mean higher priority. The <timeout> field is ignored for hardware watchdog messages.

The WATCHDOG-ADD command may fail if either the named port does not exist or is allocated to another user, if the message number already exists or if the <condition> field has an invalid value.

3.12.2 WATCHDOG-DELETE: Remove watchdog from a port

Request

```
R X <length> <port> <messagenr>
```

Response

```
S X <length> <port> <errcode> <messagenr>
```

Fields

```
u_long          <length>
u_long          <port>
u_char          <errcode>
u_long          <messagenr>
```

Description

The WATCHDOG-DELETE command removes the watchdog message indicated by the <messagenr> field. If the indicated message does not exist an error is returned in the <errcode> field.

3.12.3 WATCHDOG-STATUS: Show watchdog status on a port

Request

```
R W <length> <port>
```

Response

```
S W <length> <port> <errcode>
<messagenr> <condition> <timeout> <message>
```

Fields

```
u_long          <length>
u_long          <port>
u_char          <errcode>
u_long          <messagenr>
u_long          <condition>
u_long          <timeout>
u_char[20]      <message>
```

Description

The WATCHDOG-STATUS command lists all messages associated with the port named in the <port> field. The <errcode> field is set to an appropriate error code if the user performing the WATCHDOG-STATUS command has not previously allocated the port.

The fields <messagenr>, <condition>, <timeout> and <message> list the message's number, the condition under which it is sent, the timeout value in microseconds and the message contents.

A user who is logged in as 'root' may always ask the watchdog status of any port.

3.13 Filter Packets

By using filter rules it is possible to limit the amount of traffic from a CAN port to a user's host. Filter rules match messages and take action based on the contents of the message. Two basic actions are possible: either to relay the message to the user's host or to ignore it. Both actions can be performed for messages that are transmitted on or received from a port. A filter rule also counts the number of matches.

Filter rules are ordered by means of a filter number which ranges from 0 to 7. Higher numbers mean lower priority. There is always an unnumbered default filter rule, with the lowest priority, which states that any message sent on a port always matches and thereby allowing the message to actually to be sent. The normal action to take, when matching messages in filter rules, is to stop matching as soon as a message matches a certain rule.

Setting up a filter rule which matches watchdog messages makes it possible to be informed when a watchdog message is sent.

3.13.1 FILTER-ADD: Add CAN message filter to ports

Request

```
R I <length> <port> <filternr> <action> <message> <messagemask>
```

Response

```
S I <length> <port> <errcode> <filternr> <action>
<message> <messagemask>
```

Fields

```
u_long          <length>
u_long          <port>
u_char          <errcode>
u_long          <filternr>
u_long          <action>
u_char[20]      <message>
u_char[20]      <messagemask>
```

Description

The FILTER-ADD command adds a message filter to the port named in the `<port>` field. The `<filternr>` field indicates which filter to add and must be a unique number. The `<action>` field indicates the disposition of a message that matches. A value of `TS` or `RS` means that the message should be passed on to the user's host when it is either transmitted or received over the port. A value of value of `TI` or `RI` means that no special action should be taken if the message matches the rule.

Note that for matching a message both the `<message>` and `<messagemask>` as well the action field are searched. This means that if a higher priority rule matches a message, but the message does not match the direction (transmit or receive), the filter will search for another filter that does match and has the transmit direction of the message under scrutiny.

3.13.2 FILTER-DELETE: Remove CAN message filter from ports

Request

```
R Q <length> <port> <filternr>
```

Response

```
S Q <length> <port> <errcode> <filternr>
```

Fields

```
u_long          <length>
u_long          <port>
```

```
u_char          <errcode>
u_long          <filternr>
```

Description

The FILTER-DELETE command removes the filter with number <filternr> from the port named in the <port> field. If the filter does not exist an error code is returned in the <errcode> field.

3.13.3 FILTER-STATUS: Report CAN message filter for ports

Request

```
R F <length> <port>
```

Response

```
S F <length> <port> <errcode>
<nroffilters> { <filternr> <action> <message> <messagemask>
<counter> }
```

Fields

```
u_long          <length>
u_long          <port>
u_char          <errcode>
u_long          <nroffilters>
u_long          <filternr>
u_long          <action>
u_char[20]      <message>
u_char[20]      <messagemask>
u_long          <counter>
```

Description

The FILTER-STATUS command lists all filters associated with the port named in the <port> field. The <errcode> field is set to an appropriate error code if the user performing the FILTER-STATUS command has not previously allocated the port. A user who is logged in as ‘root’ may always ask the filters of any port.

The <nroffilters> field names the number of filters associated with the port. The <filternr>, <action>, <message> and <messagemask> have the same meaning as described in the FILTER-ADD command. The <counter> field indicates how many messages have matched the filter.

Note that the only method for resetting a counter is to delete the associated filter rule and recreate it.

3.14 CAN Packets

Messages received on a CAN port are sent from the CGC to the user’s host as unsolicited packets on the data connection. As soon as a port is allocated to a user the CGC may send these packets. This means that the user’s receiving application must be ready to accept them at any time.

CAN messages sent to the CGC from the users host for transmission on a CAN port are sent as unsolicited packets on the data connection too. They will be transmitted on a CAN port if the port is allocated by the user and the CAN message matches any of the filter rules defined for the port.

To send messages on a CAN port the CAN-SEND packet is used, likewise the CGC sends unsolicited CAN-SEND packets to the user when a filter rule matches a CAN message transmitted on a CAN port. CAN messages received on a port are sent as unsolicited CAN-RECEIVE packets.

Because of network and queuing delays errors on a CAN port may occur some time after messages have been offered for transmission. The CAN-ERROR packet reports errors with two timestamps: one is a timestamp which the user had attached when sending the CAN-SEND packet and one that indicates the actual time the error occurred.

3.14.1 CAN-SEND: Send CAN messages to a port

Response

```
U C <length> <port> <nrofmessages> { <message> }
```

Fields

```
u_long          <length>
u_long          <port>
u_long          <nrofmessages>
u_char[20]      <message>
```

Description

If the receiver of the CAN-SEND packet is the user's host then the CAN-SEND packet contains a collection of CAN messages sent to the indicated CAN port. If the receiver is the CGC then the CAN-SEND packet lists the number of CAN messages to be transmitted on the indicated CAN port. Up to eight CAN messages can be transmitted with a single CAN-SEND packet.

3.14.2 CAN-RECEIVE: Received CAN messages

Response

```
U Y <length> <port> <nrofmessages> { <message> }
```

Fields

```
u_long          <length>
u_long          <port>
u_long          <nrofmessages>
u_char[20]      <message>
```

Description

The CAN-RECEIVE packet is sent by the CGC whenever messages have been received on the port named in the `<port>` field. The `<nrofmessages>` field tells how many messages have been received and is followed by the list of received messages.

3.14.3 CAN-ERROR: CAN message errors

Response

```
U E <length> <port>
<nrofmessages> { <timestamp0> <timestamp1> <errcode> }
```

Fields

```
u_long          <length>
u_long          <port>
u_long          <nrofmessages>
u_long          <timestamp0>
u_long          <timestamp1>
u_char          <errcode>
```

Description

The CAN-ERROR packet is sent by the CGC whenever an error occurs transmitting or receiving messages on the port named in the <port> field. The <errcode> field details the error that occurred. The <timestamp0> is the timestamp sent with the original CAN-SEND message, the <timestamp1> field is the time at which the error occurred. If the error occurs while no messages are being transmitted the <timestamp0> field will have the value zero.

Chapter 4

CGC Host Library

4.1 Introduction

The CGC host library (or CGC library) for the CGC protocol handles most of the details of communications management between a host application and the CGC server on the BI-0501. It is a C language based library which defines a number of data structures and functions which collectively manage the CGC protocol.

In the following paragraphs an overview of the library is given along with a description of the various functions available for application programmers. It is assumed that the reader is familiar with the CGC protocol described in chapter 3.

4.2 Prerequisites

Each source code file making calls to the CGC library must include the file `<cgclib.h>`. This file defines the necessary data structures and constants used by the library functions. It is probably also necessary to include those system files that define data types such as `struct sockaddr_in`. For example, on a typical UNIX system one should have the following include order:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <cgclib.h>
```

All functions and data types declared in the CGC library are prefixed with the string `cgc_`, all constants and macros are prefixed with the string `CGC_`.

The CGC library functions do very little error checking on the parameters passed to them. No particular checking is done for NULL pointers or values that are out of bounds for a certain parameter. The function descriptions in the following paragraphs however do specify the expected parameters and their range exactly. It is left to the application programmer's discretion to ensure that the correct parameters values are passed.

Two types of errors may occur when using the CGC library: those detected by the operating system the application is used on and those detected by the CGC server on the BI-0501. In order not to confuse the two, all CGC library functions return either 0 if they completed successfully or -1 if an operating system error was detected. Most operating systems provide for some means (e.g. the global `errno` variable on UNIX systems) of detecting the exact nature of the last error encountered. Errors detected by the CGC server are usually stored in a separate variable on return from a CGC library function. Therefore the user should check both the return value from any CGC library function as well as the error code returned from the CGC server.

The CGC library proper consists of two layers: an operating system independent upper layer and a more or less operating system dependent lower layer. The upper layer manages communication with

the CGC server through a function call interface. The lower layer handles the actual packet transmission and reception and is operating system dependent. The upper layer fills packet buffers based on parameters passed to CGC library functions and offers these packet buffers to the lower layer for transmission. The lower layer in turn transmits these packet buffers to the CGC server and awaits response packets from it. The response packets are then decoded and passed back to the upper layer for further processing. The main reason for this two-level layering is to allow applications with different synchronization requirements to be implemented without a specific model of synchronization imposed on them by the CGC library. Windowed style applications usually have a radically different event loop handling than simple server style applications.

Memory allocated and deallocated by the CGC library uses the system functions `malloc()` and `free()`. The CGC library never calls `malloc()` with a 0 size, nor does it ever call `free()` with a NULL pointer. Some of the functions in the CGC library may return pointers to structures allocated with `malloc()`. These pointers should be passed to `free()` when the memory they occupy is no longer in use. If such a structure contains pointers (for example to character strings) then the memory referenced by those pointers should be released first. If a pointer has the value NULL no call to `free()` should be done of course.

Several functions in the CGC library reference a structure called `can_msg`. This structure holds a CAN message to be transmitted or received. It has the following declaration:

```
struct can_msg {
  unsigned long   cm_tm;
  unsigned long   cm_id;
  unsigned short  cm_rsv;
  unsigned char   cm_rtr;
  unsigned char   cm_dlc;
  unsigned char   cm_data[8];
};
```

where the fields have the following meaning: the `cm_tm` is a timestamp field measured in microseconds, the `cm_id` field is the CAN message ID, the `cm_rsv` field is reserved and should be 0, the `cm_rtr` field is the CAN RTR bit and is non-zero if the RTR bit is set, the `cm_dlc` field is the CAN data length code, the `cm_data` field, finally, holds the data bytes contained in the CAN message.

When sending CAN messages the `cm_tm` field is ignored by the CGC server as is the `cm_rsv` field. The other fields are expected to have correct values as demanded by the CAN protocol. This means in particular that the `cm_id` and `cm_dlc` fields should have legal values, the CGC server does not check for this. The `cm_data` field is always eight bytes long, even if the `cm_dlc` field indicates that there are less that eight bytes in the message.

All IP addresses and IP port numbers returned by the CGC library and passed from the user's application to the CGC library functions are in network byte order.

4.3 Connection Management

The connection management functions initialise the CGC library and manage TCP connections to the CGC server on the BI-0501.

4.3.1 cgc_init()

SYNOPSIS

```
#include <cgclib.h>
int cgc_init(void);
```

PARAMETERS

None.

DESCRIPTION

The `cgc_init()` function initializes the CGC library and should therefore always be the first call before any other CGC library function can be called.

RETURN VALUE

0	CGC library is initialised;
-1	CGC library could not be initialised.

SEE ALSO

`cgc_deinit()`.

4.3.2 cgc_deinit()

SYNOPSIS

```
#include <cgclib.h>
int cgc_deinit(void);
```

PARAMETERS

None.

DESCRIPTION

The `cgc_deinit()` function should be called to deallocate all resources in the CGC library. It should be called when the application is done using any of the other functions of the CGC library. Note that `cgc_deinit()` does not deallocate resources held by the application on the CGC server. This is the responsibility of the application.

After `cgc_deinit()` returns only `cgc_init()` may be called.

RETURN VALUE

0	CGC library is initialised;
-1	CGC library could not be initialised.

SEE ALSO

`cgc_init()`.

4.3.3 cgc_conn_cmdch()

SYNOPSIS

```
#include <cgclib.h>
int cgc_conn_cmdch(unsigned long addr, unsigned short port,
  int *cmdfd);
```

PARAMETERS

`unsigned long addr`	TCP host address (in network byte order) where the CGC server is located;
`unsigned short port`	TCP port number (in network byte order) at which the CGC server listens for connection requests;
`int *cmdfd`	Pointer to a location where the file descriptor referencing the connection is stored.

DESCRIPTION

The `cgc_conn_cmdch()` function initiates a TCP connection to the CGC server specified by the parameters `addr` and `port`. The file descriptor referencing the connection is returned in the location pointed to by the `cmdfd` pointer. If the connection cannot be established `cgc_conn_cmdfd()` returns the value -1 and value in the location pointed to by `cmdfd` is undefined.

The `cgc_conn_cmdfd()` function is typically the first function in the CGC library to be called after calling `cgc_init()`. It establishes the command connection between the CGC server and the user's host application. The function to call after `cgc_conn_cmdfd()` is the `cgc_u_login()` function which returns the port number to use for the data connection (i.e. for the exchange of CAN messages). This port number should then be passed to the `cgc_conn_datach()` function to create the data connection. Finally one should call the function `cgc_conn_pwrch()` to be able to receive power down messages from the BI-0501.

RETURN VALUE

0	Connection established;
-1	Connection establishment failed.

SEE ALSO

`cgc_conn_datach()`, `cgc_conn_pwrch()`, `cgc_disconnect()`.

4.3.4 cgc_conn_datach()

SYNOPSIS

```
#include <cgclib.h>
int cgc_conn_datach(unsigned long addr, unsigned short port,
  int *datafd);
```

PARAMETERS

`unsigned long addr`	TCP host address (in network byte order) where the CGC server is located;
`unsigned short port`	TCP port number (in network byte order) at which the CGC server listens for connection requests;
`int *datafd`	Pointer to a location where the file descriptor referencing the connection is stored.

DESCRIPTION

The `cgc_conn_datach()` function initiates a TCP connection to the CGC server specified by the parameters `addr` and `port`. The file descriptor referencing the connection is returned in the location pointed to by the `datafd` pointer. If the connection cannot be established `cgc_conn_datafd()` returns the value -1 and value in the location pointed to by `datafd` is undefined.

RETURN VALUE

0	Connection established;
-1	Connection establishment failed.

SEE ALSO

`cgc_conn_cmdch()`, `cgc_conn_pwrch()`, `cgc_disconnect()`.

4.3.5 cgc_conn_pwrch()

SYNOPSIS

```
#include <cgclib.h>
int cgc_conn_pwrch(unsigned long addr, unsigned short port,
  int *pwrfd);
```

PARAMETERS

`unsigned long addr`	The UDP host address (in network byte order) on which to receive power failure notification from the CGC server;
`unsigned short port`	UDP port number (in network byte order) on which to receive power failure notification;
`int *pwrfd`	Pointer to a location where the file descriptor referencing the connection is stored.

DESCRIPTION

The `cgc_conn_pwrch()` creates a UDP “connection” on which to receive power failure messages from the CGC server. The `addr` parameter is the address of the host on which to receive power failure notification, the `port` parameter is the UDP port on which to receive those messages. The `pwrfd` parameter finally is a pointer to a location where the file descriptor referencing the connection is stored.

Note that power down notification uses the UDP protocol. This means that there is no permanent UDP connection between the CGC server and the user's host application for these types of messages. Therefore the `addr` parameter is not used by the `cgc_conn_pwrch()` function, only the `udpport` parameter is used.

RETURN VALUE

0	Connection established;
-1	Connection establishment failed.

SEE ALSO

SYS-POWERDOWN packet, `cgc_conn_cmdch()`, `cgc_s_pwrnotify()`.

4.3.6 cgc_disconnect()

SYNOPSIS

```
#include <cgclib.h>
int cgc_disconnect(int fd);
```

PARAMETERS

`int fd`	The file descriptor from which to disconnect a connection to the CGC server;

DESCRIPTION

The `cgc_disconnect()` function closes the file descriptor referencing a connection with the CGC server. The file descriptor `fd` can be any of the command, data or power failure file descriptors.

RETURN VALUE

0	Disconnected from CGC server;
-1	Disconnect failed.

SEE ALSO

`cgc_conn_cmdch()`, `cgc_conn_datach()`, `cgc_conn_pwrch()`.

4.3.7 cgc_doio()

SYNOPSIS

```
#include <cgclib.h>
int cgc_doio(int cmdfd, int datafd, int pwrfd);

unsigned char cgc_pkt_in[CGC_PKT_MAX], cgc_pkt_out[CGC_PKT_MAX];
int cgc_pkt_ilen, cgc_pkt_olen;
```

PARAMETERS

`int cmdfd`	The file descriptor to use for the normal CGC server I/O;
`int datafd`	The file descriptor to use to receive (and send) CAN messages;
`int pwrfd`	A file descriptor on which to listen for power down messages from the CGC server.

DESCRIPTION

The `cgc_doio()` function is called by the other CGC library functions when they want to read or write packets. The `cgc_doio()` function takes packets to send from the global variable `cgc_pkt_out` and the variable `cgc_pkt_olen`. The packet is written to the file descriptor referenced by the parameter `cmdfd`. Next, `cgc_doio()` will try to read a packet back from the CGC server on the same file descriptor and store the result in the global variable `cgc_pkt_in` and the length in `cgc_pkt_ilen`. While reading from the `cmdfd` file descriptor `cgc_doio()` will also listen to messages coming from the `pwrfd` file descriptor. If a message is read from that file descriptor, the global variable `cgc_powerdown` will be set to non-zero and the function defined by `cgc_s_pwrnotify` will be called if not NULL.

If any of the three parameters has the value -1, the `cgc_doio()` will not attempt to send or receive on the associated connection.

RETURN VALUE

0	No error
-1	A read or write error occurred.

SEE ALSO

4.4 User Management

The user management functions allow one to connect to the CGC server on the BI-0501 and subsequently use the BI-0501's CAN ports. Before any other function can be used it is necessary to login on the CGC server, using the `cgc_u_login()` function. After using the CGC server the user should call the `cgc_u_logout()` function which deallocates all resources allocated to the user. The `cgc_u_status()` function can be used to inquire about the currently logged in users.

4.4.1 cgc_u_login()

SYNOPSIS

```
#include <cgclib.h>
int cgc_u_login(int *err, char *name, unsigned short *port);
```

PARAMETERS

`int *err`	Pointer to a location where the result of the login is stored;
`char *name`	Pointer to the name of the user wishes to login to the CGC server;
`unsigned short *port`	Pointer to a location where the port number for the data connection is stored.

DESCRIPTION

The `cgc_u_login()` function attempts to login the named user. The `name` parameter should point to a NULL terminated character string of at most eight characters. The `err` parameter should point to a location where the error code as returned from the CGC server is stored. If the `*err` parameter as the value `CGC_ERR_NONE` then the login has succeeded.

The `port` parameter is a pointer to a location where the port number (in network byte order) for the data connection is stored. That value is typically passed to `cgc_conn_datach()`.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications error occurred.

SEE ALSO

USER-LOGIN packet, `cgc_u_logout()`, `cgc_u_status()`.

4.4.2 cgc_u_logout()

SYNOPSIS

```
#include <cgclib.h>
int cgc_u_logout(int *err, char *name);
```

PARAMETERS

`int *err`	Pointer to a location where the result of the login is stored;
`char *name`	Pointer to the name of the user who wishes to logout from the CGC server.

DESCRIPTION

The `cgc_u_logout()` function deallocates all resources assigned to the named user. The `name` parameter should point to a NULL terminated character string of at most eight characters. The `err` parameter should point to a location where the error code as returned from the CGC server is stored. If the `*err` parameter has the value `CGC_ERR_NONE` the logout has succeeded. The user wishing to perform the logout function must be the same user that performed the login.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications error occurred.

SEE ALSO

USER-LOGOUT packet, `cgc_u_login()`, `cgc_u_status()`

4.4.3 cgc_u_status()

SYNOPSIS

```
#include <cgclib.h>
int cgc_u_status(int *nrofusers, struct cgc_user **user)
```

PARAMETERS

`int *nrofusers`	Pointer to a location where the number of entries pointed to by the `struct cgc_user` pointer `user` is stored;
`struct cgc_user **user`	Pointer to an array of `cgc_user` information structures.

DESCRIPTION

This function requests the list of users currently logged in with the CGC server and their origin. The parameter `nrofusers` should point to a location where the number of users currently logged in is stored. The `user` parameter is a pointer to a location where a pointer is stored which points to an array of type `cgc_user`. This array has `*nrofusers` entries and is allocated by the CGC library. The `cgc_user` structure has the following declaration:

```
struct cgc_user {
  int           err;
  char          *name
  unsigned long ipaddr;
  unsigned shortport;
  int           proto;
};
```

The `err` field is an individual error code for each entry, the `name` field points to a NULL character string of at most 8 characters. The `ipaddr` and `port` fields define the IP address and port number on the host from which the user is logged in. Both are in network byte order. The `proto` field finally defines the protocol the user is using.

If no users are logged in the value returned in `nrofusers` will be 0. The memory allocated for the `cgc_user` array should be deallocated using `free()`.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications error occurred.

SEE ALSO

USER-STATUS packet, `cgc_u_login()`, `cgc_u_logout()`

4.5 CGC Server Configuration

Functions in this paragraph allow querying the configuration of the CGC server and the BI-0501. Some parameters may also be changed with the `cgc_s_config()` function. When changing parameters on the CGC server or when resetting the CGC server (with `cgc_s_reset()`) all TCP/IP connections will be broken by the CGC server and have to be re-established by the host applications.

4.5.1 cgc_s_nop()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_nop(unsigned long *ts0, unsigned long *ts1);
```

PARAMETERS

`unsigned long *ts0`	Pointer to a location where the local hosts' time stamp is stored;
`unsigned long *ts1`	Pointer to a location where the return timestamp from the CGC server will be stored.

DESCRIPTION

The `cgc_s_nop()` function can be used to test the availability of the CGC server. The `ts0` parameter should point to a location where a timestamp is be stored. This timestamp is a simple `unsigned long` value and is not further interpreted by the CGC server but sent back unaltered. Commonly one would use the value from the hosts system's `time()` system call. The `ts1` parameter is a pointer to a location where a timestamp value as returned from the CGC server.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

SYS-NOP packet, `cgc_s_reset()`.

4.5.2 cgc_s_reset()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_reset(int *err);
```

PARAMETERS

`int *err`	Pointer to a location where the (possible) return value from the CGC server will be stored.

DESCRIPTION

The `cgc_s_reset()` function resets the CGC server to its initial state. This has the effect of logging out all logged in users and deallocating all resources such as CAN ports and message filters allocated to those users. Only the user who is currently logged in as 'root' is allowed to reset the CGC server. As a consequence of resetting the CGC server all TCP/IP connections to the CGC server are no longer operational and should be closed.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	A communications failure with the CGC occurred.

SEE ALSO

SYS-RESET packet, `cgc_s_nop()`.

4.5.3 cgc_s_info()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_info(struct cgc_info **info);
```

PARAMETERS

`struct cgc_info **info` Pointer to a location where the pointer to the `cgc_info` structure will be stored.

DESCRIPTION

The `cgc_s_info()` function requests the configuration information of the CGC server and the BI-0501 to be returned. The parameter `info` should point to a location where a pointer to an allocated `struct cgc_info` will be stored. The `cgc_info` structure has the following declaration:

```
struct cgc_info {
  unsigned long          ts;
  char                   *version;
  char                   *serialnr;
  char                   *prodname;
  char                   *os;
  unsigned long          cpuid;
  unsigned long          temp;
  unsigned char          eaddr[6];
  unsigned long          eipaddr;
  unsigned long          sipaddr;
  unsigned long          nrofmemblocks;
  struct cgc_memconfig   *memconfig;
  unsigned long          nrofcanports;
  struct cgc_canconfig   *canconfig;
};
```

Each of the fields has the same layout as described in the SYS-INFO packet description in the previous chapter. The `char*` fields are pointer to NULL terminated character strings and should be deallocated before deallocting the `cgc_info` structure. The `cgc_memconfig` and `cgc_canconfig` structures have the following declaration:

```
struct cgc_memconfig {
  unsigned long   memtype;
  unsigned long   baseaddr;
  unsigned long   length;
};
```

and

```
struct cgc_canconfig {
  int             controller;
  unsigned long   fifodepth;
};
```

As with the character pointers in the `cgc_info` structure, the `memconfig` and `canconfig` pointers should be deallocated before deallocating the `cgc_info` structure.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications error occurred.

SEE ALSO

SYS-INFO packet, `cgc_s_config()`

4.5.4 cgc_s_config()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_config(unsigned long *tm,
  unsigned long *eipaddr, unsigned long *sipaddr, char **root);
```

PARAMETERS

`unsigned long *tm`	Pointer to the location where the new CGC server time is found and where the new actual value is stored;
`unsigned long *eipaddr`	Pointer to the location where the new CGC server IP address for the BI-0501's Ethernet interface is stored;
`unsigned long *sipaddr`	Pointer to the location where the new CGC server IP address for the BI-0501's serial port is stored;
`char **root`	Pointer to the location where a pointer to the new name for the 'root' user is stored.

DESCRIPTION

The `cgc_s_config()` function changes three essential parameters on the BI-0501: the current time and the Ethernet and serial port IP addresses. The `tm` parameter should point to the location where the new system time can be found to be set. Likewise, the `eipaddr` and `sipaddr` parameters should point to locations where the new IP addresses for the BI-0501's Ethernet and serial port are to be found. Both the Ethernet IP address and serial port IP address should be in network byte order. The parameter `root` is a pointer to a location where a pointer to the new 'root' name is stored. If the name should not be changed then either this pointer may be NULL or should point to a NULL pointer. If a value should not be changed, it should be given the value 0.

After return of `cgc_s_config()` the locations pointed to by `tm`, `eipaddr` and `sipaddr` are updated with the actual values as set by the CGC server. Only the user who is currently logged in as 'root' may change these parameters. After changing any of the `tm`, `eipaddr` or `sipaddr` values the CGC server will reset itself. If the name of the 'root' user was changed then the CGC server will not reset itself.

RETURN VALUE

0	Communications with the CGC finished OK;
-1	Communications failure.

SEE ALSO

SYS-CONFIG packet, `cgc_s_nop()`.

4.5.5 cgc_s_serial()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_serial(int *speed, int *databits,
       int *parity, int *stopbits);
```

PARAMETERS

`int *speed`	A pointer to a location where the baud rate to set is stored and where the actual baud rate as received from the CGC server is stored;
`int *databits`	A pointer to a location where the number of databits is stored;
`int *parity`	A pointer to a location where the type of parity is stored;
`int *stopbits`	A pointer to a location where the number of stopbits is stored.

DESCRIPTION

The `cgc_s_serial()` parameters allows one to change the serial port parameters on the BI-0501. The `speed` parameter is a pointer to a location where the baud rate at which to configure the port is located. After returning from `cgc_s_serial()` this location will contain the actual baud rate. Likewise are the `databits` parameter, `parity` parameter and `stopbits` parameters pointers to locations where respectively the number of databits, parity and stopbits to configure are found.

Only the user who is currently logged in as ‘root’ can actually change the serial port parameters. From all other users the change is ignored and the current settings are returned.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

SYS-SERIAL packet.

4.5.6 cgc_s_pwrnotify()

SYNOPSIS

```
#include <cgclib.h>
int cgc_s_pwrnotify(int *err, unsigned short udpport,
  int nrofports, struct cgc_pwrnotify *pwrnotify, void (*notify)(int));

int cgc_powerdown;
```

PARAMETERS

`int *err`	Pointer to a location where the error code returned from the CGC server is stored;
`unsigned short udpport`	An UDP port number in the range 1024 to 65535 on which to receive power failure notification;
`int nrofports`	The number of CAN ports for which to set CAN messages that are sent in case of a power failure;
`struct cgc_pwrnotify *pwrnotify`	Pointer to a location where the CAN messages can be found;
`void (*notify(int))`	Pointer to a function to call when a power failure has been detected.

DESCRIPTION

The `cgc_s_pwrnotify()` function installs a power down notification function which is called when a SYS-POWERDOWN message is received from the CGC server. The notification function is called with a single integer argument which will have the value 1 in case of a power down. If `cgc_s_pwrnotify()` is called with a NULL argument for the `notify` parameter, the currently installed function is cancelled. In this case no protocol interaction with the CGC server occurs.

The other parameters to the `cgc_s_pwrnotify()` function have the following meaning: the `err` parameter is a pointer to a location where the error code returned from the CGC server is stored, `udpport` is the UDP port to which the CGC server should send a power failure message, the `nrofports` parameter is the number of entries in the array pointed to by the `pwrnotify` parameter (i.e. the number of ports) for which to set the power failure notification. The pwrnotify parameter is a pointer to an array where the CAN port numbers and the CAN messages to be transmitted in case of a power failure are stored. Each array entry has the following definition:

```
struct cgc_pwrnotify {
  int             err;
  int             port;
  struct can_msg  msg;
};
```

The `err` field is used by `cgc_s_pwrnotify()` to store the return error code from the CGC server. The `port` field is the port number for which to set the CAN message and the `msg` field is the CAN message to send in case of a power failure on the BI-0501.

The global variable `cgc_powerdown` is set to the value 1 when a SYS-POWERDOWN message has been received. This is done regardless if a notification function has been installed or not.

RETURN VALUE

0	Power down function has been installed or removed.
-1	Power down function could not be installed or removed.

SEE ALSO

SYS-POWERDOWN packet.

4.6 CAN Port Management

The port management functions allow the caller to allocate CAN ports for message transmission and reception as well as configuring CAN ports. Most port management use a structure called `cgc_port` which has the following definition:

```
struct cgc_port {
  int    err;
  int    port;
  int    speed;
  char   *user;
};
```

The fields in this structure have the following meaning. The `err` field is used to store the error code as received from the CGC server, the `port` field names the port for which this structure is applicable, the `speed` field defines the CAN bus speed. The `user` field finally points to a NULL terminated character string with the name of the user to which the port is allocated. Not all fields are used by all port management functions which expect a `struct cgc_port` as a parameter, these fields are then set to 0. The function descriptions below mention which fields are used.

4.6.1 cgc_p_add()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_add(int nrofports, struct cgc_port *port);
```

PARAMETERS

`int *nrofports`	The number of ports in the array pointed to by the port parameter;
`struct cgc_port *port`	A pointer to an array of ports to add.

DESCRIPTION

The `cgc_p_add()` function allocates one or more ports to a user. The parameter `nrofports` is the number of entries in the `cgc_port` structure that the `port` parameter points to. The user must of course be logged in into the CGC server. On return the entries in the array that the `port` parameter points to will be updated with the error code returned from the CGC server for each port.

The `cgc_p_add()` function uses the `port` and `err` fields from the `cgc_port` structure.

RETURN VALUE

0	Communication with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-ADD packet, `cgc_p_delete()`.

4.6.2 cgc_p_delete()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_delete(int nrofports, struct cgc_port *port)
```

PARAMETERS

`int nrofports`	The number of ports in the array pointed to by the `port` parameter;
`struct cgc_port *port`	A pointer to an array of ports to delete.

DESCRIPTION

The `cgc_p_delete()` function removes the ports named in the array of ports pointed to by the `port` `cgc_port` structure. The `nrofports` parameter is the number of entries in this array. On return the array pointed to by the `port` parameter will have the error code field `err` filled in for each port for which deletion was requested.

The `cgc_p_delete()` function uses the `port` and `err` field from the `cgc_port` structure.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-DELETE packet, `cgc_p_add()`.

4.6.3 cgc_p_status()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_status(int nrofports, struct cgc_port *port)
```

PARAMETERS

`int nrofports`	The number of entries in the array of ports pointed to by the port parameter;
`struct cgc_users *port`	A pointer to an array of ports for which the status is requested.

DESCRIPTION

The `cgc_p_status()` function queries the CGC server about the status of the ports named in the array pointed to by the `port` pointer. The `nrofports` parameter is the number of entries in the `cgc_port` array. For each port the following fields in the `cgc_port` structure are filled in on return: the `err` field holds the error code returned for the port, the `port` field names the port for which the `cgc_port` entry is applicable, the `speed` field defines the CAN bus speed at which the port operates. The `user` field, finally, points to a NULL terminated character string naming the user to which the port is allocated. The storage for character string is allocated with `malloc()` and should be deallocated with `free()` before deallocating a `cgc_port` entry.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-STATUS packet, `cgc_p_status()`.

4.6.4 cgc_p_getspeed()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_getspeed(int nrofports, struct cgc_port *port);
```

PARAMETERS

`int nrofports`	The number of entries in the array pointed to by the `port` parameter;
`struct cgc_port *port`	A pointer to an array of `cgc_port` structures from which the bit rate is requested.

DESCRIPTION

The `cgc_p_getspeed()` function requests the bit rates at which the ports mentioned in the array pointed to by the `port` pointer operate. The number of entries in this array is given in the `nrofports` parameter. On return the following fields will be filled: the `err` field has the error code for the field, the `speed` field is the bit rate at which the CAN bus for this port operates.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-GETSPEED packet, `cgc_p_setspeed()`.

4.6.5 cgc_p_setspeed()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_setspeed(int nrofports, struct cgc_port *port);
```

PARAMETERS

`int nrofports`	The number of entries in the array of `cgc_port` structures the port parameter points to;
`struct cgc_port *port`	A pointer to an array of `cgc_port` structures.

DESCRIPTION

The `cgc_p_setspeed()` function changes the bit rate of one or more ports. The `nrofports` parameter is the number of entries in the array of `cgc_port` structures pointed to by the `port` parameter. For each port the bit rate is set depending on the `speed` field CAN port bit rate to set. On return the following fields will be filled in: the `err` field will contain the error code associated with this port, the `speed` field will contain the actual value of the CAN port bit rate.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-SETSPEED packet, `cgc_p_getspeed()`.

4.6.6 cgc_p_statistics()

SYNOPSIS

```
#include <cgclib.h>
int cgc_p_statistics(int port, struct cgc_portstats *stats);
```

PARAMETERS

`int port`	The number of the CAN port for which the statistics are requested;
`struct cgc_portstats *stats`	Pointer to a location where the CAN port's statistics are stored.

DESCRIPTION

The `cgc_p_statistics()` function requests the current transmission and reception counters from the CAN port named in the `port` parameter. The parameter `stats` should point to a location where the counters are stored. The `cgc_portstats` structure has the following definition:

```
struct cgc_portstats {
unsigned long    txmsg;
unsigned long    txbyte;
unsigned long    txfull;
unsigned long    txerr;
unsigned long    rxmsg;
unsigned long    rxbyte;
unsigned long    rxfull;
unsigned long    rxerr;
unsigned long    overrun;
unsigned long    erract;
unsigned long    errpasv;
unsigned long    busoff;
};
```

The fields have the same meaning as described in the definition of the PORT-STATISTICS packet.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

PORT-STATISTICS packet.

4.7 Watchdog Management

The watchdog functions allow setting messages which are sent on a CAN port at regular intervals or when the BI-0501 is interrupted from its regular work by a hardware watchdog reset. Sending one or more CAN messages at regular intervals is useful to implement time based polling of CAN devices without the need for message transmission from the users' host to the CGC server first.

4.7.1 cgc_w_add()

SYNOPSIS

```
#include <cgclib.h>
int cgc_w_add(int port, int *err, int msgnr, int cond,
       unsigned long tmo, struct can_msg *msg);
```

PARAMETERS

`int port`	The port number to set the watchdog message on;
`int *err`	A pointer to a location where the error code will be stored when `cgc_w_add()` returns;
`int msgnr`	The watchdog message number;
`int cond`	The condition for which the watchdog message is valid;
`unsigned long tmo`	The interval if the watchdog message should be sent at regular intervals or 0 if not used;
`struct can_msg *msg`	A pointer to a location where the CAN message is to be found.

DESCRIPTION

The `cgc_w_add()` function adds a CAN message to the watchdog named by the `msgnr` parameter. The `port` parameter names the port for which to add the watchdog. The `cond` parameter identifies the condition under which the CAN messages are to be sent. If the watchdog is a periodic watchdog, the `tmo` parameter is the number of microseconds in the watchdog period, otherwise it is ignored. The `msg` field holds the CAN message. The `err` parameter finally points to a location where the returned error code from the CGC server is stored.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

WATCHDOG-ADD packet, `cgc_w_delete()`.

4.7.2 cgc_w_delete()

SYNOPSIS

```
#include <cgclib.h>
int cgc_w_delete(int port, int msgnr, int *err);
```

PARAMETERS

`int port`	The port number from which to delete the watchdog;
`int msgnr`	The watchdog to delete;
`int *err`	A pointer to a location where the error code from the CGC server should be stored.

DESCRIPTION

The `cgc_w_delete()` function deletes the watchdog named in the parameter `msgnr` from the port mentioned in the parameter `port`. The location pointed to by the `err` parameter will be set to the error code returned from the CGC server.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

WATCHDOG-DELETE packet, `cgc_w_delete()`.

4.7.3 cgc_w_status()

SYNOPSIS

```
#include <cgclib.h>
int cgc_w_status(int port, int *err,
       int *nrofmsgs, struct cgc_wdstmsg **msg);
```

PARAMETERS

`int port`	The port number for which the watchdog status is requested;
`int *err`	A pointer to the location where the error code returned from the CGC server will be stored;
`int *nrofmsgs`	Pointer to a location where the number of watchdog messages returned in the `cgc_wdstmsg` array is stored.
`struct cgc_wdstmsg **msg`	Pointer to a location where the pointer to an array with `nrofmsgs` entries, each describing a watchdog message is stored.

DESCRIPTION

The `cgc_w_status()` function requests the status of all watchdogs set on the port specified in the `port` parameter. The location pointed to by the `err` parameter will be set to the error code returned from the CGC server. The `nrofmsgs` parameter is a pointer to a location where the number of watchdogs in the array pointed to by the `*msg` parameter will be stored. The `msg` parameter is a pointer to a location where the pointer to an array of `cgc_wdstmsg` structures will be stored. The `cgc_wdstmsg` structure has the following layout:

```
struct cgc_wdstmsg {
int             msgnr;
int             cond;
unsigned long   tmo;
struct can_msg  msg;
};
```

where the fields have the following meaning: the `msgnr` is the watchdog message number, the `cond` is the condition under which the watchdog will operate, the `tmo` field is the timeout interval in microseconds or 0 and `msg` is the CAN message to send if the watchdog is activated.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

WATCHDOG-STATUS packet, `cgc_w_add()`.

4.8 Filter Management

By using filters it is possible to block CAN messages received by the CGC server from being sent to the user's host application. This can help to minimize traffic to only those messages that the user is interested in. It is also possible to have a CAN message sent to the user's host because a message was transmitted on a port. This feature of the filters is useful when using watchdog messages. By installing a filter rule which matches the CAN message being sent by the watchdog, the user's application is notified automatically when this happens.

CAN message filtering consists of two parts: a message mask determines which parts (i.e. bits) of the message should be considered for matching and a filter message determines the bits that should be matched.

4.8.1 cgc_f_add()

SYNOPSIS

```
#include <cgclib.h>
int cgc_f_add(int port, int *err, int fltnr, int action,
  struct can_msg *msg, struct can_msg *msgmask);
```

PARAMETERS

`int port`	The port number to set the filter message on;
`int *err`	A pointer to a location where the error code will be stored when `cgc_f_add()` is finished;
`int fltnr`	The filter rule number;
`int action`	The action to take when a CAN message meets the filter condition;
`struct can_msg *msg`	A pointer to a location where the CAN message to match is stored;
`struct can_msg *msgmask`	A pointer to a location where the mask message to be applied before matching the CAN message is stored.

DESCRIPTION

The `cgc_f_add()` function adds a CAN message filter rule to the port named in the `port` parameter. The filter rule to add is named in the `fltnr` parameter. The `err` parameter points to the location where the error code returned from the CGC server is stored. The `action` parameter defines the action to take when a CAN message matches the filter rule. The parameters `can_msg` and `can_msgmsk` point to the CAN message with which to match and a mask message defining the significant bits in the message.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

FILTER-ADD packet, cgc_f_delete().

4.8.2 cgc_f_delete()

SYNOPSIS

```
#include <cgclib.h>
int cgc_f_delete(int port, int *err, int fltnr);
```

PARAMETERS

`int port`	The port number to delete the filter message from;
`int *err`	A pointer to a location where the error code will be stored when `cgc_f_delete()` returns;
`int fltnr`	The filter rule number to delete;

DESCRIPTION

The `cgc_f_delete()` function removes a CAN message filter rule from the port named in the `port` parameter. The filer rule to remove is named in the `fltnr` parameter. The `err` parameter points to the location where the error code returned from the CGC server is stored.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

FILTER-DELETE packet, `cgc_f_add()`.

4.8.3 cgc_f_status()

SYNOPSIS

```
#include <cgclib.h>
int cgc_f_status(int port, int *err, int *nrofflt,
       struct cgc_flt **flt);
```

PARAMETERS

`int port`	The port number for which the filter status is requested;
`int *err`	A pointer to a location where the error code will be stored when `cgc_f_status()` returns;
`int *nrofflt`	Pointer to a location where the number of filters in the array pointed to by `*flt` is stored;
`struct cgc_flt **flt`	A pointer to a location where the pointer to the array with filters defined for this port is stored.

DESCRIPTION

The `cgc_f_status()` function requests the status of all filters defined for the port named in the `port` parameter. The `err` parameter is a pointer to a location where the error code from the

CGC server is to be stored. The `nrofflt` parameter is a pointer to a location where the number of filters in the array pointed to by the `*flt` parameter is stored. The `flt` parameter is a pointer to a location where the array of filters defined for the port are stored. The `struct cgc_flt` has the following declaration:

```
struct cgc_flt {
   int                  fltnr;
   int                  action;
   int                  counter;
   struct can_msg       msg;
   struct can_msgmsk    msgmsk;
};
```

where the fields have the following meaning: `fltnr` is the number of the filter rule, `action` is the action to take when a CAN message matches the this filter rule, `counter` is the number of matches, and `msg` and `msgmsk` are the CAN message to match message to and the mask to apply before matching.

RETURN VALUE

0	Communications with the CGC server succeeded;
-1	Communications with the CGC server failed.

SEE ALSO

FILTER-STATUS packet, `cgc_f_add()`.

4.9 CAN Message Reception and Transmission

CAN message transmission and reception in the user's application is done using so called unsolicited request and response messages. This means that at any time the user can send messages and messages received by the CGC server from a CAN bus on the BI-0501 can be sent to the user's application at any time. There is however a single CGC host library function that deals with these messages: `cgc_c_canio()`. This function is passed a list of CAN ports and messages and for each message an indication if it should be transmitted or received. On return the lists are modified to contain the CAN messages recently received from the CGC server.

4.9.1 cgc_c_canio()

SYNOPSIS

```
#include <cgclib.h>
int cgc_c_canio(int datafd,
       int nrofmsgs,
       int *port, int *txrx, struct can_msg *msg);
```

PARAMETERS

`int datafd`	The file descriptor on which to send or receive CAN messages;
`int nrofmsgs`	The maximum number of messages in the arrays pointed to by the `port`, `txrx` and `msg` pointers;
`int *port`	A pointer to a list of ports of at most `nrofmsgs` long for which either messages should be transmitted or received;
`int *txrx`	Pointer to a list indicating if messages should be transmitted or received;
`struct can_msg *msg`	A pointer to a list of CAN messages to be transmitted or received.

DESCRIPTION

The `cgc_c_canio()` function receives and transmits CAN messages. The CGC server may send CAN messages at any time to the user's application. The CAN messages sent by the CGC server to the user's application can be either so called "send" or "receive" messages. Send messages are those that were sent by the CGC server on a particular CAN port, receive messages are those that were received from a particular CAN port.

On entry to `cgc_c_canio()` the parameters have the following meaning: `datafd` is the file descriptor referencing the data connection with the CGC server, `nrofmsgs` is the number of entries in the arrays pointed to by the `port`, `txrx` and `msg` pointers.

Before attempting to receive CAN messages from the CGC server, `cgc_c_canio()` first examines the `txrx` array and for each location that has the `CGC_CAN_TX` flag set, the port number and CAN message are collected and sent to the CGC server for transmission on the indicated CAN port. This process continues until a `txrx` array entry is found which has the flag `CGC_CAN_LAST` set too.

After transmitting all CAN messages, `cgc_c_canio()` attempts to read CAN messages sent by the CGC server to the user's host application. It reads CAN messages from the data connection for up to `nrofmsgs` CAN messages. Each CAN message is then put in the next free location in the `msg` array and the associated `port` entry is updated to reflect the CAN port for which the message is applicable. Each applicable entry in the `txrx` array has either the flag `CGC_CAN_TX` or `CGC_CAN_RX` set depending if the CAN message is a CAN-SEND message or a CAN-RECEIVE message. If no more CAN messages remain in the input queue, then the last entry in the `txrx` array will have the flag `CGC_CAN_LAST` set too.

The `cgc_c_canio()` function is normally expected to operate in a non-blocking fashion: if no messages are received, `cgc_c_canio()` returns immediately.

RETURN VALUE

0	Communications with the CGC server finished OK;
-1	Communications with the CGC server failed.

SEE ALSO

CAN-SEND packet, CAN-RECEIVE packet.