



BCC Communication Protocol v 3.10

© 2025 Robox SpA

The content of this document is prepared with the utmost care/diligence, and subjected to careful control.
Robox Spa, however, disclaims all liability, direct and indirect, to users and in general to any third party, for any inaccuracies, errors, omissions, damages (direct, indirect, consequential, punishable and sanctionable) resulting from the aforementioned contents

Table of contents

Parte I General 8

1 General index	8
2 Protocol conventions	8
3 Protocol specifications	9
4 Monitor specifications	11
5 Oscilloscope specifications	13
6 Transfer specifications	14

Parte II Messages 23

1 Alarm handling	23
Command for alarm history	23
Command for alarm stack	24
Get alarm history	24
Get alarm stack	26
Get enhanced alarm history	26
Query alarm history	30
Query alarm stack information	31
Query all alarm stack entries	31
Query single alarm stack entry	34
Reset alarm stack	37
Set user alarm	37
2 Date/Time handling	38
Get current date and time	38
Set current date and time	38
3 Debug and process handling	39
Add a breakpoint	40
Data format for process contents inspection	42
Data format for process debug context	47
Debug session watch-dog	49
Delete a breakpoint	49
Execute a process command	50
Execute a process debug command	51
Inspect contents of a process	52
List available remote process	54
List defined breakpoints	54
List object blocks	55
List OS attached functions	57
Query debug context for a process	59
Query information for a breakpoint	60
Query process flash information	62
Query process information	63
Query runtime status for process	67
Query status for a breakpoint	69
Query trace information for process	69
Start a debug session	70
Stop a debug session	71
4 Device handling	71

Begin an OOW session	72
End an OOW session	73
Execute a generic ASCII command	73
Query current mode	74
Query info for an OOW session	74
Query system information	75
Request a CMOS ram reset	81
Request a hardware reset	81
Request a software reset	81
Request device auto configuration	82
Resolve a /proc object	82
Set current mode	83
5 Field bus device handling	84
Field bus supported interface type IDs	85
Field bus entry data types	85
Force NMT status to EtherCAT Interface	86
Read an entry from CANopen Interface	87
Read an entry from EtherCAT (CoE) Interface	90
Read an entry from Local Interface	92
Read interface information	95
Read NMT status from CANopen Interface	96
Read NMT status from EtherCAT Interface	97
Read NMT status from Local Interface	98
Write an entry to CANopen Interface	99
Write an entry to EtherCAT (CoE) Interface	102
Write an entry to Local interface	104
Write an extended entry to CANopen Interface	106
Write an extended entry to EtherCAT (CoE) Interface	109
Write an extended entry to Local interface	112
Write interface information	114
Write NMT command to CANopen Interface	115
Write NMT command to Local Interface	116
6 Field bus handling	117
Query CANopen C402 information	118
Query CANopen EMCY message information	119
Query Robox CANopen channel diagnostic	120
Query Robox CANopen workstation diagnostic	121
Read a CANopen EMCY message	124
Read a CANopen object	125
Read a COE object	126
Read an EtherCAT NMT	128
Read data from Tx/Rx CANopen PDO	129
Read one or more CANopen NMT	130
Write a CANopen object	132
Write a COE object	133
Write an EtherCAT NMT	135
Write one or more CANopen NMT	136
7 Flash handling	136
Create a flash folder	137
Create and initialize flashes	138
Delete a flash folder	140
Delete files from a flash folder	140
Format a flash	141

Load a file to a flash folder	142
Manage flash volumes	146
Query contents from a flash folder	148
Query information of a file in a flash folder	150
Query information of a flash	153
Query information of a flash by folder	155
Query information of a flash folder	156
Query list of flashes	158
Query tree of flash folders	158
Rename a file in a flash folder	159
Save a file from a flash folder	159
Set attributes in a flash folder	162
8 General handling	164
9 I/O handling	167
Force input channel	168
Force input word 16bit	168
Force output channel	169
Force output word 16bit	170
Get input channel	171
Get input word 16bit	171
Get output channel	172
Get output word 16bit	173
Release all input channel	174
Release all output channel	174
Release input channel	175
Release input word 16bit	175
Release output channel	176
Release output word 16bit	176
Set output channel	177
Set output word 16bit	178
10 Ladder diagram handling	179
Cancel live changes	179
Confirm live changes	180
Load a ladder task to memory	180
Load live changes	181
Query ladder monitor status	182
Restart ladder monitor	183
Save a ladder task from memory	184
Start ladder monitor	184
Start live changes testing	187
Stop ladder monitor	188
Watchdog for ladder monitor	188
Watchdog for live changes	189
11 Monitor handling	189
Create a monitor	189
Destroy a monitor	191
Query a monitor status	191
Query list of monitors	192
Query monitor statistics	193
Quick monitor	193
Start a monitor	194
Stop a monitor	195
Watchdog for a monitor	196

Write a monitor	196
12 Network handling	197
Change a network user	198
Create a new network user	199
Delete a network user	199
Kill a network client	200
Network login	200
Network logout	200
Query a keep alive session information of a network client	201
Query list of network clients	202
Query list of network users	202
Query network information	203
Query network statistics	204
Start a keep alive session for a network client	205
Stop a keep alive session for a network client	206
13 Oscilloscope handling	206
Create an oscilloscope	207
Destroy a oscilloscope	208
Start an oscilloscope	209
Stop an oscilloscope	210
Query an oscilloscope status	211
Query list of oscilloscopes	212
Query oscilloscopes statistics	213
Watchdog for an oscilloscope	213
14 Protocol handling	214
Debug command	214
Ping answer	215
Ping command	215
15 Register handling	216
Get 16bit integer register	216
Get 32bit integer register	217
Get float register	218
Get real register	219
Get string register	220
Set 16bit integer register	221
Set 32bit integer register	222
Set float register	222
Set real register	223
Set string register	224
16 Report handling	225
Command for report	225
Command for system report	226
Get report contents	227
Get system report contents	228
Query report information	230
Query system report information	231
17 RPE handling	231
Begin a group authority session	232
Command for an object in a group authority session	233
End a group authority session	234
Get information for an axes group	235
Get positions for an axes group	237

Jog command for an object in a group authority session	238
List available axes groups	239
Load an object to a group authority session	241
Query status for an object in a group authority session	242
Resolve an axes group	244
Save an object from a group authority session	245
Update positions for an object's step inline point in a group authority session	246
Update positions for an object's point in a group authority session	247
Watchdog for group authority session	248
18 Variable handling	249
Enumerate variables	249
Force a variable	262
Force a variable (safe)	263
Read a variable	264
Read a variable (safe)	265
Register a dynamic variable	266
Release a variable	267
Release a variable (safe)	267
Release all variables	268
Release all variables (safe)	268
Un-register a dynamic variable	269
Un-register all dynamic variables	270
Write a variable	270
Write a variable (safe)	271
Parte III Network interfaces	272
1 Network interfaces	272
2 Ethernet network example	272
3 Modem network example	273
Parte IV Miscellaneous	273
1 Messages map	273
2 NACK error codes	283
3 Standard variables	289
Index	303

General

General index

Messages categories:

- [Alarm handling](#) ^[23]
- [Date/time handling](#) ^[38]
- [Debug and process handling](#) ^[39]
- [Device handling](#) ^[71]
- [Field bus device handling](#) ^[84]
- [Field bus handling](#) ^[117]
- [Flash handling](#) ^[136]
- [General messages](#) ^[164]
- [I/O handling](#) ^[167]
- [Ladder diagram handling](#) ^[179]
- [Monitor handling](#) ^[189]
- [Network handling](#) ^[197]
- [Oscilloscope handling](#) ^[206]
- [Protocol handling](#) ^[214]
- [Register handling](#) ^[216]
- [Report handling](#) ^[225]
- [RPE handling](#) ^[231]
- [Variable handling](#) ^[243]

Protocol specifications:

- [Protocol specification](#) ^[9]
- [Monitor specifications](#) ^[11]
- [Oscilloscope specifications](#) ^[13]
- [Transfer specifications](#) ^[14]
- [Messages map](#) ^[273]
- [NACK error codes](#) ^[283]
- [Standard variables](#) ^[289]

Other information:

- [Protocol conventions](#) ^[8]
- [Network interfaces](#) ^[272]

Protocol conventions

BCC communication protocol conventions, related to the manual itself.

Primitive data types

Here a list of the primitive data types used by this manual, referring a Intel386 hardware system.

Label	Size	Description
U8	1	Unsigned 8bit value
U16	2	Unsigned 16bit value
I16	2	Signed 16bit value
U32	4	Unsigned 32bit value
I32	4	Signed 32bit value
U64	8	Unsigned 64bit value

Label	Size	Description
I64	8	Signed 64bit value
FLT	4	Floating point (C float format, IEEE)
DBL	8	Floating point (C double format, IEEE)
B	1	Byte (or character), like U8
VAR	10	BCC3 variable
STRZ	?	ASCII string, zero terminated

When using a data type, the following symbol [n] means a linear array of N items of that data type: if N is omitted, means a dynamic array.

BCC3 variable type


A BCC3 variable is a 10 byte structure that identify one or more value of the same type on device: the variable can be used either to read value and to write value.

Type (2byte)	IdData, (8byte)
-----------------	--------------------

Where:

- Type identify a unique variable type
- IdData contains information for identifying a specific (or group of) variable of that type.

The IdData has no a specific format and will depend from Type value, that have the following mapping standard

From	To	Description
0x0000	0x0063	Reserved
0x0064	0x4FFF	Standard variables 
0x5000	0x7FFF	Reserved
0x8000	0x9FFF	User defined variables
0xA000	0xFFFF	Reserved

Protocol specifications

The BCC communication protocol has been designed in order to have an efficient communication protocol between Robox devices and Robox application and development tools.

Typical usage

This protocol is never used directly over a communication device but is carried by a lower level transport layer: normally we use our DLE/CRC16 transport layer.

This protocol is intended to be used as a point-to-point communication: multi-point communication is available, for example, by using in conjunction the BCC protocol over TCP/IP network connection.

Message format

The BCC message is divided into two parts:

- an [header](#)^[10], of a fixed length, that contain all routing information and message content;
- an optional [data area](#)^[11], whose size is contained in header, that contain message specific data.

Message header

The BCC message header is a fixed 9 byte structure with the following informations:

Type	Label	Description
U8	DST	Destination ID
U8	DCH	Destination channel ID
U8	SRC	Source ID
U8	SCH	Source channel ID
U16	MSG	Message code
U8	LEN	Data size (byte)
U8	PID	Protocol ID
U8	RES	Reserved (ex DLC)

Destination fields (DST,DCH)

These information area used to route the message correctly to the message target.

Source fields (SRC,SCH)

These information are used by the receiver to send answer or other messages back to the sender.

More about destination and source

DST is not intended as target device number, but is only used internally by the software to route correctly the message itself. By default, DST has the following meaning:

- 0x00 is used to route to a local connection (begin-point)
- 0x01 is used to route to the end-point connection
- 0x02 is used to route to a socket connection (begin-point)
- 0x03 is used to route special commands for network end-point
- 0x04 to 0x7F are reserved to Robox
- 0x80 to 0xFF are available for user applications

An example:

If you connect your PC to an RBXM via RS232 standard cable, the begin-point is the PC and the end-point is the RBXM.

When you want to send a message from PC to the RBXM, from a local connection (SRC=0, a begin-point) you send it to (DST=1, the end-point): channel ID are depending on the message type. Note that 0 and 2 area both begin-point.

When an answer come from the end-point, DST and SRC are swapped, so the message is routed back to the local connection (DST=0, begin-point) to the correct original sender. The same example could be applied to the socket connection too.

The DCH field normally is 0 if the message is a new request, or other value as requested by particular group or sequence of messages

Message code (MSG)

This code identify the meaning of the message, and can have a range from 0 to 0x7FFF: the 15th bit is reserved to be used as reply flag: is that message is not handled and the bit is on, a nack or error reply will occur, otherwise if bit is not set the message is simply lost. This bit should avoid some dead-loop condition.

To have more information about the message, see the messages specification.

Note: in message documentation, when the reply bit is required, the message code is printed as 'AS + <msgcode>' (AS indicate the 0x8000 bit), otherwise only <msgcode> is printed.

Data size (LEN)

Specify how many bytes of data are following the message header: if 0, the message has no data. The LEN field can have a value between 0 and 255.

Protocol ID (ID)

This field can be used to avoid wrong and confusing answer for some messages, especially to distinguish multiple answer from multiple query. Normally is used as a cyclic value from 0 to 255.

For each command message (especially type Ack/Nack), the answer PID must be the same of the request, otherwise the answer will be invalidated.

Reserved (RES)

This field (ex DLC) is now reserved for future use.

Please use 0xFF has default value in order to avoid troubles with existing communication devices.

Message data area

The message data area is optional: if not present, the LEN field in the message header is 0. otherwise it specify the exact size in bytes.

The message data area is always transmitted immediately after it's header: an incoming message should not be considered completed until it's full data area (if present) has been received.

Unused or reserved fields inside data area (and data size field) must be zero filled.

Monitor specifications

When you need to receive a group of data (variables) at fixed frequency from a connected device, the ideal solution is to program and play a realtime variable monitor. This will send you required data at request frequency, if possible.

Using a monitor

In order to use a monitor you have to perform the following steps:

1. Create the monitor
2. Start the monitor
3. Receive the data until desired or operation expired
4. Stop the monitor
5. Destroy the monitor

Each monitor is identified by a couple of 32bit values, called owner ID and monitor ID. You should assign a value to owner ID in order to distinguish you from other possible monitor clients (use a pseudo random 32bit value): the monitor ID is assigned by the connected device (don't make any assumption about the value of next monitor ID).

First of all create your monitor definition with [bccMonCreate](#)^[189] command: if successfully, it will give you the monitor ID. When you are ready to receive data from monitor, start it with the [bccMonStart](#)^[194] command, by indicating owner & monitor ID and required data frequency (hz): if successfully, the device will respond with the effective data frequency applied (not necessary what you required). If you don't agree with the data frequency, stop now the monitor with [bccMonStop](#)^[195] command.

While receiving data, at applied frequency (or less if device is overloaded), you have to periodically grant data transmission (command [bccMonWd](#)^[196]) by indicating granted time in [ms]: the device will transmit data until this time is expired or a new grant replace the old one. This means that the device will send data only if feel that client is connected and receive data, otherwise it will expire and stop the monitor automatically.

Caution:

- If you give a grant time (for example) of 5000ms, be sure to transmit the new grant before that time (for example at 4500ms) otherwise you risk that the monitor expire before new grant is acknowledged. If expired and yet valid, when you send a new [bccMonWd](#)^[196], the monitor is automatically restarted.
- If you create a monitor and lost it, when you create an other monitor and start it, if you use [bccMonWd](#)^[196] for all owner monitors, you will restart the lost monitor too. Then, Before create the new monitor, destroy all monitor for that owner or check monitor status with [bccMonStatus](#)^[197].

Incoming data from the monitor is carried by [bccData](#)^[165] message: see [bccMonStart](#)^[194] for more information.

When you don't want more data from monitor, or you don't receive data for a time (you can use the same grant time), you can stop the monitor with the [bccMonStop](#)^[195] command. If monitor is no more needed, can be destroyed with the [bccMonDestroy](#)^[191] command.

Each command you send for the monitor (bccMonStart, bccMonStop, bccMonDestroy) always need the owner and monitor ID: if these informations are not matched on the device, the operation will be denied.

Using multiple monitor

In the case you need multiple monitor at one time (for example required data does not fit in a single monitor), you can perform the following task (similar to single monitor usage):

1. Create all monitor definition and keep a list of all created monitor ID
2. Start separately each monitor (with monitor ID) or start all monitor of the owner (with special monitor ID 0xFFFFFFFF).
3. Receive the data for all monitors until desired or operation expired: grant periodically the transmission separately for each monitor (with monitor ID) or grant all owner monitor (with special monitor ID 0xFFFFFFFF)
4. Stop separately each monitor (with monitor ID) or start all monitor of the owner (with special monitor ID 0xFFFFFFFF)

5. Destroy separately each monitor (with monitor ID) or start all monitor of the owner (with special monitor ID 0xFFFFFFFF).

Oscilloscope specifications

When you need to receive a small group of data (variables) at high and fixed frequency from a connected device, the ideal solution is to program and play a real-time variable oscilloscope. This will send you required data at request frequency, if possible.

HINT: oscilloscope is used to get data from a minimum frequency of 100HZ up to a maximum of 1KHZ, where possible: for lower frequency, consider using a [monitor](#)^[11] instead.

Using an oscilloscope

In order to use an oscilloscope you have to perform the following steps:

1. Create the oscilloscope
2. Start the oscilloscope
3. Receive the data until desired or operation expired
4. Stop the oscilloscope
5. Destroy the oscilloscope

Each oscilloscope is identified by a couple of 32bit values, called owner ID and oscilloscope ID. You should assign a value to owner ID in order to distinguish you from other possible oscilloscope clients (use a pseudo random 32bit value): the oscilloscope ID is assigned by the connected device (don't make any assumption about the value of next ID).

First of all create your oscilloscope definition with [bccOscCreate](#)^[207] command: if successfully, it will give you the oscilloscope ID. When you are ready to receive data from oscilloscope, start it with the [bccOscStart](#)^[209] command, by indicating owner & oscilloscope ID and required data frequency (HZ): if successfully, the device will respond with the effective data frequency applied (not necessary what you required). If you don't agree with the data frequency, stop now the oscilloscope with [bccOscStop](#)^[210] command.

While receiving data, at applied frequency (or less if device is overloaded), you have to periodically grant data transmission (command [bccOscWd](#)^[213]) by indicating granted time in [ms]: the device will transmit data until this time is expired or a new grant replace the old one. This means that the device will sent data only if feel that client is connected and receive data, otherwise it will expire and stop the oscilloscope automatically.

Caution:

- If you give a grant time (for example) of 5000ms, be sure to transmit the new grant before that time (for example at 4500ms) otherwise you risk that the oscilloscope expire before new grant is acknowledged. If expired and yet valid, when you send a new [bccOscWd](#)^[213], the oscilloscope is automatically restarted.
- If you create an oscilloscope and lost it, when you create an other oscilloscope (with same owner) and start it, if you use [bccOscWd](#)^[213] for all owner oscilloscope, you will restart the lost oscilloscope too. Then, Before create the new oscilloscope destroy all oscilloscope for that owner or check oscilloscope status with [bccOscStatus](#)^[211].

Incoming data from the oscilloscope is carried by [bccData](#)^[165] message: see [bccOscStart](#)^[209] for more informations.

When you don't want more data from oscilloscope, or you don't receive data for a time (you can use the same grant time), you can stop the oscilloscope with the [bccOscStop](#)^[210] command. If oscilloscope is no more needed, can be destroyed with the [bccOscDestroy](#)^[208] command.

Each command you send for the oscilloscope (bccOscStart, bccOscStop, bccOscDestroy) always need the owner and oscilloscope ID: if these informations are not matched on the device, the operation will be denied.

Using multiple oscilloscopes

In the case you need multiple oscilloscope at one time (for example to use synchronized oscilloscopes), you can perform the following task (similar to single oscilloscope usage):

1. Create all oscilloscope definition and keep a list of all created oscilloscope ID
2. Start separately each oscilloscope (with oscilloscope ID) or start all oscilloscope of the owner (with special oscilloscope ID 0xFFFFFFFF).
3. Receive the data for all oscilloscopes until desired or operation expired: grant periodically the transmission separately for each oscilloscope (with oscilloscope ID) or grant all owner oscilloscope (with special oscilloscope ID 0xFFFFFFFF)
4. Stop separately each oscilloscope (with oscilloscope ID) or start all oscilloscope of the owner (with special oscilloscope ID 0xFFFFFFFF)
5. Destroy separately each oscilloscope (with oscilloscope ID) or start all oscilloscope of the owner (with special oscilloscope ID 0xFFFFFFFF).

Transfer specifications

The following transfer specifications are available in BCC/31 protocol:

- [download transfer sequence](#) ¹⁴
- [upload transfer sequence](#) ¹⁵
- [data save sequence](#) ¹⁶
- [data load sequence](#) ¹⁸
- [data exchange sequence](#) ¹⁹

Download transfer sequence

When you need to download some information from the connected device to local, you can use the download transfer sequence.

First send the request message, with optional request parameters (REQDATA).

On failure, a [bccNack](#) ¹⁶⁴ is received.

On success, a [bccAck](#) ¹⁶⁴ is received with following data (ACKDATA):

Offset	Type	Label	Description
0	U32	COUNT	No. of item that will be received (or 0xFFFFFFFF for continue stream terminated only by bccEndData ¹⁶⁵ message).

Note: the SCH value of this [bccAck](#) ¹⁶⁴ will be de DCH for all subsequent commands/message for this sequence.

Now, in order to receive item you have to periodically send [bccIBlock](#) ¹⁶⁶ message until end of data or operation is aborted. The message has the following data (IBDATA):

Offset	Type	Label	Description
0	U8	TX	No. of consecutive item that can be received (e.g. transmitted from the connected device)

If there are no data, the sequence is closed by a [bccNoData](#)^[166] message.

As authorized by the [bccIBlock](#)^[166] total TX count, the connected device will transmit items (data structure defined as ITEMDATA) with [bccData](#)^[165] and [bccEndData](#)^[165] message. Notice that transporting data with [bccEndData](#)^[165] is not mandatory: it could be used only as end of sequence marker.

Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.

Notes:

- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value) or can always send pid = 0 meaning that no sequence check is required.
- [bccEndData](#)^[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time with sending [bccAbort](#)^[165] command to the connected device.
- Data structure REQDATA, ACKDATA, IBDATA and ITEMDATA are defined in the request message documentation, but always have the minimum structure declared here.

Upload transfer sequence

When you need to upload some information from local to the connected device, you can use the upload transfer sequence.

First send the request message, with following parameters (REQDATA):

Offset	Type	Label	Description
0	U32	COUNT	No. of item that will be transmitted (or 0xFFFFFFFF for continuous stream terminated only by bccEndData ^[165] message).

On failure, a [bccNack](#)^[164] is received.

On success, a [bccAck](#)^[164] is received with optional data (ACKDATA).

Notes:

- the SCH value of this [bccAck](#)^[164] will be de DCH for all subsequent commands/message for this sequence.

Periodically the connected device will send [bccIBlock](#)^[166] in order to authorize you to send a number of consecutive items: the message has the following data (IBDATA):

Offset	Type	Label	Description
0	U8	TX	No. of consecutive items that can be transmitted (e.g. received from connected device)

If you have no data to send, send a [bccNoData](#)^[166] and close the sequence.

As authorized by [bccIBlock](#)^[166] total TX count, send items (data structure defined as ITEMDATA) with [bccData](#)^[165] and [bccEndData](#)^[165] message. Notice that transporting data with [bccEndData](#)^[165] is not mandatory: it could be used only as end of sequence marker.

Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.

Notes:

- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value): if pid is always 0, it mean that no sequence verification is required.
- [bccEndData](#)^[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time by sending a [bccAbort](#)^[165] command to the connected device.
- Data structure REQDATA, ACKDATA, IBDATA and ITEMDATA are defined in the request message documentation, but always have the minimum structure declared here.

Data save sequence

When you need to save a block of binary data from the connected device to local, you can use the data save sequence.

First send the request message, with the request parameters (REQDATA):

Offset	Type	Label	Description
0	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data.
1	U8[15]		(riservati)

On failure, a [bccNack](#)^[164] is received.

On success, a [bccAck](#)^[164] is received with following data (ACKDATA):

Offset	Type	Label	Description
0	U32	SIZE	Size of data to be saved [bytes]
4	U8[12]		(riservati)

Notes:

- the SCH value of this [bccAck](#)^[164] will be de DCH for all subsequent commands/message for this sequence.

Now, in order to receive item you have to periodically send [bccIBlock](#)^[166] message until end of data or operation is aborted. The message has the following data:

Offset	Type	Label	Description
0	U8	TX	No. of consecutive item that can be received (e.g. transmitted from the connected device)
15	U8[15]		(riservati)

If there are no data, the sequence is closed by a [bccNoData](#)^[166] message.

As authorized by the [bccIBlock](#)^[166] total TX count, the connected device will transmit items with [bccData](#)^[165] and [bccEndData](#)^[165] message. Notice that transporting data with [bccEndData](#)^[165] is not mandatory: it could be used only as end of sequence marker. Items have the following structure:

Offset	Type	Label	Description
0	U32	OFFSET	Item data offset
4	B[]	DATA	Item data (max 251 bytes)

Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.

After you receive [bccEndData](#)^[165] (or [bccNoData](#)^[166]) message you should send a [bccCompleted](#)^[167] message (if all things are right) to report that the file transfer has been completed: if you need more time before sending this message, please periodically send a [bccWait](#)^[167] message to inform the connected device about the delay. The [bccCompleted](#)^[167] data are:

Offset	Type	Label	Description
0	U32	SIZE	Size of saved data[bytes]
4	U32	COUNT	No. of total binary packet received

At any time the device can decide to abort the transfer. In this case you will receive the notification via the [bccAborted](#)^[165] message.

Notes:

- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value) or can always send pid = 0 meaning that no sequence check is required.
- [bccEndData](#)^[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time by sending a [bccAbort](#)^[165] command to the connected device.
- Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.
- Data structure REQDATA and ACKDATA are defined in the request message documentation, but always have the minimum structure declared here.

- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)^[167] messages: when the limit is reached, the operation will expire.

Data load sequence

When you need to load a block of binary data from local to the connected device, you can use the data load sequence.

First send the request message, with at least the following data (REQDATA):

Offset	Type	Label	Description
0	U32	SIZE	Size of data to be loaded [bytes]
4	U8[12]	-	(reserved)

On failure, a [bccNack](#)^[164] is received.

On success, a [bccAck](#)^[164] is received with at least the following data (ACKDATA):

Offset	Type	Label	Description
0	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data.
1	U8[15]	-	(reserved)

Note: the SCH value of this [bccAck](#)^[164] will be de DCH for all subsequent commands/message for this sequence.

Periodically the connected device will send [bccIBlock](#)^[166] in order to authorize you to send a number of consecutive items: the message has the following data:

Offset	Type	Label	Description
0	U8	TX	No. of consecutive item that can be transmitted (e.g. received from the connected device)

If you have no data to send, send a [bccNoData](#)^[166] and close the sequence.

As authorized by [bccIBlock](#)^[166] total TX count, send items with [bccData](#)^[165] and [bccEndData](#)^[165] message. Notice that transporting data with [bccEndData](#)^[165] is not mandatory: it could be used only as end of sequence marker. Items have the following structure:

Offset	Type	Label	Description
0	U32	OFFSET	Item data offset
4	B[]	DATA	Item data (max 251 bytes)

After you send the [bccEndData](#)^[165] (or [bccNoData](#)^[166]) message you should wait the [bccCompleted](#)^[167] message that report what the device has received all data, with the following format:

Offset	Type	Label	Description
0	U32	SIZE	Total size of loaded data [bytes]
4	U32	COUNT	No. of total binary packet received

Notes:

- if this message is not received in an amount of time, you can consider the transfer aborted (not confirmed). If the remote device need more time in order to complete the operation, it can periodically send the [bccWait](#)^[167] message, until the operation completes.
- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value): if pid is always 0, it mean that no sequence verification is required.
- [bccEndData](#)^[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time by sending a [bccAbort](#)^[165] command to the connected device, if you are not already in abort condition).
- Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.
- Data structure REQDATA and ACKDATA are defined in the request message documentation, but always have the minimum structure declared here.
- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)^[167] messages: when the limit is reached, the operation will expire.

Data exchange sequence

The data exchange sequence combine in a single operation the following activities:

- [loading a block of data](#)^[19] (QUERYDATA)
- [waiting a block of data ready](#)^[21] (ANSWERDATA)
- [saving a block of data](#)^[21] (ANSWERDATA)

It is a sort of combination of a data load sequence followed by a data save sequence, but in the same operation context.

Loading a block of data

This phase will send the data block (QUERYDATA) to the device.

First send the request message, with at least the following data (REQDATA):

Offset	Type	Label	Description
0	U32	SIZE	Size of data to be loaded [bytes]
4	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data, that will be used in phase 3.
5	U8[11]	-	(reserved)

On failure, a [bccNack](#)_[164] is received.

On success, a [bccAck](#)_[164] is received with at least the following data (ACKDATA):

Offset	Type	Label	Description
0	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data.
1	U8[15]	-	(reserved)

Note: the SCH value of this [bccAck](#)_[164] will be de DCH for all subsequent commands/message for this sequence, for all phases.

Periodically the connected device will send [bccIBlock](#)_[166] in order to authorize you to send a number of consecutive items: the message has the following data:

Offset	Type	Label	Description
0	U8	TX	No. of consecutive item that can be transmitted (e.g. received from the connected device)

If you have no data to send, send a [bccNoData](#)_[166] and close the sequence.

As authorized by [bccIBlock](#)_[166] total TX count, send items with [bccData](#)_[165] and [bccEndData](#)_[165] message. Notice that transporting data with [bccEndData](#)_[165] is not mandatory: it could be used only as end of sequence marker. Items have the following structure:

Offset	Type	Label	Description
0	U32	OFFSET	Item data offset
4	B[]	DATA	Item data (max 251 bytes)

After you send the [bccEndData](#)_[165] (or [bccNoData](#)_[166]) message you should wait the [bccCompleted](#)_[167] message that report what the device has received all data, with the following format:

Offset	Type	Label	Description
0	U32	SIZE	Total size of loaded data [bytes]
4	U32	COUNT	No. of total binary packet received

If the [bccCompleted](#)_[167] message is not received in an amount of time, you can consider the transfer aborted (not confirmed).

If the remote device need more time in order to complete the operation, it can periodically send the [bccWait](#)_[167] message as needed..

When the [bccCompleted](#)_[167] is received, the operation will go to phase 2.

Notes:

- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value).
- [bccEndData](#)^[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time by sending a [bccAbort](#)^[165] command to the connected device, if you are not already in abort condition).
- Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.
- Data structure REQDATA and ACKDATA are defined in the request message documentation, but always have the minimum structure declared here.
- Data structure QUERYDATA is fully defined in the request message documentation.
- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)^[167] messages: when the limit is reached, the operation will expire.

Waiting a block of data

This phase will wait that the data block (ANSWERDATA) is ready from the device.

For an amount of time the sequence will wait for the [bccReady](#)^[167] message, with the following data (READYDATA):

Offset	Type	Label	Description
0	U32	SIZE	Size of data to be saved [bytes]
4	U8[12]		(riservati)

When the [bccReady](#)^[167] is received, the operation will go to phase 3.

If the remote device need more time in order to prepare the answer data it can periodically send the [bccWait](#)^[167] message as needed.

Notes:

- [bccReady](#)^[167] will have pid = 0.
- The sequence can be aborted at any time by sending a [bccAbort](#)^[165] command to the connected device, if you are not already in abort condition.
- Any time the connected device abort the sequence, you will receive a [bccAborted](#)^[165] message.
- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)^[167] messages: when the limit is reached, the operation will expire.
- Data structure READYDATA is defined in the request message documentation, but always have the minimum structure declared here.
- Data structure ANSWERDATA is fully defined in the request message documentation.

Saving a block of data

This phase will receive the data block (ANSWERDATA) from the device.

In order to receive data you have to periodically send [bccIBlock](#)^[166] message until end of data or operation is aborted. The message has the following data:

Offset	Type	Label	Description
0	U8	TX	No. of consecutive item that can be received (e.g. transmitted from the connected device)
15	U8[15]		(riservati)

If there are no data, the sequence is closed by a [bccNoData](#)_[166] message.

As authorized by the [bccBlock](#)_[166] total TX count, the connected device will transmit items with [bccData](#)_[165] and [bccEndData](#)_[165] message. Notice that transporting data with [bccEndData](#)_[165] is not mandatory: it could be used only as end of sequence marker. Items have the following structure:

Offset	Type	Label	Description
0	U32	OFFSET	Item data offset
4	B[]	DATA	Item data (max 251 bytes)

Any time the connected device abort the sequence, you will receive a [bccAborted](#)_[165] message.

After you receive [bccEndData](#)_[165] (or [bccNoData](#)_[166]) message you should send a [bccCompleted](#)_[167] message (if all things are right) to report that the file transfer has been completed: if you need more time before sending this message, please periodically send a [bccWait](#)_[167] message to inform the connected device about the delay. The [bccCompleted](#)_[167] data are:

Offset	Type	Label	Description
0	U32	SIZE	Size of saved data[bytes]
4	U32	COUNT	No. of total binary packet received

When the [bccCompleted](#)_[167] is sent, the operation is completed.

At any time the device can decide to abort the transfer. In this case you will receive the notification via the [bccAborted](#)_[165] message.

Notes:

- [bccData](#)_[165] (or [bccEndData](#)_[165]) messages will begin with pid = 0 and will be incremented by 1 at each message (255 + 1 will restart from 0 value).
- [bccEndData](#)_[165] contain last item (with or without data) and after it the sequence is completed.
- The sequence can be aborted at any time by sending a [bccAbort](#)_[165] command to the connected device.
- Any time the connected device abort the sequence, you will receive a [bccAborted](#)_[165] message.
- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)_[167] messages: when the limit is reached, the operation will expire.
- Data structure ANSWERDATA is fully defined in the request message documentation.

Messages

Alarm handling

These messages are used getting and setting alarms to a connected device.

Alarm history:

- [bccAlarmHInfo](#)^[30], query alarm history
- [bccAlarmHList](#)^[24], get alarm history
- [bccAlarmHListE](#)^[26], get enhanced alarm history
- [bccAlarmHCmd](#)^[23], command for alarm history

Miscellaneous:

- [bccGetAlarm](#)^[26], get alarm stack
- [bccSetAlarm](#)^[37], set user alarm
- [bccResetAlarm](#)^[37], reset alarm stack

Alarm stack:

- [bccAlarmSInfo](#)^[31], query alarm stack information
- [bccAlarmSGet](#)^[34], query single alarm stack entry
- [bccAlarmSList](#)^[31], query all alarm stack entries
- [bccAlarmSCmd](#)^[24], command for alarm stack

Command for alarm history

Code:	AS + 517
Symbolic:	bccAlarmHCmd

This command will execute an alarm history specific command on the connected device. Request has following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Command flags: 0x00000001 Clear alarm history 0x00000002 Set new history size
4	U32	SIZE	New history size (no. of message), if enabled

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Flags

NACK code	Description	Extra data
		2=Size
nackReadOnly	Alarm history is read-only and cannot be modified.	

Command for alarm stack

Code:	AS + 522
Symbolic:	bccAlarmSCmd

This command will execute an alarm stack specific command on the connected device. Request has following parameters:

Offset	Type	Label	Description
0	U16	CMD	Command code: 1=Clear all alarms 2=Clear stack position (ALIX) 3=Clear specific alarm (ALCODE, ALAXIS)
2	U32	ALIX	Alarm index
6	U16	ALCODE	Alarm code (0=all)
8	U16	ALAXIS	Alarm axis ID (0=all)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Command 2=Alarm index 3=Alarm code 4=Alarm axis ID

Get alarm history

Code:	AS + 516
-------	-----------------

Symbolic	bccAlarmHList
----------	----------------------

This command will get complete (or partial) alarm history from the remote device.: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FROMID	First alarm history ID required
4	U32	TOID	Last alarm history ID required

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U32	ID	Alarm history ID
4	U8	HH	Time (hour) of the item
5	U8	MM	Time (minute) of the item
6	U8	SS	Time (second) of the item
7	U8	DD	Date (day) of the item
8	U8	MO	Data (month) of the item
9	U8	YY	Data (year - 2000) of the item
10	U32	CODE	Alarm code
14	U32	-	Reserved
18	U32	-	Reserved
22	U32	-	Reserved
26	STRZ	TEXT	Alarm text description (optional)

In case of RTE firmware (see [bccSysInfo](#)^[75], FIRMWTYPE=8), the CODE field contains the following data:

Offset	Type	Label	Description
0	U16	ALCODE	RTE alarm code (0=none)
2	U16	ALAXIS	RTE alarm axis ID (0=none)

As general, you can get items with the following order:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

When using RTE version 33.16.x (or lower), you should consider two different cases:

1. NEWID is greater/equal than OLDID.
2. NEWID is less than OLDID.

In case 1, you can get items with a single request, as:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

In case 2, you must get items with two different request, as:

- from OLDID to 0xFFFFFFFF and from 0 to NEWID for native order.
- from NEWID to 0 and from 0xFFFFFFFF to OLDID for reversed order.

Get alarm stack

Code:	AS + 500
Symbolic:	bccGetAlarm

This command will get the complete alarm stack (8 alarm codes). Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U16	ALARM0	Alarm code 0
2	U16	ALARM1	Alarm code 1
4	U16	ALARM2	Alarm code 2
6	U16	ALARM3	Alarm code 3
8	U16	ALARM4	Alarm code 4
10	U16	ALARM5	Alarm code 5
12	U16	ALARM6	Alarm code 6
14	U16	ALARM7	Alarm code 7

On failure, a [bccNack](#)₁₆₄ is received.

For more informations about alarm, see your hardware documentation.

NOTE: this command is not supported by RTE (see [bccAlarmSList](#)₃₁).

Get enhanced alarm history

Code:	AS + 524
Symbolic:	bccAlarmHListE

This command will get complete (or partial) enhanced alarm history from the remote device: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Query flags: 0x00000001 Query for extra parameters (ALPx)(#1) 0x00000002 Query for original alarm text 0x00000004 Disable query for alarm text 0x00000008 Query using extended alarm format 0x00000010 Query source text parameter index (#2) 0x00000020 Query source text in neutral language (#3) 0x00000040 Disable common prefixes for alarm text
4	U32	FROMID	First alarm history ID required
8	U32	TOID	Last alarm history ID required

(#1) When the extended format is required, parameters are always queried.

(#2) Query of the source text parameter index is available only with the extended format and apply both to system and user text parameters.

(#3) Query source text in neutral language is available only if query for original alarm text (0x2) is active.

ACKDATA structure is the following:

Offset	Type	Label	Description
0	U32	COUNT	Transfer items count (see download transfer sequence ^[14])
4	U32	HSIZE	History size (no. of storable alarms)
8	U32	HID	History content ID

If initial request fails, [bccNack](#)^[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

ITEMDATA structure is the following (in case of normal alarm format):

Offset	Type	Label	Description
0	U32	ID	Alarm history ID
4	U8	HH	Time (hour) of the item
5	U8	MM	Time (minute) of the item
6	U8	SS	Time (second) of the item
7	U8	DD	Date (day) of the item
8	U8	MO	Data (month) of the item
9	U8	YY	Data (year - 2000) of the item
10	U16	ALCODE	Alarm code (0=none)
12	U16	ALAXIS	Alarm axis ID (0=none)
14	U8[2]	-	(reserved)
...	DBL	ALP1	Alarm extra parameter 1 (#1)
...	DBL	ALP2	Alarm extra parameter 2 (#1)
...	DBL	ALP3	Alarm extra parameter 3 (#1)
...	STRZ	TEXT	Alarm text (#2)

(#1) Fields present only if FLAGS parameter contains 0x00000001 value.

(#2) Fields present only if FLAGS parameter does not contains 0x00000004 value.

ITEMDATA structure is the following (in case of extended alarm format):

Offset	Type	Label	Description
0	U32	ID	Alarm history ID
4	U8	HH	Time (hour) of the item

Offset	Type	Label	Description
5	U8	MM	Time (minute) of the item
6	U8	SS	Time (second) of the item
7	U8	DD	Date (day) of the item
8	U8	MO	Data (month) of the item
9	U8	YY	Data (year - 2000) of the item
10	U16	ALCODE	Alarm code (0=none)
12	U16	ALAXIS	Alarm axis ID (0=none)
14	U8	NITEMS	N. of alarm items
15	..	ITEM0	Alarm item 0
...	...	ITEM1	Alarm item 1
...	...	ITEMn	Alarm item N

where every ITEMx structure is the following:

Offset	Type	Label	Description
+0	U8	TYPE	Item type: 1 = Signed 8bit value (I8) 2 = Unsigned 8bit value (U8) 3 = Signed 16bit value (I16) 4 = Unsigned 16bit value (U16) 5 = Signed 32bit value (I32) 6 = Unsigned 32bit value (U32) 7 = Signed 64bit value (I64) 8 = Unsigned 64bit value (U64) 9 = Float value (FLT) 10 = Double value (DBL)

Offset	Type	Label	Description
			11 = String value (STRZ) 12 = Alarm text (STRZ) 13 = System source text parameter (I32) 14 = User source text parameter index (I32)
+1	...	DATA	Item data (content and length according type)

As general, you can get items with the following order:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

When using RTE version 33.16.x (or lower), you should consider two different cases:

1. NEWID is greater/equal than OLDID.
2. NEWID is less than OLDID.

In case 1, you can get items with a single request, as:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

In case 2, you must get items with two different request, as:

- from OLDID to 0xFFFFFFFF and from 0 to NEWID for native order.
- from NEWID to 0 and from 0xFFFFFFFF to OLDID for reversed order.

Query alarm history

Code:	AS + 515
Symbolic:	bccAlarmHInfo

This command will query information about alarm history on the remote device. Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	OLDID	Oldest alarm history ID
4	U32	NEWID	Newst alarm history ID
8	U32	HCNT	No. of alarm history between OLDID and NEWID (all included)

Offset	Type	Label	Description
12	U32	HSIZE	History size (no. of storable alarms)
16	U32	HID	History content ID

On failure, a [bccNack](#)₁₆₄ is received.

Alarm history report handling is based on the following statements:

- Item ID are allocated progressively, with an exception: at value 0xFFFFFFFF, next valid ID will be 0.
- The no. of items from OLDID and NEWID will be calculated as: if NEWID great or equal than OLDID maximum no. is (NEWID - OLDID + 1), otherwise is ((0xFFFFFFFF - OLDID + 1) + NEWID + 1).

Query alarm stack information

Code:	AS + 519
Symbolic:	bccAlarmSInfo

This command will query information about alarm stack from the connected device. Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	ASCAP	Alarm stack capacity (n. of alarms)
4	U32	ASCNT	Alarm count in stack
8	U32	CID	Alarm stack content ID

On failure, a [bccNack](#)₁₆₄ is received.

Query all alarm stack entries

Code:	AS + 521
Symbolic:	bccAlarmSList

This command will query the complete alarm stack from the connected device: this is a standard [download transfer sequence](#)₁₄.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Query flags:

Offset	Type	Label	Description
			0x00000001 Query for extra parameters (ALPx)(#2) 0x00000002 Query for original alarm text only 0x00000004 Disable query for alarm text 0x00000008 Enable LCID/CID verification 0x00000010 Query using extended alarm format 0x00000020 Query source text parameter index (#3) 0x00000040 Query source text in neutral language (#4) 0x00000080 Disable common prefixes for alarm text
4	U32	LCID	Local alarm stack content ID (#1)

(#1) Information required only when LCID/CID verification is enabled.

(#2) When the extended format is required, parameters are always queried.

(#3) Query of the source text parameter index is available only with the extended format and apply both to system and user text parameters.

(#4) Query source text in neutral language is available only if query for original alarm text (0x2) is active.

ACKDATA structure is the following:

Offset	Type	Label	Description
0	U32	COUNT	Transfer items count (see download transfer sequence ^[14])
4	U32	ASCAP	Alarm stack capacity (n. of alarms)
8	U32	CID	Alarm stack content ID

If initial request fails, [bccNack](#)^[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

NACK code	Description	Extra data
nackSameData	The CID value is the same as LCID, so the query is not needed (only when verification is required).	

ITEMDATA structure is the following (in case of normal alarm format):

Offset	Type	Label	Description
0	U32	ALIX	Alarm index (1-ASCAP)
4	U16	ALCODE	Alarm code (0=none)
6	U16	ALAXIS	Alarm axis ID (0=none)
8	DBL	ALP1	Alarm extra parameter 1
16	DBL	ALP2	Alarm extra parameter 2
24	DBL	ALP3	Alarm extra parameter 3
32	U8[8]	-	Reserved
40	STRZ	TEXT	Alarm text

ITEMDATA structure is the following (in case of extended alarm format):

Offset	Type	Label	Description
0	U32	ALIX	Alarm index (1-ASCAP)
4	U16	ALCODE	Alarm code (0=none)
6	U16	ALAXIS	Alarm axis ID (0=none)
8	U8	NITEMS	N. of alarm items
9	..	ITEM0	Alarm item 0
...	...	ITEM1	Alarm item 1
...	...	ITEMn	Alarm item N

where every ITEMx structure is the following:

Offset	Type	Label	Description
+0	U8	TYPE	Item type:

Offset	Type	Label	Description
			1 = Signed 8bit value (I8) 2 = Unsigned 8bit value (U8) 3 = Signed 16bit value (I16) 4 = Unsigned 16bit value (U16) 5 = Signed 32bit value (I32) 6 = Unsigned 32bit value (U32) 7 = Signed 64bit value (I64) 8 = Unsigned 64bit value (U64) 9 = Float value (FLT) 10 = Double value (DBL) 11 = String value (STRZ) 12 = Alarm text (STRZ) 13 = System source text parameter (I32) 14 = User source text parameter index (I32)
+1	...	DATA	Item data (content and length according type)

Query single alarm stack entry

Code:	AS + 520
Symbolic:	bccAlarmSGet

This command will query a single alarm stack entry from the connected device. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Query flags: 0x00000001 Query for extra parameters (ALPx)(#1)

Offset	Type	Label	Description
			0x00000002 Query for original alarm text only 0x00000004 Disable query for alarm text 0x00000008 Query using extended alarm format 0x00000010 Query source text parameter index (#2) 0x00000020 Query source text in neutral language (#3) 0x00000040 Disable common prefixes for alarm text
4	U32	ALIX	Alarm index (1-ASCAP, see bccAlarmSInfo _[31])

(#1) When the extended format is required, parameters are always queried.

(#2) Query of the source text parameter index is available only with the extended format and apply both to system and user text parameters.

(#3) Query source text in neutral language is available only if query for original alarm text (0x2) is active.

On success, a [bccAck](#)_[164] is received with following data (in case of normal alarm format):

Offset	Type	Label	Description
0	U32	ALIX	Alarm index (1-ASCAP)
4	U32	CODE	Alarm code
8	DBL	ALP1	Alarm extra parameter 1
16	DBL	ALP2	Alarm extra parameter 2
24	DBL	ALP3	Alarm extra parameter 3
32	U8[8]	-	Reserved
40	STRZ	TEXT	Alarm text

or with the following structure (in case of extended alarm format):

Offset	Type	Label	Description
0	U32	ALIX	Alarm index (1-ASCAP)
4	U32	CODE	Alarm code

Offset	Type	Label	Description
			(0=none)
8	U8	NITEMS	N. of alarm items
9	..	ITEM0	Alarm item 0
...	...	ITEM1	Alarm item 1
...	...	ITEMn	Alarm item N

where every ITEMx structure is the following:

Offset	Type	Label	Description
+0	U8	TYPE	Item type: 1 = Signed 8bit value (I8) 2 = Unsigned 8bit value (U8) 3 = Signed 16bit value (I16) 4 = Unsigned 16bit value (U16) 5 = Signed 32bit value (I32) 6 = Unsigned 32bit value (U32) 7 = Signed 64bit value (I64) 8 = Unsigned 64bit value (U64) 9 = Float value (FLT) 10 = Double value (DBL) 11 = String value (STRZ) 12 = Alarm text (STRZ) 13 = System source text parameter index (I32) 14 = User source text parameter index (I32)
+1	...	DATA	Item data (content and length according type)

In case of RTE firmware (see [bccSysInfo](#)^[75], FIRMWTYPE=8), the CODE field contains the following data:

Offset	Type	Label	Description
0	U16	ALCODE	RTE alarm code (0=none)
2	U16	ALAXIS	RTE alarm axis ID (0=none)

On failure, a [bccNack](#)₁₆₄ is received.

Reset alarm stack

Code:	AS + 502
Symbolic:	bccResetAlarm

This command will try to reset alarms on the device.

On success, a [bccAck](#)₁₆₄ is received: it does not mean that alarms have been really reset, but to be sure you will have to check again with [bccGetAlarm](#)₂₆ command.

On failure, a [bccNack](#)₁₆₄ is received.

NOTE: this command is not supported by RTE (see [bccAlarmSCmd](#)₂₄).

Set user alarm

Code:	AS + 501
Symbolic:	bccSetAlarm

This command will set the user alarm on the device, given a code. Request parameters are the following:

Offset	Type	Label	Description
0	U16	ALARM	User alarm code (see your hardware documentation for valid values)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal alarm code	

WARNING: on some device, setting certain alarm code, can cause "missing power" condition.

NOTE: this command is not supported by RTE.

Date/Time handling

These messages are used getting and setting date and time to a connected device.

- [bccGetDateTime](#)^[38], get current date and time
- [bccSetDateTime](#)^[38], set current date and time

Get current date and time

Code:	AS + 503
Symbolic:	bccGetDateTime

This command will get current device date and time. Request has no parameters.

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U8	HOUR	Hour (0-23)
1	U8	MIN	Minutes (0-59)
2	U8	SEC	Seconds (0-59)
3	U8	DAY	Current day (1-31)
4	U8	MONTH	Current month (1-12)
5	U16	YEAR	Current 4 digit year

On failure, a [bccNack](#)^[164] is received.

Set current date and time

Code:	AS + 504
Symbolic:	bccSetDateTime

This command will set current device date and time. Request parameters are the following:

Offset	Type	Label	Description
0	U8	HOUR	Hour (0-23)
1	U8	MIN	Minutes (0-59)
2	U8	SEC	Seconds (0-59)
3	U8	DAY	Current day (1-31)
4	U8	MONTH	Current month (1-12)
5	U16	YEAR	Current 4 digit year
7	U8	FLAGS	Settable field bits:

Offset	Type	Label	Description
			0x01 = HOUR field settable 0x02 = MIN field settable 0x04 = SEC field settable 0x08 = DAY field settable 0x10 = MONTH field settable 0x20 = YEAR field settable

Notes:

- Remember to set FLAGS for fields you want to set, otherwise they will not set. At least one bit must be set in FLAGS, otherwise the command will fail.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal 1=Flags 2=Date/time

Debug and process handling

These messages are used to handle debug and process activities for a connected device.

General:

- [bccDebugStart](#)^[70], Start a debug session
- [bccDebugStop](#)^[71], Stop a debug session
- [bccDebugWd](#)^[49], Debug session watch-dog

Breakpoint specific:

- [bccBreakpAdd](#)^[40], Add a breakpoint
- [bccBreakpDel](#)^[49], Delete multiple breakpoint
- [bccBreakpList](#)^[54], List defined breakpoints
- [bccBreakpInfo](#)^[60], Ask information for a breakpoint
- [bccBreakpStatus](#)^[69], Ask status for a breakpoint

Process specific:

- [bccProcessList](#)^[54], List available remote process
- [bccProcessInfo](#)^[63], Query process information
- [bccProcessFlashInfo](#)^[62], Query process flash information
- [bccProcessCmd](#)^[50], Execute a process command
- [bccProcessDbgCmd](#)^[51], Execute a process debug command
- [bccProcessGetDebugContext](#)^[59], Query debug context for a process
- [bccProcessGetTrace](#)^[69], Ask trace information for a process
- [bccProcessStatus](#)^[67], Ask R/T status for a process

Object block specific:

- [bccObjBlockList](#)^[55], List object blocks
- [bccProcessInspect](#)^[52], Inspect contents for a process

Operating system specific:

- [bccOsAttachedFList](#)^[57], List of OS attached function

Add a breakpoint

Code:	AS + 720
Symbolic:	bccBreakpAdd

This command will request to add a breakpoint, with the following parameters:

Offset	Type	Label	Description												
0	U32	DSID	Debug session ID												
4	U8	MODE	Breakpoint modality: 0=Inactive 1=Active 2=Active only for MCNT times 3=Skip MCNT times before active												
5	U16	MCNT	Parameter for MODE field												
7	U8	WHAT	What to set as breakpoint: 1=Physical address. WDATA is: <table border="1"> <tr> <th>Offset</th><th>Type</th><th>Label</th><th>Description</th></tr> <tr> <td>+0</td><td>U32</td><td>ADDR</td><td>Physical address</td></tr> <tr> <td>+4</td><td>U16</td><td>TYPE</td><td>Data type (use VAR type code)</td></tr> </table> 2=Process and step number. WDATA is:	Offset	Type	Label	Description	+0	U32	ADDR	Physical address	+4	U16	TYPE	Data type (use VAR type code)
Offset	Type	Label	Description												
+0	U32	ADDR	Physical address												
+4	U16	TYPE	Data type (use VAR type code)												

Offset	Type	Label	Description												
			<table> <tr> <th>Offset</th><th>Type</th><th>Label</th><th>Description</th></tr> <tr> <td>+0</td><td>U16</td><td>PID</td><td>Process ID</td></tr> <tr> <td>+2</td><td>U32</td><td>STEP</td><td>Step number</td></tr> </table> <p>3=Variable. WDATA is a standard VAR structure (#1)</p>	Offset	Type	Label	Description	+0	U16	PID	Process ID	+2	U32	STEP	Step number
Offset	Type	Label	Description												
+0	U16	PID	Process ID												
+2	U32	STEP	Step number												
8	U8	WDATA[10]	Data for what field, format according WHAT value.												
18	U8	TYPE	Breakpoint type: 1=Execution (fetch) 2=Write 3=Read/Write												
19	U8	COND	Breakpoint condition (for TYPE 2 and 3) refer to CDATA field: 0=No condition 1=Equal 2=Different 3=Less than 4=Less or equal than 5=Greater than 6=Greater or equal than 7=Mask equal (use CMASK field) 8=Mask not equal (use CMASK field)												
20	B8	CDATA[8]	Breakpoint condition data: format is depending on WHAT field (and VAR base type for WHAT=3).												
28	U32	CMASK	Breakpoint condition mask, for COND 7 and 8.												
32	U32	FLAGS	Breakpoint flags: 0x00000001 Stop process (when BP reached) 0x00000002 Stop trace (when BP reached)												

Offset	Type	Label	Description
			0x00000004 Save storage (when BP reached)
36	STRZ	INFO	Optional ascii information (for WHAT=3 variable name)

(#1) String type variables are not supported in this operation.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U32	BPID	Breakpoint unique ID

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackOutOfResource	Unable to set more breakpoints	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Address 2=Process ID 3=Step (in process) 4=Variable 5=Mode 6=What 7=Cond 8=Flags 9=Type 10=Address already has a BP
nackIllegalDebug	Illegal debug session	

Data format for process contents inspection

Contents data of a process is organized as a structured sequence of tags with any associated data.

Offset	Type	Name	Description
0	U8	TAG_ID	Tag code
1	U8[...]	TAG_DATA	Tag data (optional)

Currently the tag codes provided are as follows:

Name	Value	Description
<i>tag_invalid</i>	0	Invalid tag (reserved)
<i>tag_i8</i>	1	8 bit signed data (i8)
<i>tag_i16</i>	2	16 bit signed data (i16)
<i>tag_i32</i>	3	32 bit signed data (i32)
<i>tag_i64</i>	4	64 bit signed data (i64)
<i>tag_u8</i>	5	8 bit unsigned data (u8)
<i>tag_u16</i>	6	16 bit unsigned data (u16)
<i>tag_u32</i>	7	32 bit unsigned data (u32)
<i>tag_u64</i>	8	64 bit unsigned data (u64)
<i>tag_float</i>	9	32 bit floating point data (float)
<i>tag_real</i>	10	64-bit floating point data (double)
<i>tag_bool_false</i>	11	Boolean false value (has no data)
<i>tag_bool_true</i>	12	Boolean true value (has no data)
<i>tag_null</i>	13	Null value (has no data)
<i>tag_string</i>	14	String value
<i>tag_enum_i32</i>	15	Enumeration type value (i32)
<i>tag_enum_u32</i>	16	Enumeration type value (u32)
<i>tag_begin_uniform_array</i>	17	Start a block for uniform array type
<i>tag_begin_heterogeneous_array</i>	18	Start a block for heterogeneous array type
<i>tag_lazy_range</i>	19	Skip a range of values within a heterogeneous array (expected but not supported at current version)
<i>tag_lazy_id</i>	20	Skip a single value either within a heterogeneous array, or within a <i>tag_begin_static</i> or <i>tag_begin_instance</i> block
<i>tag_begin_static</i>	21	Start a block description for static fields of a type
<i>tag_begin_instance</i>	22	Start a block description for instance fields of a type

Name	Value	Description
<i>tag_begin_data</i>	23	Start a block root that will contain all data
<i>tag_missing</i>	24	Indicates that the expected field is not available (e.g., due to optimizations), has no data
<i>tag_end</i>	25	End of block (has no data)
<i>tag_begin_expr_data</i>	26	Start a block for additional (optional) data for an expression
<i>tag_begin_expr_value</i>	27	Start a block of data for a specific value for an expression part (within a <i>tag_begin_expr_data</i>)
<i>tag_base_instance</i>	28	Start a block for instance fields of a basic type (within an instance block)

Received data always have *tag_begin_data* as the start tag and *tag_end* as the end tag. More complex tags will be described below (DATA_FMT=0x101).

tag_string

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	SIZE	Length of the string (including final \0).
4	U8[SIZE]	VALUE	Representation of the string as a UTF-8 sequence, with final \0.

tag_enum_i32

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	TYPE_ID	Data type ID of the enumeration
4	I32	VALUE	Value of the enumeration

tag_enum_u32

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	TYPE_ID	Data type ID of the enumeration
4	U32	VALUE	Value of the enumeration

tag_begin_uniform_array

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U8	NSIZE	Number of array dimensions [1, 3]
1	U32[NSIZE]	SIZES	Length of the individual dimensions of the array
...	U32	TYPE_ID	Scalar type ID of data described

Notes:

- In the case of simple types (i8, i16, i32, i64, u8, u16, u32, u64, double, float, bool), the block following the tag directly contains the data without any associated tag, the size of each individual element is (1, 2, 4, 8, 1, 2, 4, 8, 4, 1) bytes, respectively.
- As an optimization, in case the content is of *tag_begin_instance* elements, the type field of the *tag_begin_instance* is omitted.
- The data block is closed by *tag_end*.

tag_begin_heterogeneous_array

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U8	NSIZE	Number of array dimensions [1, 3]
1	U32[NSIZE]	SIZES	Length of the individual dimensions of the array

Notes:

- Each individual array element is preceded by a tag.
- The data block is closed by *tag_end*.

tag_lazy_range

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	COUNT	Number of elements skipped, i.e., not

Offset	Type	Name	Description
			expanded

tag_lazy_id

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	TYPE_ID	Data type ID given of the skipped value, i.e., not expanded

tag_begin_static

The tag payload is composed as follows:

Offset	Type	Name	Description
0	u32	type_id	Data type ID that is described

Note:

- The data block is closed by *tag_end*.

tag_begin_instance

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	TYPE_ID	Data type ID being described

Note:

- In case you are describing a class type, the elements of the base class will be inserted immediately after the elements present directly in the class indicated by the type (and so on transitively) and will be preceded by the tag *tag_base_type*.
- The data block is closed by *tag_end*.

tag_base_type

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	TYPE_ID	Basic data type ID that is described

tag_begin_expr_value

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U32	BEGIN_POS	Initial position (in the expression) of the partial value description
4	U32	END_POS	End position (in the expression) of the partial value description

Data format for process debug context

Debug context data of a process is organized as a structured sequence of tags with any associated data.

Offset	Type	Name	Description
0	U8	TAG_ID	Tag code
1	U8[...]	TAG_DATA	Tag data (optional)

Currently the tag codes provided are as follows:

Name	Value	Description
<i>tag_invalid</i>	0	Invalid tag (reserved)
<i>tag_end</i>	1	End block (has no data)
<i>tag_begin_data</i>	2	Start main data block
<i>tag_begin_debug_files</i>	3	Debug file block start
<i>tag_debug_file</i>	4	Single debug file
<i>tag_begin_external_type_ids</i>	5	Start block ID external data types
<i>tag_external_type_id</i>	6	Single external data type ID

Received data always have *tag_begin_data* as the start tag and *tag_end* as the end tag. More complex tags (DATA_FMT=0x100) will be described below.

tag_debug_file

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U8	FILETYPE	File type: 0 = invalid 1 = EELF/R4D file 2 = R4D file

Offset	Type	Name	Description
1	U8	-	-
2	U16	FILEPATH_L	Length of the file path (including \0)
4	U8[FILEPATH_L]	FILE_PATH	Full path file name (in flash) that contains debugging information
	U32	RELOC_FLAGS (#2)	Relocation settings: 0x1 Enable path_id relocation 0x2 Enable relocation module_id 0x4 Enable relocation file_id 0x8 Enable relocation context_id 0x10 Enable relocation step_id 0x20 Enable relocation type_id 0x40 Enable relocation shared_module_id
...	U32	RELOC_PATH_ID_OFF	Offset relocation path_id (#1)
...	U32	RELOC_MODULE_ID_OFF	Offset relocation module_id (#1)
...	U32	RELOC_FILE_ID_OFF	Offset relocation file_id (#1)
...	U32	RELOC_CONTEXT_ID_OFF	Offset relocation context_id (#1)
...	U32	RELOC_STEP_ID_OFF	Offset relocation step_id (#1)
...	U32	RELOC_TYPE_ID_OFF	Offset relocation type_id (#1)
...	U32	RELOC_SHARED_MODULE_ID_OFF	Offset relocation shared_module_id (#1)

(#1) In case of value 0, and corresponding enable in reloc_flags, the ids in that category are queued to the corresponding ones in the previous debug file; if the category in reloc_flags is not enabled, the ids remain unchanged.

(#2) The default of the current implementation is always value 0x7f (all enable flags).

tag_external_type_id

The tag payload is composed as follows:

Offset	Type	Name	Description
0	U16	-	-
2	U16	FULLNAME_L	Full name length
4	U8[FULLNAME_L]	FULLNAME	Qualified full name, various names are in reverse order
...	U32	TYPE_ID	Value type ID

Debug session watch-dog

Code:	AS + 712
Symbolic:	bccDebugWd

This command will refresh debug session watchdog for specified ID. Request parameters are the following:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID
4	U32	TIMEWD	Watchdog time [ms]

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Required debug session (by ID) not found or expired	
nackIllegalArgs	Illegal arguments	U16 What illegal: 2=Watchdog time [ms]

Delete a breakpoint

Code:	AS + 721
Symbolic:	bccBreakpDel

This command will request to delete an existing breakpoint, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID

Offset	Type	Label	Description
4	U8	NLIST	No. of break point (0xFF=All breakpoint for session)
5	U32	LIST[]	List of breakpoint ID to delete

On success, a [bccAck](#)₁₆₄ is received with the no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Breakpoint not found	U16 Breakpoint index
nackIllegalDebug	Illegal debug session	

Execute a process command

Code:	AS + 705
Symbolic:	bccProcessCmd

This command will request to execute a specific remote process command, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session if available)
4	U32	FLAGS	Operating flags: 0x00000001 Search process by filename (ignore PID)
8	U16	PID	Process ID
10	U16	CMDID	Command ID: 1=Load 2=Start 3=Stop 4=Reload 5=Reload and restart 6=Reload and synchronized restart
12	U8[16]	-	Reserved
28	STRZ	FILENAME	Process filename

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a `bccNack`¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
<code>nackNotFound</code>	Process not found	
<code>nackFileNotExist</code>	Process filename not found	
<code>nackIllegalArgs</code>	Illegal parameters	U16 What illegal: 1 = Flags 2 = Command ID 3 = Filename
<code>nackIllegalDebug</code>	Illegal debug session	
<code>nackRequestError</code>	Error executing the command (request)	U16 NACK Error code

Execute a process debug command

Code:	AS + 702
Symbolic:	<code>bccProcessDbgCmd</code>

This command will request to execute a specific remote process debug command, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID
4	U32	FLAGS	Operating flags:
8	U16	PID	Process ID
10	U16	CMDID	Command ID: 1=Start 2=Stop 3=Continue 4=Step in 5=Step over 6=Run to (STEP required) 7=Execute until out 8=Enable trace (opt, enable up to STEP) 9=Disable trace 10=Kill (stop and unschedule it)
12	U32	STEP	ID step

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackIllegalArgs	Illegal parameters	U16 What illegal : 1 = STEP value 2 = Command 3 = Not applicable 4 = Flags
nackIllegalDebug	Illegal debug session	
nackRequestError	Error executing the command (request)	U16 NACK Error code

Inspect contents of a process

Code:	AS + 707
Symbolic:	bccProcessInspect

This command will inspect contents of a remote user process: this is a standard [data exchange sequence](#)^[19].

The REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	QSIZE	Query data size
4	U8[12]	-	(reserved)
16	U32	DSID	Debug session ID (0=Use read-only session, if available)
20	U32	FLAGS	Operating flags
24	U16	PID	Process ID
26	U16	QTYPE	Query type: 1 = Inspect R++ expression

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackIllegalDebug	Illegal debug session	
nackIllegalArgs	Illegal parameters	U16 What illegal:

NACK code	Description	Extra data
		1 = Flags 2 = Query size 3 = Query type

Specific data for type 1 (inspect R++ expression)

The QUERYDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Query flags: 0x1 = Query sub-expressions data
4	U16	DATA_FMT	Maximum data format version handled by client.
6	U16	CTX_OFF	Context key offset (#1)
8	U16	EXPR_OFF	Expression offset (#1)
...	STRZ	CTX	Context key
...	STRZ	EXPR	Expression

(#1) Offset is related to the QUERYDATA buffer itself.

The ANSWERDATA structure is the following

Offset	Type	Label	Description
0	U16	RESULT	Result code: 0=operation completed 1=operation completed but with warning (#1) 2=operation not completed due to errors (#1)
2	U16	DATA_OFF	Data offset (#2)
4	U16	DATA_SIZE	Data size
6	U16	DATA_FMT	Data format version: 0=invalid 0x100=R++ tag format (v1) 0x101=R++ tag format (v2)
...	...	DATA	Contents data ⁴²

(#1) Warning and error informations are contained in *DATA*, according to *DATA_FMT*. For example, for *DATA_FMT=0x1xx*, any context and/or content errors in the expression, could be signaled with the **tag_error** within the resulting data (so there is a chance for more errors).

(#2) Offset is related to the *ANSWER*

List available remote process

Code:	AS + 700
Symbolic:	bccProcessList

This command will request a list of ID of all available remote user process, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session if available)
4	U32	FLAGS	Operating flags:

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U8	COUNT	N. of process
1	U16	PID0	Process ID 0
3	U16	PID1	Process ID 1
...

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal? 1 = Flags
nackIllegalDebug	Illegal debug session	

List defined breakpoints

Code:	AS + 722
Symbolic:	bccBreakpList

This command will request a list of existing breakpoints, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U8	NLIST	No. of break point
1	U32	LIST[]	List of breakpoint ID

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalDebug	Illegal debug session	

List object blocks

Code:	AS + 770
Symbolic:	bccObjBlockList

This command will list all loaded and instanced object blocks of the connected device: this is a standard [download transfer sequence](#)₁₄.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Compiler information 0x00000002 Instance information 0x00000004 Instance parameters 0x00000008 Instance occurrence

If initial request fails, [bccNack](#)₁₆₄ is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U16	RECID	Record type ID: 1=General information 2=Compiler information 3=Instance information

Offset	Type	Label	Description
			4=Instance parameters 5=Instance occurrence
2	U8[...]	DATA	Data according type ID

Notes:

- Instance information and compiler information records are always referred to the previous general information record.
- Instance parameter records (multiple) are always referred to the previous instance information record.
- Instance occurrence records (multiple) are always referred to the previous instance information record.

Specific data for type 1 (general information):

Offset	Type	Label	Description
+0	U32	FLAGS	Class flags: (see HEADER.FLAGS in OBB file specifications)
+4	U32	VERSION	Class version (nvMake format)
+8	U8	HH	Compilation time (hour)
+9	U8	MM	Compilation time (minute)
+10	U8	SS	Compilation time (second)
+11	U8	DD	Compilation date (day)
+12	U8	MM	Compilation date (month)
+13	U8	YY	Compilation date (year - 2000)
+14	U32	CID	Class unique ID
+18	U8[14]	--	(reserved)
+32	STRZ	NAME	Object block class name

Specific data for type 2 (compiler information):

Offset	Type	Label	Description
+0	U32	FLAGS	Compiler flags: (none)
+4	STRZ	INFO	Compiler informations

Specific data for type 3 (instance information):

Offset	Type	Label	Description
+0	U32	FLAGS	Instance flags: (none)
+4	U32	DIM	Array dimension (0=none)
+8	U32	IID	Instance unique ID
+12	U8[20]	--	(reserved)
+32	STRZ	NAME	Instance name

Specific data for type 4 (instance parameter):

Offset	Type	Label	Description
+0	U32	IIX	Instance index
+4	STRZ	PARAM	Instance parameter

Specific data for type 5 (instance occurrence):

Offset	Type	Label	Description
+0	U32	FLAGS	Occurrence flags: (none)
+4	U16	PID	Owner process ID
+6	U32	ADDR	Occurrence address

List OS attached functions

Code:	AS + 736
Symbolic:	bccOsAttachedFList

This command will list all operating system attached functions of the connected device: this is a standard [download transfer sequence](#)¹⁴.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)

If initial request fails, [bccNack](#)^[164] is received. Specific errors":

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U16	RECID	Record type ID: 1=Hooking group 2=Attached function
2	U8[...]	DATA	Data according type ID

Notes:

- Attached function records (multiple) are always referred to the previous hooking group record.

Specific data for type 1 (hooking group):

Offset	Type	Label	Description
+0	U32	FLAGS	Group flags: (none)
+4	U32	GID	Group ID
+8	U8[24]	--	(reserved)
+32	STRZ	NAME	Group name

Specific data for type 2 (attached function):

Offset	Type	Label	Description
+0	U32	FLAGS	Function flags: (none)
+4	U32	FID	Function ID
+8	U32	FPAR	Function parameter
+12	U8[20]	--	(reserved)
+32	STRZ	TEXT	Function description

Query debug context for a process

Code:	AS + 708
Symbolic:	bccProcessGetDebugContext

This command queries the debug context for a remote user process: this is a standard [data save sequence](#).^[16]

The structure of REQDATA is as follows:

Offset	Type	Label	Description
0	U8[16]	-	(reserved data saving sequence . ^[16])
16	U16	DSID	Debug session ID (0=Use read-only session if available)
18	U32	FLAGS	Operational flags: 0x1=Interrogate debug file 0x2=Interrogate external type_id
22	U16	PID	Process ID
24	U16	DATA_FMT	Maximum version of the data format managed by the client.

The structure of ACKDATA is as follows:

Offset	Type	Label	Description
0	U8[16]	-	(reserved data saving sequence . ^[16])
16	U16	DATA_FMT	Data format version

If the initial request fails, [bccNack](#).^[16] is received. Specific errors:

NACK Code	Message description	Extra data
nackNotFound	Process not found	
nackIllegalDebug	Illegal debugging session	
nackIllegalArgs	Illegal parameters	U16 What illegal: 1 = Flags 2 = Data format version

The structure of INDATA is as follows:

Offset	Type	Label	Description
0	U8[...]	DATA	Context data ^[47]

Query information for a breakpoint

Code:	AS + 723
Symbolic:	bccBreakpInfo

This command will request definition information about a specific breakpoint, with following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID
4	U32	BPID	Breakpoint ID

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description												
0	U32	-	Reserved												
4	U8	MODE	Breakpoint modality: 0=Inactive 1=Active 2=Active only for MCNT times 3=Skip MCNT times before active												
5	U16	MCNT	Parameter for MODE field												
7	U8	WHAT	What to set as breakpoint: 1=Physical address. WDATA is: <table border="1"> <tr> <th>Offset</th><th>Type</th><th>Label</th><th>Description</th></tr> <tr> <td>+0</td><td>U32</td><td>ADDR</td><td>Physical address</td></tr> <tr> <td>+4</td><td>U16</td><td>TYPE</td><td>Data type (use VAR type code)</td></tr> </table>	Offset	Type	Label	Description	+0	U32	ADDR	Physical address	+4	U16	TYPE	Data type (use VAR type code)
Offset	Type	Label	Description												
+0	U32	ADDR	Physical address												
+4	U16	TYPE	Data type (use VAR type code)												

Offset	Type	Label	Description												
			<p>2=Process and step number. WDATA is:</p> <table> <tr> <th>Offset</th><th>Type</th><th>Label</th><th>Description</th></tr> <tr> <td>+0</td><td>U16</td><td>PID</td><td>Process ID</td></tr> <tr> <td>+2</td><td>U32</td><td>STEP</td><td>Step number</td></tr> </table> <p>3=Variable. WDATA is a standard VAR^[9] structure (*)</p>	Offset	Type	Label	Description	+0	U16	PID	Process ID	+2	U32	STEP	Step number
Offset	Type	Label	Description												
+0	U16	PID	Process ID												
+2	U32	STEP	Step number												
8	U8	WDATA[10]	Data for what field, format according WHAT value.												
18	U8	TYPE	<p>Breakpoint type:</p> <p>1=Execution (fetch)</p> <p>2=Write</p> <p>3=Read/Write</p>												
19	U8	COND	<p>Breakpoint condition (for TYPE 2 and 3) refer to CDATA field:</p> <p>0=No condition</p> <p>1=Equal</p> <p>2=Different</p> <p>3=Less than</p> <p>4=Less or equal than</p> <p>5=Greater than</p> <p>6=Greater or equal than</p> <p>7=Mask equal (use CMASK field)</p> <p>8=Mask not equal (use CMASK field)</p>												
20	B8	CDATA[8]	Breakpoint condition data: format is depending on WHAT field (and VAR base type for WHAT=3).												
28	U32	CMASK	Breakpoint condition mask, for COND 7 and 8.												
32	U32	FLAGS	<p>Breakpoint flags:</p> <p>0x00000001 Stop process (when BP reached)</p>												

Offset	Type	Label	Description
			0x00000002 Stop trace (when BP reached) 0x00000004 Save storage (when BP reached)
36	STRZ	INFO	Optional ascii information (for WHAT=3 variable name)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Breakpoint not found	
nackIllegalDebug	Illegal debug session	

Query process flash information

Code:	AS + 706
Symbolic:	bccProcessFlashInfo

This command will query for flash information of a remote user process with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session, if available)
4	U32	FLAGS	Operating flags
8	U16	PID	Process ID

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U32	-	Reserved
4	U32	-	Reserved
8	STRZ	FILENAME	Filename

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackFileNotExist	Process has no corresponding flash file	

NACK code	Description	Extra data
nackIllegalDebug	Illegal debug session	
nackIllegalArgs	Illegal parameters	U16 What illegal: 1 = Flags

Query process information

Code:	AS + 701
Symbolic:	bccProcessInfo

This command will request information about a specific remote user process, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session if available)
4	U32	FLAGS	Operating flags: 0x00000001 Search by name (#1) (ignore PID) 0x00000002 Search by flash filename (#1) (ignore PID)
8	U16	PID	Process ID
10	STRZ	NAME	Optional search process name (if flag supplied) or flash filename (if flag supplied)

(#1) Search flags are mutually exclusive.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U8	HH	Time (hour) of the process
1	U8	MM	Time (minute) of the process
2	U8	SS	Time (second) of the process
3	U8	DD	Date (day) of the process

Offset	Type	Label	Description
4	U8	MO	Data (month) of the process
5	U8	YY	Data (year - 2000) of the item
6	U16	PID	Process ID
8	U32	PTYPE	Process type: 0x00000000 Generic 0x00000001 R 16bit 0x00000002 (riservato) 0x00000003 RHLL 16bit 0x00000004 (riservato) 0x00000005 RHLL 32bit 0x00000006 DSP56 0x00000007 R, 32bit 0x00000008 R3.COF, 32bit 0x00000009 Ladder diagram 0x0000000A R3.ELF PPC, 32bit 0x0000000B R3.ELF X86, 32bit 0x0000000C R3.ELF ARM, 32bit 0x0000000D System, 32bit 0x0000000E R4.ELF ARM, 32bit
12	U32	PCF	Process capabilities flags: 0x00000001 Support breakpoints 0x00000002 Support trace 0x00000004 Support process control 0x00000008 Process can be stopped

Offset	Type	Label	Description
			0x00000010 Process can be started 0x00000020 Process can be killed 0x00000040 Support variable forcing
16	U16	CRC16	Binary file crc16 value (0=Information not available)
18	U16	-	Reserved
20	B[64]	DATA	Data according language type
84	STRZ	NAME	Process name

Specific DATA for PTYPE 0x00000007 (R):

Offset	Type	Label	Description
+0	U16	PSTYPE	Process subtype: 0x0000 = Unknown 0x0001 = Time sharing 0x0002 = Rule 0x0003 = Rule on input 0x0004 = Timed asynchronous rule 0x0006 = Periodic rule
+2	U32	MSIZE	Memory size [byte]
+6	U32	STEPC	No. of executable steps
+10	B[32]	CCID	Compiler ID string

Specific DATA for PTYPE 0x8 (R3.COF):

Offset	Type	Label	Description
+0	U16	PSTYPE	Process subtype: 0x0000 = Unknown 0x0001 = Time sharing 0x0002 = Rule

Offset	Type	Label	Description
			0x0003 = Rule on input 0x0004 = Timed asynchronous rule 0x0006 = Periodic rule
+2	U32	MSIZE	Memory size [byte]
+6	U32	STEPC	No. of executable steps
+10	B[32]	CCID	Compiler ID string
+42	U16	N_EXT	No. of external symbols
+44	U16	N_UNR	No. of unresolved symbols

Specific DATA for PTYPE 0x9 (Ladder diagram):

Offset	Type	Label	Description
+0	U8	PTYPE	Process type: 0 = Synchronous 1 = High priority 2 = Normal priority 3 = Low priority
+1	U8	TID	Target ID: 0x00 = Firmware RTE
+2	U16	PFREQ	Process frequency [hz]
+4	B[32]	CCID	Compiler ID string
+36	U32	VID	Task version ID
+40	U32	PLVID	Task version ID pre live changes

Specific DATA for PTYPE 0xA (R3.ELF PPC), 0xB (R3.ELF X86), 0xC (R3.ELF ARM), 0xE (R4.ELF ARM) and 0xD (System):

Offset	Type	Label	Description
+0	U16	PSTYPE	Process subtype: 0x0000 = Unknown 0x0001 = Time sharing

Offset	Type	Label	Description
			0x0002 = Rule 0x0003 = Rule on input 0x0004 = Timed asynchronous rule 0x0005 = (reserved) 0x0006 = Periodic rule 0x0007 = (reserved) 0x0008 = (reserved) 0x0009 = (reserved) 0x000A = (reserved) 0x000B = RPE extension 0x000C = XPL extension 0x000D = Service RTE-1 0x000E = Service RPE 0x000F = Service OB 0x0010 = Service RTE-2
+2	U32	MSIZE	Memory size [byte]
+6	U32	STPC	No. of executable steps
+10	B[32]	CCID	Compiler ID string
+42	U16	N_EXT	No. of external symbols
+44	U16	N_UNR	No. of unresolved symbols

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackIllegalDebug	Illegal debug session	

Query runtime status for process

Code:	AS + 704
Symbolic:	bccProcessStatus

This command will query for current runtime information of a remote user process with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session, if available)
4	U32	FLAGS	Operating flags
8	U16	PID	Process ID

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U16	STATE	Current process state: 0x0000 Undefined 0x0001 Scheduled (#1) 0x0002 Running 0x0003 Stopped 0x0004 Stopping (trying to)
2	U32	FLAGS	Process flags: 0x00000001 Trace enabled 0x00000002 Breakpoints defined 0x00000004 Variables forced 0x00000008 Runtime errors
6	U32	STEP	Current step

(#1) Scheduled means loaded in memory but not running.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackIllegalArgs	Illegal arguments	U16 What illegal? 1 = Flags
nackIllegalDebug	Illegal debug session	

Query status for a breakpoint

Code:	AS + 724
Symbolic:	bccBreakpStatus

This command will request runtime status about a specific breakpoint, with following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID
4	U32	BPID	Breakpoint ID

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	STATUS	Current breakpoint status: 0x00000001 Enabled 0x00000002 Reached
4	U32	COUNT	Event counter
8	U32	STEP	Event step (or IP) value (-1 no step)
12	DBL	VALUE	Event value
20	U16	PID	Event process ID (-1 no process or internal)
22	STRZ	NAME	Event (caller) name

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Breakpoint not found	
nackIllegalDebug	Illegal debug session	

Query trace information for process

Code:	AS + 703
Symbolic:	bccProcessGetTrace

This command will query for current trace steps for a remote user process, with the following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID (0=Use read-only session, if available)
4	U32	FLAGS	Operating flags:
8	U16	PID	Process ID

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U8	NSTEP	No. of steps
1	U32	STEP0	Step 0
...
	U32	STEPn	Step N

NOTE: if trace is not active, it will even respond [bccAck](#)₁₆₄ but providing NSTEP=0 (like no trace data).

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Process not found	
nackIllegalArgs	Illegal arguments	U16 What illegal? 1 = Flags
nackIllegalDebug	Illegal debug session	

Start a debug session

Code:	AS + 710
Symbolic:	bccDebugStart

This command will request remote device to open a debug session: inside this session, debug command (breakpoint, process, trace, etc.) can be used. The request has following parameters:

Offset	Type	Label	Description
0	U32	TIMEWD	Initial watchdog time [ms]

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	DSID	Debug session unique ID

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackOutOfResource	No more debug session available (in case of multiple debug session available)	
nackResourceBusy	Debug session already in use (in case of single debug session available)	

Notes:

- In order to keep debug session alive, you have to send periodically the [bccDebugWd](#)^[49] with the desired watchdog time.

Stop a debug session

Code:	AS + 711
Symbolic:	bccDebugStop

This command will request remote device to close a debug session. The request has following parameters:

Offset	Type	Label	Description
0	U32	DSID	Debug session ID

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Required debug session (by ID) not found	

Device handling

These messages are used for general handling of a connected device.

System configuration:

[bccSysInfo](#)^[75], query system information

[bccAutoConfig](#)^[82], request device auto configuration

[bccResolveProcObject](#)^[82], resolve a /proc object

Modality:

[bccGetMode](#)^[74], get current mode

[bccSetMode](#)^[83], set or request mode

Compatibility:

[bccAsciiCmd](#)^[73], execute a generic ASCII command

System reset:

[bccSoftwareReset](#)^[81], request a software reset

[bccHardwareReset](#)^[81], request a hardware reset

[bccCMosReset](#)^[81], request a CMos ram reset

Only One Write session management:

[bccOOWSessionBegin](#)^[72], begin an OOW session

[bccOOWSessionEnd](#)^[73], end an OOW session.

[bccOOWSessionQueryInfo](#)^[74], query information for an OOW session.

Begin an OOW session

Code:	AS + 530
Symbolic:	bccOOWSessionBegin

This command will request to begin an OOW (Only One Write) session on the current link of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags 0x1 Source TCP port is used together with source IP address as owner (supported only for TCP/IP)
4	U32	TOUT	Timeout to keep the session active [ms] (1-15000)
8	U8[7]	-	(reserved)
15	U8	TEXTSIZE	Size of the TEXT field
16	STRZ	TEXT	Session (owner) description

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Illega flags 2=Illegal timeout

NACK code	Description	Extra data
nackNotAuthorized	The operation is not authorized	
nackAlreadyActive	The OOW session is already active	
nackNotActive	The OOW session is not active (WRITE ALWAYS modality selected)	
nackIllegalContext	The operation context is illegal (for example if not over an tcp/ip link)	

End an OOW session

Code:	AS + 531
Symbolic:	bccOOWSessionEnd

This command will request to end an OOW (Only One Write) session on the current link of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackNotAuthorized	The operation is not authorized	
nackIllegalContext	The operation context is illegal (for example if not over an tcp/ip link)	

Execute a generic ASCII command

Code:	AS + 518
Symbolic:	bccAsciiCmd

This command will request to execute a generic ASCII command into connected device: this is a standard [download transfer sequence](#)₁₄.

REQDATA structure is the following:

Offset	Type	Label	Description
0	B[]	CMD	ASCII command (0 terminated)

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	B[]	TEXT	ASCII answer

NOTE: for more information about old ASCII command, read the specific device (or language) user guide.

Query current mode

Code:	AS + 508
Symbolic:	bccGetMode

This command will get current working mode of the connected device. Request has no parameters.

On success, a [bccAck](#)^[164] will be received with the following data:

Offset	Type	Label	Description
0	U8	MODE	Modality code: 0=Initializing system 1=0 Cycle 2=Programming 3=Execution 4=User defined 0 cycle 5=Other
1	U32	FIRMWTYPE	Firmware type identification. For more information take a look at a FIRMWTYPE field in answer data of bccSysInfo ^[75] command.

On failure, a [bccNack](#)^[164] is received.

Query info for an OOW session

Code:	AS + 532
Symbolic:	bccOOWSessionQueryInfo

This command will request information for an OOW session on the current link of the connected device. Request has no parameters.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Status flags 0x1 OOW session management enabled 0x2 OOW session opened 0x4 Source TCP port is used together with source IP address as owner (supported only for TCP/IP)
4	U32	TOUT	Timeout to keep the session active [ms]
8	U32	IPADDR	IPv4 source address of the session owner
12	U16	PORT	TCP source port of the session owner
14	U8	-	(reserved)
15	U8	TEXTSIZE	Size of the TEXT field
16	STRZ	TEXT	Session (owner) description

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackNotAuthorized	The operation is not authorized	
nackIllegalContext	The operation context is illegal (for example if not over an tcp/ip link)	

Query system information

Code:	AS + 507
Symbolic:	bccSysInfo

This command will query system information from the connected device. Request has no parameters.

On success, a [bccAck](#)^[164] will be received with the following data:

Offset	Type	Label	Description
0	U32	MANUFACTURER	Manufacturer identification: 0=Generic 1=Robox standard 2=Robox custom 3=Motorola 4=Parker
4	U32	MODEL	Model identification, depending of manufacturer. Always valid: 0=Generic 1=Virtual or hardware emulator 16=Motorola DSP56803 (obsolete, with vendor=3) 16=Robox RBXM 17=Robox RBXE or RBXC 18=Robox RPM 19=Robox uRbx 20=Robox CANSIN 21=Robox RBXM G2 22=Robox CPU PPC Hi-Drive 23=Robox NIOS (ethercat) 24=Robox uRMC 25=Robox IMD20 26=Robox CANSIN RTE 27=Robox RMC G2 28=Robox RBXM P2020 29=Robox μ RMC2 30=Robox μ RMC3 31=Robox IMD30 32=Robox μ RIO

Offset	Type	Label	Description
			33=Parker generic device 34=Robox EthIP/PNET board 35=Robox RP-1 36=Robox RP-2 37=Robox ECATSIN 38=Robox EC2A 39=Robox RP-0 40=Robox Safety Board 41=Robox ECATSIN RTE 42=Robox Dongle 43=Robox Integrated Drive RID20-E 44=Robox RP-2 (Bus Slave) 45=Robox RHAM 46=Robox μ RP-2 47=EFT cabinet board 48=SI-uPAC
8	U32	OSTYPE	Operating system identification: 0=Generic 1=OSF 16bit 2=OSF 16bit with integrated language 3=OSF 32bit 4=OSF 32bit with integrated language 5=Embedded OS for DSP56 6=Embedded OS for CANSIN 7=Eprom (OSE) 8=Embedded OS for NIOS (ethercat) 9=Embedded OS for IMD 10=Embedded OS for Parker generic

Offset	Type	Label	Description
			11=Ethernet IP interface for EthIP/PNET board 12=ProfiNET interface for EthIP/PNET board 13=Embedded OS for EC2x
12	U32	OSVERSION	Operating system version (nvMake format)
16	U32	FIRMWTYPE	Firmware type identification: 0=Generic 1=RRT 16bit 2=RTT 16bit integrated in OS 3=RHLL 16bit 4=RHLL 16bit integrated in OS 5=RHLL 32bit integrated in OS 6=Device specific firmware (integrated in OS) 7=RRT 32bit integrated in OS 8=RTE 32bit integrated in OS
20	U32	FIRMWVERSION	Firmware version (nvMake format)
24	B[32]	USERTEXT	User software description
56	U32	USERVERSION	User software version (nvMake format)
60	U32	ATTRIB	Device attributes: 0x00000001 Remote variable set available 0x00000002 Intel 32bit HW variable available 0x00000004 BCC protocol translation active (#1)

Offset	Type	Label	Description
64	U8	TBTYPE	Taskbin type identification: 0=None 1=Classic taskbin 2=Taskbin++ 16bit version 3=Taskbin++ 32bit version
65	U32	TBVERSION	Taskbin version (nvMake format)
69	U32	VARSETID	Variable current set unique ID
73	U32	SLIBTYPE	System library type: 0=Generic 1=RBXLIB (Robox customized GCC library)
77	U32	SLIBVERSION	System library version (nvMake format)
81	U8	BIOSTYPE	BIOS type identification: 0=None 1=Robox BIOS
82	U32	BIOSVERSION	BIOS version (nvMake format)
86	U32	FIRMWEXT	Firmware extensions: 0x00000001 Robox Path Executor (RPE) 0x00000002 Robox XPL Executor (XPL)
90	U32	RPEVERSION	RPE version (nvMake format)
94	U16	LANGUAGE	System language: 0x0000=Neutral (or not set) 0x0004=Cinese semplificato (ZH) 0x0007=Tedesco (DE) 0x0009=Inglese (UK/US)

Offset	Type	Label	Description
			0x000a=Spagnolo (ES) 0x000c=Francese (FR) 0x0010=Italiano (IT) 0x0012=Coreano (KO) 0x0016=Portoghese (PO) 0x0019=Russo (RU)
96	U32	XPLVERSION	XPL version (nvMake format)
100	U32	XPLLANGS	XPL languages supported 0x00000001 RPL 0x00000002 GCODE RS274
104	U16	IB_TX	Inter-block transmit capacity
106	U16	IB_RX	Inter-block receive capacity
108	U8	OSETYPE	OSE type identification: 0=None 1=Robox OSE
109	U32	OSEVERSION	OSE version (nvMake format)

Notes:

- To validate the structure you must have at least a data area of 69 bytes (aka the base version of the bccSysInfo command): extra missing data must be considered as 0 filled.
- This command is (and must be) always authorized.
- The attribute #1 means that someone (the device hardware or the software) is translating BCC protocol from / to another protocol; this should be considered as a warning that not all documented BCC command could be available and communication timings can be different than the native BCC protocol timings.
- If IB_TX (or IB_RX) field has value of 0 it will be assumed as default value: the default value is 4 (for backward compatibility reasons).

On failure, a [bccNack](#)₁₆₄ is received.

Request a CMOS ram reset

Code:	AS + 512
Symbolic:	bccCMosReset

This command will request a CMOS ram reset for the connected device . Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received.

WARNING: on some device, a successfully CMos ram reset could cause a device reset too.

Request a hardware reset

Code:	AS + 506
Symbolic:	bccHardwareReset

This command will request the device to have a hardware reset. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags (#1): 0x1 Reset software for main hardware 0x2 Reset software for NetController

(#1) If the value of FLAGS is 0, for backward compatibility, the software reset is intended only for main hardware. If different than 0, the reset operate only on required subjects. Available from RTE v34.26.0.

On success, a [bccAck](#)₁₆₄ is received with no data: some time after, the device will perform the required operation: check your hardware/software documentation to have more details about this operation.

On failure, a [bccNack](#)₁₆₄ is received.

Request a software reset

Code:	AS + 505
Symbolic:	bccSoftwareReset

This command will request the device to have a software reset. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags (#1): 0x1 Reset software for main hardware

Offset	Type	Label	Description
			0x2 Reset software for NetController

(#1) If the value of `FLAGS` is 0, for backward compatibility, the software reset is intended only for main hardware. If different than 0, the reset operate only on required subjects. Available from RTE v34.26.0.

On success, a [bccAck](#)^[164] is received with no data: some time after, the device will perform the required operation: check your hardware/software documentation to have more details about this operation.

On failure, a [bccNack](#)^[164] is received.

Request device auto configuration

Code:	AS + 511
Symbolic:	bccAutoConfig

This command will request auto configuration for the connected device . Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Use fast configuration 0x00000002 Notify end of operation 0x00000004 Generate file in update mode

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received.

Notes:

- If notify required (bit 0x2 in `FLAGS`), a [bccCompleted](#)^[167] is received when operation is done: if fail, a [bccAborted](#)^[165] is received.
- The [bccCompleted](#)^[167] message and the [bccAborted](#)^[165] message will have PID as request command PID.

Resolve a /proc object

Code:	AS + 523
Symbolic:	bccResolveProcObject

This command will try to resolve a /proc object, based on specified data. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags
4	U32	SRC	Query source: 1 = Query by Robox station ID (U16)
8	U32	OP	Query operation: 1 = Load file via FB/FOE 2 = Save file via FB/FOE
12	U32	-	(reserved)
16	B[]	DATA	Query data (depend on query source)

On success, a [bccAck](#)_[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags
4	U32	-	(reserved)
8	B[]	OBJPATH	The resolved /proc object path

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Query source 3=Query operation 4=Query data
nackNotFound	Object not found	

Set current mode

Code:	AS + 509
Symbolic:	bccSetMode

This command will try to set current working mode of the connected device. Request has the following parameters:

Offset	Type	Label	Description
0	U8	MODE	Request modality code: 0=Initializing system 1=0 Cycle 2=Programming 3=Execution 4=User defined 0 cycle 5=Other
1	U8	HOW	How to set mode: 0 = Immediately 1 = As soon as possible

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 Param ID 1 = Bad modality 2 = Bad set mode

Field bus device handling

These messages are used for general handling of a directly connected field bus devices.

Local Interface:

- [bccFbReadLocalEntry](#)^[92], read an entry
- [bccFbWriteLocalEntry](#)^[104], write an entry
- [bccFbWriteLocalEntryE](#)^[112], write an extended entry
- [bccFbReadLocalNmt](#)^[98], read NMT status
- [bccFbWriteLocalNmt](#)^[116], write NMT command

EtherCAT Interface:

- [bccFbReadCoeEntry](#)^[90], read an entry
- [bccFbWriteCoeEntry](#)^[102], write an entry
- [bccFbWriteCoeEntryE](#)^[108], write an extended entry
- [bccFbReadEcatNmt](#)^[97], read NMT status

CANOpen Interface:

- [bccFbReadCanEntry](#)^[87], read an entry
- [bccFbWriteCanEntry](#)^[99], write an entry
- [bccFbWriteCanEntryE](#)^[106], write an extended entry
- [bccFbReadCanNmt](#)^[96], read NMT status
- [bccFbWriteCanNmt](#)^[115], write NMT command
-

Interface commands:

- [bccFbReadIF](#)^[95], read interface information
- [bccFbWriteIF](#)^[114], write interface information

Miscellaneous:

[Field bus entry data type](#)^[85]

- [bccFbWriteEcatNmt](#)⁸⁶, force NMT status [Field bus supported interface type IDs](#)⁸⁵

Field bus supported interface type IDs

These are the currently reserved range of interface type IDs:

Interface type ID	Description
0x00000000-0x0000FFFF	(reserved)
0x00010000-0x0002FFFF	IMD class devices
0x00030000-0x0003FFFF	µRIO class devices
0x00040000-0x0004FFFF	EC2x class devices
0x00050000-0x0005FFFF	Safety Board class devices
0x00060000-0x000FFFFF	(reserved)
0x00100000-0x001FFFFF	(revoked)
0x00200000-0x002FFFFF	EVER devices
0x00300000-0x004FFFFF	(revoked)
0x00500000-0xFFFFFFFF	(reserved)

NOTE: specific device type ID are defined in the specific XML configuration file.

Filed bus entry data types

In reading and writing entries from the object dictionary you have to specific the entry data type, that is a combination of the following options:

Value	Description
0x00000001	8bit data
0x00000002	16bit data
0x00000004	32bit data
0x00000008	64bit data
0x00000010	Signed data (otherwise unsigned data)
0x00000020	Floating point data (otherwise fixed data)
0x00000040	Record type
0x00000080	Type parameter to be stored in flash
0x00000100	Object readable
0x00000200	Object writable
0x00000400	Only in pre-operational mode

Value	Description
0x00000800	Only with IGBT bridge disabled
0x00001000	Object can be mapped in PDO area
0x00002000	Object write require special handling (not all values in range are valid)
0x00004000	(reserved)
0x00008000	(reserved)
0x00010000	(reserved)
0x00020000	(reserved)
0x00040000	(reserved)
0x00080000	(reserved)
0x00100000	Object is a 0-terminated string
0x00200000	(reserved)

Force NMT status to EtherCAT Interface

Code:	AS + 913
Symbolic:	bccFbWriteEcatNmt

This command will force the NMT status to the device EtherCAT Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	NODE	Node address
8	U32	FSTATUS	NMT status to force: 0=Remove forced 1=Request Init 2=Request PreOperational 4=Request SafeOperational 8=Request Operational
12	U32	FTOUT	NMT force timeout [ms]

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: (none)
4	U32	NODE	Node address (echo)
8	U32	STATUS	Actual NMT status: 1=Init 2=PreOperational 3=Boot 4=SafeOperational 8=Operational
12	U32	FSTATUS	Forced NMT status: 1=Init 2=PreOperational 3=Boot 4=SafeOperational 8=Operational
16	U32	FTOUT	NMT force timeout [ms] (echo)

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=NMT status 4=NMT force timeout
nackInterfaceNotFound	EtherCAT interface not present	
nackInterfaceNotReady	EtherCAT interface not running	

Read an entry from CANopen Interface

Code:	AS + 920
Symbolic:	bccFbReadCanEntry

This command will read an entry from the device CANopen Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ⁸⁵
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

In all other cases, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type T is I64
- If the (DTYPE & 0x3F) value is 0x8, the type T is U64
- In all other cases T is DBL

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	CANopen interface not present	
nackInterfaceNotReady	CANopen interface not running	

Read an entry from EtherCAT (CoE) Interface

Code:	AS + 910
Symbolic:	bccFbReadCoeEntry

This command will read an entry from the device EtherCAT (CoE) Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags:


Offset	Type	Label	Description
			0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type 
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

In all other cases, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value

Offset	Type	Label	Description
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type T is I64
- If the (DTYPE & 0x3F) value is 0x8, the type T is U64
- In all other cases T is DBL

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	COE interface not present	
nackInterfaceNotReady	COE interface not running	

Read an entry from Local Interface

Code:	AS + 900
Symbolic:	bccFbReadLocalEntry

This command will read an entry from the device Local Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ^[85]
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

Otherwise the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- In all other cases <T> is DBL

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	Local interface not present	
nackInterfaceNotReady	Local interface not running	

Read interface information

Code:	AS + 950
Symbolic:	bccFbReadIF

This command will read interfaces information from the device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request all interfaces
4	U32	NODE	Node address

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 All interfaces present
4	U32	NODE	Node address (echo)
8	U32	IFTYPE	Interface type identifier ^[85] I
12	U32	NUMIF	Number of interface types (#1)
16	U32	IFTYPES0	Interface type 0 (#1)
20	U32	IFTYPES1	Interface type 1 (#1)
...

(#1) These fields are present only if the flag 0x00000001 is set in FLAGS.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address

Read NMT status from CANopen Interface

Code:	AS + 922
Symbolic:	bccFbReadCanNmt

This command will read the NMT status from the device CANopen Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x40000000 Request the ERRTEXT field
4	U32	NODE	Node address

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Required node address
8	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+8	U32	ERRCODE	Error code
+12	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flags is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+8	U32	STATUS	Actual NMT status:

Offset	Type	Label	Description
			0=Init 1=Reset node 2=Reset communication 4=Stop 5=Operational 127 = PreOperational

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address
nackInterfaceNotFound	CANopenI interface not present	
nackInterfaceNotReady	CANopen interface not running	

Read NMT status from EtherCAT Interface

Code:	AS + 912
Symbolic:	bccFbReadEcatNmt

This command will read the NMT status from the device EtherCAT Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	NODE	Node address

On success, a [bccAck](#)_[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: (none)
4	U32	NODE	Node address (echo)
8	U32	STATUS	Actual NMT status: 1=Init 2=PreOperational

Offset	Type	Label	Description
			3=Boot 4=SafeOperational 8=Operational
12	U32	FSTATUS	Forced NMT status: 1=Init 2=PreOperational 3=Boot 4=SafeOperational 8=Operational

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address
nackInterfaceNotFound	EtherCAT interface not present	
nackInterfaceNotReady	EtherCAT interface not running	

Read NMT status from Local Interface

Code:	AS + 902
Symbolic:	bccFbReadLocalNmt

This command will read the NMT status from the device Local Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x40000000 Request the ERTEXT field
4	U32	NODE	Node address

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags:

Offset	Type	Label	Description
			0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+8	U32	ERRCODE	Error code
+12	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+8	U32	STATUS	Actual NMT status: 0=Init 1=PreOperational 2=Operational

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address
nackInterfaceNotFound	Local interface not present	
nackInterfaceNotReady	Local interface not running	

Write an entry to CANopen Interface

Code:	AS + 921
Symbolic:	bccFbWriteCanEntry

This command will write an entry to the device CANopen Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index
16	DBL	VALUE	Entry value

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ^[85]
+20	DBL	VALUE	Entry value (echo)
+28	U32	ADDR	Entry address (#1)
...	DBL	DEFVAL	Entry default value (#2)
...	DBL	MINVAL	Entry minimum value (#3)
...	DBL	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	CANopen interface not present	
nackInterfaceNotReady	CANopen interface not running	

Write an entry to EtherCAT (CoE) Interface

-

Code:	AS + 911
Symbolic:	bccFbWriteCoeEntry

This command will write an entry to the device EtherCAT (CoE) Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index
16	DBL	VALUE	Entry value

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTXT field present

Offset	Type	Label	Description
			0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ⁸⁵
+20	DBL	VALUE	Entry value (echo)
+28	U32	ADDR	Entry address (#1)
...	DBL	DEFVAL	Entry default value (#2)
...	DBL	MINVAL	Entry minimum value (#3)
...	DBL	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	EtherCAT (CoE) interface not present	
nackInterfaceNotReady	EtherCAT (CoE) interface not running	

Write an entry to Local interface

Code:	AS + 901
Symbolic:	bccFbWriteLocalEntry

This command will write an entry to the device Local Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index
16	DBL	VALUE	Entry value

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags:


Offset	Type	Label	Description
			0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type 
+20	DBL	VALUE	Entry value (echo)
+28	U32	ADDR	Entry address (#1)
...	DBL	DEFVAL	Entry default value (#2)
...	DBL	MINVAL	Entry minimum value (#3)
...	DBL	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index
nackInterfaceNotFound	Local interface not present	
nackInterfaceNotReady	Local interface not running	

Write an extended entry to CANopen Interface

Code:	AS + 924
Symbolic:	bccFbWriteCanEntryE

This command will write an extended entry to the device CANopen Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index

Offset	Type	Label	Description
12	U32	SUBIX	Entry sub-index
16	U32	DTYPE	Entry data type ^[85]
20	<T>	VALUE	Entry value

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- If the (DTYPE & 0x100000) value is 0x100000, the type <T> is STRZ
- In all other cases <T> is DBL

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ^[85]
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

Otherwise the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- In all other cases <T> is DBL

On failure, a [bccNack](#) ^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags

NACK code	Description	Extra data
		2=Node address 3=Index/sub-index 4=Data type
nackInterfaceNotFound	CANopen interface not present	
nackInterfaceNotReady	CANopen interface not running	

Write an extended entry to EtherCAT (CoE) Interface

Code:	AS + 914
Symbolic:	bccFbWriteCoeEntryE

This command will write an extended entry to the device EtherCAT (CoE) Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index
16	U32	DTYPE	Entry data type ⁸⁵
20	<T>	VALUE	Entry value

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- If the (DTYPE & 0x100000) value is 0x100000, the type <T> is STRZ
- In all other cases <T> is DBL

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present 0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ^[85]
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

Otherwise the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- In all other cases <T> is DBL

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index 4=Data type
nackInterfaceNotFound	EtherCAT (CoE) interface not present	
nackInterfaceNotReady	EtherCAT (CoE) interface not running	

Write an extended entry to Local interface

Code:	AS + 904
Symbolic:	bccFbWriteLocalEntryE

This command will write an extended entry to the device Local Interface object dictionary. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Request the ADDR field 0x00000002 Request the DEFVAL field 0x00000004 Request the MINVAL field 0x00000008 Request the MAXVAL field 0x00000010 Request the TEXT field 0x40000000 Request the ERRTEXT field
4	U32	NODE	Node address
8	U32	IX	Entry index
12	U32	SUBIX	Entry sub-index
16	U32	DTYPE	Entry data type ⁸⁵
20	<T>	VALUE	Entry value

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- If the (DTYPE & 0x100000) value is 0x100000, the type <T> is STRZ
- In all other cases <T> is DBL

On success, a [bccAck](#) ¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 ADDR field present 0x00000002 DEFVAL field present 0x00000004 MINVAL field present

Offset	Type	Label	Description
			0x00000008 MAXVAL field present 0x00000010 TEXT field present 0x40000000 ERRTEXT field present 0x80000000 ERRCODE field present
4	U32	NODE	Node address (echo)
8	U32	IX	Entry index (echo)
12	U32	SUBIX	Entry sub-index (echo)
16	...	EXDATA	Extra data

The EXDATA structure is the following when the 0x80000000 flag is set in FLAGS:

Offset	Type	Label	Description
+16	U32	ERRCODE	Error code
+20	STRZ	ERRTEXT	Error text (#1)

(#1) Present only when 0x40000000 is set in FLAGS.

Otherwise, if the 0x80000000 flag is not set in FLAGS, the EXDATA structure is the following:

Offset	Type	Label	Description
+16	U32	DTYPE	Entry data type ⁸⁵
+20	...	EXDATA2	Extra data 2

If the 0x100000 is set in DTYPE, the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	STRZ	VALUE	Entry actual value

Otherwise the EXDATA2 structure is the following:

Offset	Type	Label	Description
+20	<T>	VALUE	Entry actual value
+28	U32	ADDR	Entry address (#1)
...	<T>	DEFVAL	Entry default value (#2)
...	<T>	MINVAL	Entry minimum value (#3)

Offset	Type	Label	Description
...	<T>	MAXVAL	Entry maximum value (#4)
...	STRZ	TEXT	Entry description (#5)

(#1) Present only when 0x00000001 is set in FLAGS.

(#2) Present only when 0x00000002 is set in FLAGS (offset depends of previous fields presence).

(#3) Present only when 0x00000004 is set in FLAGS (offset depends of previous fields presence).

(#4) Present only when 0x00000008 is set in FLAGS (offset depends of previous fields presence).

(#5) Present only when 0x00000010 is set in FLAGS (offset depends of previous fields presence).

Data type <T> is determined as following:

- If the (DTYPE & 0x3F) value is 0x18, the type <T> is I64
- If the (DTYPE & 0x3F) value is 0x8, the type <T> is U64
- In all other cases <T> is DBL

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Index/sub-index 4=Data type
nackInterfaceNotFound	Local interface not present	
nackInterfaceNotReady	Local interface not running	

Write interface information

Code:	AS + 951
Symbolic:	bccFbWriteIF

This command will write interface information to the device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	NODE	Node address

Offset	Type	Label	Description
8	U32	IFTYPE	Interface type identifier ⁸⁵

On success, a [bccAck](#) ¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: (none)
4	U32	NODE	Node address (echo)
8	U32	IFTYPE	Interface type identifier ⁸⁵ (echo)

On failure, a [bccNack](#) ¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=Interface type

Write NMT command to CANopen Interface

Code:	AS + 923
Symbolic:	bccFbWriteCanNmt

This command will write the NMT command to the device CANopen Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x40000000 Request the ERTEXT field
4	U32	NODE	Node address
8	U32	CMD	NMT command: 1=Request Start 2=Request Stop 128=Request to enter PreOperational 129=Request Reset node 130=Request Reset communication

On success, a [bccAck](#) ¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: (none)
4	U32	NODE	Required node address
8	U32	PSTATUS	Previous NMT status: 0=Init 1=Reset node 2=Reset communication 4=Stop 5=Operational 127=PreOperational
12	U32	CMD	NMT command (echo)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=NMT command
nackInterfaceNotFound	CANopen interface not present	
nackInterfaceNotReady	CANopen interface not running	

Write NMT command to Local Interface

Code:	AS + 903
Symbolic:	bccFbWriteLocalNmt

This command will write the NMT command to the device Local Interface. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	NODE	Node address
8	U32	CMD	NMT command: 0=Request Init

Offset	Type	Label	Description
			1=Request PreOperational 2=Request Operational

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: (none)
4	U32	NODE	Node address (echo)
8	U32	PSTATUS	Previous NMT status: 0=Init 1=PreOperational 2=Operational
12	U32	CMD	NMT command (echo)

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Node address 3=NMT Command
nackInterfaceNotFound	Local interface not present	
nackInterfaceNotReady	Local interface not running	

Field bus handling

These messages are used to handle the field bus (through the operating system) of the connected device.

CANopen interface:

- [bccCanObjRead](#)^[125], read a CANopen object
- [bccCanObjWrite](#)^[132], write a CANopen object
- [bccCanNmtRead](#)^[130], read one or more CANopen NMT

- [bccCanNmtWrite](#)^[136], write one or more CANopen NMT

- [bccCanEmcyRead](#)^[124], read a CANopen EMCY message

COE interface:

- [bccCoeObjRead](#)^[126], read a COE object
- [bccCoeObjWrite](#)^[133], write a COE object

EtherCAT interface:

- [bccEcatNmtRead](#)^[128], read a EtherCAT NMT
- [bccEcatNmtWrite](#)^[135], write a EtherCAT NMT

- [bccCanEmcyInfo](#)^[119], query CANopen EMCY message information
- [bccCanRbxChDiag](#)^[120], query Robox CANopen channel diagnostic
- [bccCanRbxWsDiag](#)^[121], query Robox CANopen workstation diagnostic
- [bccCanPdoRead](#)^[129], read element from Tx/Rx CANopen PDO
- [bccCanC402Info](#)^[118], query CANopen C402 information

Query CANopen C402 information

Code:	AS + 759
Symbolic:	bccCanC402Info

This command will query CANopen C402 information. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	NWS	No. of workstation (max 41)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	NWS	No. of WSDAT structures (max 23)
2	WSDAT	DATA1	Information for 1st workstation
8	WSDAT	DATA2	Information for 2nd workstation
...
..	WPDAT	DATA{NWS}	PDO data for Nth workstation

Each WPDAT structure have the following data:

Offset	Type	Label	Description
+0	U16	WSID	Workstation ID
+2	U16	STSW	Status word

Offset	Type	Label	Description
+4	U16	CTLW	Control word

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=No. of workstation

Query CANopen EMCY message information

Code:	AS + 755
Symbolic:	bccCanEmcyInfo

This command will query information for a CANopen EMCY (emergency) message. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Enable text
4	U16	ERRCOD	Error code
6	U8	ERRREG	Error register
7	U8[5]	MSDATA	Manufacturer specific data
12	U16	DTYPE	Drive type: 0=Generic 5=Parker-SBC (HID 1) 6=Parker-SBC (SLVD 1) 10=Sinamics (CanSin 1)

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	STRZ	TEXT	Emergency text

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Error code 3=Error register 4=Manufacturer specific data 5=Drive type

Query Robox CANopen channel diagnostic

Code:	AS + 756
Symbolic:	bccCanRbxChDiag

This command will query diagnostic information about one or more Robox CANopen channel (DS 301). Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	CHID	Channel ID
6	U16	NCH	No. of channel (max 24)

On success, a [bccAck](#)¹⁶⁴ is received with the following data:

Offset	Type	Label	Description
0	U16	NCH	No. of CHDAT structures (max 24)
2	CHDAT	DATA1	Data for 1st channel
12	CHDAT	DATA2	Data for 2nd channel
...
..	CHDAT	DATA{NCH}	Data for Nth channel

Each CHDAT structure have the following data:

Offset	Type	Label	Description
+0	U16	CHID	Channel ID
+2	U32	CHSTS	Channel status: 0x00000001 Successful boot-up operation

Offset	Type	Label	Description
			0x00000002 Successful main software load 0x00000004 Successful configuration 0x80000000 Not present channel
+6	U32	DIAG	Channel diagnostic: 0x00000001 Fault in Can Handler 1 0x00000002 Fault in Can Handler 2 0x00000004 Fault in Can Handler 3 0x00010000 Watchdog fault (immediate) 0x00020000 Watchdog fault (filtered) 0x00040000 Handshake not ready 0x00080000 Global communication fault 0x00100000 PLL lost - no synch signal is sent

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Channel ID 3=No. of channel

Query Robox CANopen workstation diagnostic

Code:	AS + 757
Symbolic:	bccCanRbxWsDiag

This command will query diagnostic information about one or more Robox CANopen workstation . Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	NWS	No. of workstation (max 41)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)¹⁶⁴ is received with the following data:

Offset	Type	Label	Description
0	U16	NWS	No. of WSDAT structures (max 41)
2	WSDAT	DATA1	Data for 1st workstation
8	WSDAT	DATA2	Data for 2nd workstation
...
..	WSDAT	DATA{NWS}	Data for Nth workstation

Each WSDAT structure have the following data:

Offset	Type	Label	Description
+0	U16	WSID	Workstation ID (#1)
+2	U32	DIAG	Workstation diagnostic: 0x00000001 Missing Rx PDO 1 0x00000002 Missing Rx PDO 2 0x00000004 Missing Rx PDO 3 0x00000008 Missing Rx PDO 4 0x00000010 Before time Rx PDO 1 0x00000020 Before time Rx PDO 2 0x00000040 Before time Rx PDO 3 0x00000080 Before time Rx PDO 4

Offset	Type	Label	Description
			0x00000100 Missing mandatory workstation
			0x00000200 Missing workstation
			0x00000400 Last request RTR failed
			0x00000800 Workstation presence imposed
			0x00001000 Reconfiguring workstation
			0x00002000 Error during reconfiguration
			0x00010000 Emergency message present
			0x00020000 No communication (in operational mode)
			0x00040000 Workstation kind : mandatory
			0x00080000 Workstation kind : reloadable on connection
			0x00100000 Configuration satisfactory done
			0x00200000 Workstation not present on first configuration
			0x00400000 Problem occurs during configuration on SDO command
			0x00800000 Problem occurs during configuration on NMT command
			0x01000000 Configuration done
			0x40000000 Workstation not started

Offset	Type	Label	Description
			0x80000000 Undefined workstation

(#1) Workstation ID is also know (or defined) as Robox ID.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=No. of workstation

Read a CANopen EMCY message

Code:	AS + 754
Symbolic:	bccCanEmcyRead

This command will read a CANopen EMCY (emergency) message. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Enable text
4	U16	WSID	Workstation ID (#1)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U16	ERRCOD	Error code
2	U8	ERRREG	Error register
3	U8[5]	MSDATA	Manufacturer specific data
8	STRZ	TEXT	Emergency text (if enabled)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

NACK code	Description	Extra data
		2=Workstation ID

Read a CANopen object

Code:	AS + 750
Symbolic:	bccCanObjRead

This command will read the specified CANopen object. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Enable text on error
4	U16	WSID	Workstation ID (#1)
6	U16	OBJIX	Object index
8	U16	SUBIX	Object sub-index
10	I16	TYPE	Data type: -1=As signed data +1=As unsigned data 5=As float

Note (#1): Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 Error on SDO
4	DBL	DATA	Object data
12	U32	ERRCOD	Error code (if error)
16	STRZ	ERRTXT	Error text (if error)

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=Object index

NACK code	Description	Extra data
		4=Object sub-index 5=Data type

Read a COE object

Code:	AS + 760
Symbolic:	bccCoeObjRead

This command will read the specified COE object. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x1 Enable text on error
4	U16	WSID	Workstation ID (#1)
6	U16	OBJIX	Object index
8	U16	SUBIX	Object sub-index
10	U16	DTYPE	Data type (container data): 1 = Boolean (U8) 2 = Integer 8 (I8) 3 = Integer 16 (I16) 4 = Integer 32 (I32) 5 = Unsigned 8 (U8) 6 = Unsigned 16 (U16) 7 = Unsigned 32 (U32) 8 = Real 32 (FLT) 9 = Visible String (STRZ) 10= Octet String (...) 11 = Unicode String (...) 12 = Time of Day (...) 13 = Time Difference (...) 15 = Domain (...) 16 = Integer 24 (I32) 17 = Real 64 (DBL)

Offset	Type	Label	Description
			18 = Integer 40 (I64)
			19 = Integer 48 (I64)
			20 = Integer 56 (I64)
			21 = Integer 64 (I64)
			22 = Unsigned 24 (U32)
			24 = Unsigned 40 (U64)
			25 = Unsigned 48 (U64)
			26 = Unsigned 56 (U64)
			27 = Unsigned 64 (U64)
			30 = Byte (U8)
			45 = BITARR8 (U8)
			46 = BITARR16 (U16)
			47 = BITARR32 (U32)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)¹⁶⁴ is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 Error on SDO
...

In case of FLAGS with 0x1 value, data is the following:

Offset	Type	Label	Description
4	U32	ERRCOD	Error code
8	STRZ	ERRTXT	Error text

In case of FLAGS with no 0x1 value, data is the following:

Offset	Type	Label	Description
4	U8	DSIZE	Size of data
5	U8	-	(reserved)
6	U8[DSIZE]	DATA	Data

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=Object index 4=Object sub-index 5=Data type

Read an EtherCAT NMT

Code:	AS + 762
Symbolic:	bccEcatNmtRead

This command will read one or more EtherCAT NMT. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	NWS	No. of workstation (max 63)(#2)

(#1) Workstation ID is also know (or defined) as Robox ID.

(#2) In case of WSID = 0 (all), the NWS field is not considered

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	NWS	No. of NMT structures (max 63)
2	NMT	DATA1	NMT data for 1st workstation
6	NMT	DATA2	NMT data for 2nd workstation
...
...	NMT	DATA{NWS}	NMT data for Nth workstation

Each NMT structure have the following data:

Offset	Type	Label	Description
+0	U16	WSID	Workstation ID (#1)
+2	U8	STSC	Status code:

Offset	Type	Label	Description
			0=Not present 1=Init 2=Pre Operational 3=Boot 4=Safe Operational 8=Operational
+3	U8	-	(reserved)

(#1) Workstation ID is also know (or defined) as Robox ID.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=No. of workstation

Read data from Tx/Rx CANopen PDO

Code:	AS + 758
Symbolic:	bccCanPdoRead

This command will read data from a Tx/Rx CANopen PDO. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 read from TxPDO 0x00000002 read from RxPDO
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	N	No. of workstation (max 41)
8	U16	PDO	PDO index
10	U16	ITEM	PDO item index

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	N	No. of WSDAT structures (max 23)
2	WSDAT	DATA1	PDO data for 1st workstation
12	WSDAT	DATA2	PDO data for 2nd workstation
...
..	WSDAT	DATA{N}	PDO data for Nth workstation

Each WSDAT structure have the following data:

Offset	Type	Label	Description
+0	U16	WSID	Workstation ID
+2	DBL	DATA	PDO data

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=No. of workstation 4=PDO index 5=PDO item index

Read one or more CANopen NMT

Code:	AS + 752
Symbolic:	bccCanNmtRead

This command will read one or more CANopen NMT. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	N	No. of workstation (max 63)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U16	N	No. of NMT structures (max 63)
2	NMT	DATA1	NMT data for 1st workstation
6	NMT	DATA2	NMT data for 2nd workstation
...
...	NMT	DATA{N}	NMT data for Nth workstation

Each NMT structure have the following data:

Offset	Type	Label	Description
+0	U16	WSID	Workstation ID (#1)
+2	U8	STSC	Status code: 4=Stop 5=Operational 127=Preoperational
+3	U8	STSF	Status flags: 0x01 Not present ws 0x02 Not physically present ws (even if forced presence) 0x04 Last RTR for APDO failed 0x08 Ws forced presence 0x10 Configuration in progress 0x20 Configuration aborted 0x80 Ws sent PDO shorter than programmed

(#1) Workstation ID is also know (or defined) as Robox ID.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID

NACK code	Description	Extra data
		3=No. of workstation

Write a CANopen object

Code:	AS + 751
Symbolic:	bccCanObjWrite

This command will write the specified CANopen object. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x00000001 Enable text on error
4	U16	WSID	Workstation ID (#1)
6	U16	OBJIX	Object index
8	U16	SUBIX	Object sub-index
10	I16	TYPE	Data type: -1=I8 +1=U8 -2=I16 +2=U16 -3=I24 +3=U24 -4=I32 +4=U32 5=float
12	DBL	DATA	Object data

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 Error on SDO
4	U32	ERRCOD	Error code (if error)
8	STRZ	ERRTXT	Error text (if error)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=Object index 4=Object sub-index 5=Data type

Write a COE object

Code:	AS + 761
Symbolic:	bccCoeObjWrite

This command will write the specified COE object. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: 0x1 Enable text on error
4	U16	WSID	Workstation ID (#1)
6	U16	OBJIX	Object index
8	U16	SUBIX	Object sub-index
10	U16	DTYPE	Data type (container data): 1=Boolean (U8) 2=Integer 8 (I8) 3=Integer 16 (I16) 4=Integer 32 (I32) 5=Unsigned 8 (U8) 6=Unsigned 16 (U16) 7=Unsigned 32 (U32) 8=Real 32 (FLT) 9=Visible String (STRZ) 10=Octet String (...) 11=Unicode String (...) 12=Time of Day (...) 13=Time Difference (...)

Offset	Type	Label	Description
			15=Domain (...) 16=Integer 24 (I32) 17=Real 64 (DBL) 18=Integer 40 (I64) 19=Integer 48 (I64) 20=Integer 56 (I64) 21=Integer 64 (I64) 22=Unsigned 24 (U32) 24=Unsigned 40 (U64) 25=Unsigned 48 (U64) 26=Unsigned 56 (U64) 27=Unsigned 64 (U64) 30=Byte (U8) 45=BITARR8 (U8) 46=BITARR16 (U16) 47=BITARR32 (U32)
12	U8	DSIZE	Size of data
13	U8	-	(reserved)
14	U8[DSIZE]	DATA	Data

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Reply flags: 0x00000001 Error on SDO
...

In case of FLAGS with 0x1 value, data is the following:

Offset	Type	Label	Description
4	U32	ERRCOD	Error code
8	STRZ	ERRTXT	Error text

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=Object index 4=Object sub-index 5=Data type 6=Data size 7=Data

Write an EtherCAT NMT

Code:	AS + 763
Symbolic:	bccEcatNmtWrite

This command will write one or more EtherCAT NMT. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	ETHID	Ethernet ID (0=dont care)
8	U8	STSC	Status code: 1=Init 2=Pre Operational 3=Boot 4=Safe Operational 8=Operational
9	U8	-	(reserved)

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)_[164] is received with no data.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=Ethernet ID

NACK code	Description	Extra data
		4=Status code

Write one or more CANopen NMT

Code:	AS + 753
Symbolic:	bccCanNmtWrite

This command will write one or more CANopen NMT. Request has the following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U16	WSID	Workstation ID (0=all) (#1)
6	U16	N	No. of workstation
8	U16	CMD	NMT command code: 1=Start 2=Stop 128=Enter preoperational 129=Reset node 130=Reset communication

(#1) Workstation ID is also know (or defined) as Robox ID.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Workstation ID 3=No. of workstation 4=NMT command code

Flash handling

These messages are used for handling flashes of the connected device.

Flash handling:

- [bccFlashInfo](#)^[153], query flash information

File handling:

[bccFlashFileLoad](#)^[142], load a file to a flash folder

- [bccFlashInfoByFolder](#)^[155], query flash information by folder
 - [bccFlashList](#)^[138], query list of flashes
 - [bccFlashFormat](#)^[141], format a flash
 - [bccFlashInit](#)^[138], define and initialize flashes
 - [bccFlashDisk](#)^[146], handle disk flashes
 - [bccFlashDir](#)^[148], query flash folder content
 - [bccFlashSetAttributes](#)^[162], set attributes in flash
- [bccFlashFileSave](#)^[159], save a file from a flash folder
- [bccFlashFileDelete](#)^[140], delete flash files
- [bccFlashFileRename](#)^[159], rename a flash file
- [bccFlashFileInfo](#)^[153], query flash file information
- Folder handling:
- [bccFlashTree](#)^[158], query flash folder tree
- [bccFlashFolderCreate](#)^[137], create a flash folder
- [bccFlashFolderDelete](#)^[140], delete a flash folder
- [bccFlashFolderInfo](#)^[156], query flash folder information

Create a flash folder

Code:	AS + 113
Symbolic:	bccFlashFolderCreate

This command will request to create a new flash folder with the specified path:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Create recursively (all PATH levels) 0x00000002 Make the folder (#1)
4	U32	ATTRIB	Flash folder attributes: 0x00000020 Hidden folder 0x00000040 Read only folder
8	STRZ	PATH	Flash folder absolute path (0-terminated, using / as folder separator)

(#1) The make flag will tell the command to be successfully even if the folder already exist.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal params	U16 What illegal: 1=Flags 2=Attributes 3=Folder path
nackFolderNotExist (nackNotFound)	Folder path (full/partially) not found	
nackReadOnly	The flash is read-only and the new folder cannot be created.	
nackFolderExist	The folder path already exist (and no make flag specified)	

Create and initialize flashes

Code:	AS + 112
Symbolic:	bccFlashInit

This command will request to create, define and initialize flashes for the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Query only effective configuration
4	U8	NFLA	Number of required flashes (min 1).
5	...	FLA0	Flash 0 request
...	...	FLA1	Flash 1 request
...	

Where FLA_x contains the following parameters:

Offset	Type	Label	Description
+0	U8[4]	NAME	Flash name (3 + 0 terminator)
+4	U32	SIZE	Required size [kbyte]

Notes:

- for standard flashes the name is /Fx, where x is @ for system flash, and from A to Z for users defined flashes
- for compact flashes use the [bccFlashDisk](#)¹⁴⁶ command.

On success, a [bccAck](#)_[164] is received with the following data:

Offset	Type	Label	Description
0	U8	NFL	Number of effective flashes
1	...	FLA0	Flash 0 effective data
...	...	FLA0	Flash 1 effective data
...	

Where FLA_x contains the following parameters:

Offset	Type	Label	Description
+0	U8x4	NAME	Flash name (3 + 0 terminator)
+4	U32	SIZE	Effective size [kbyte]

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=No. of flashes 3=Flashes 100+x=Flash x name 200+x=Flash x size
nackReadOnly	The flash (or the device itself) is read-only and cannot be initialized.	

After the [bccAck](#)_[164] is received, the initialization operation will begin as soon as possible: when operation normally end you will receive a [bccCompleted](#)_[167] message (with no data). If operation is aborted from the device, you will receive a [bccAborted](#)_[165].

Notes:

- Operation cannot be aborted by command, but only from remote.
- Both [bccCompleted](#)_[167] and [bccAborted](#)_[165] will have PID as request command PID.

During the operation you can optionally request the operation status by sending a [bccCheck](#)_[166] message (with no parameters): as DCH value you must use the SCH value from first [bccAck](#)_[164] message. The [bccAck](#)_[164] answer for a [bccCheck](#)_[166], will contain following data:

Offset	Type	Label	Description
0	U8	PERC	Operation progress (%)
1	STRZ	MSG	Optional status message

Delete a flash folder

Code:	AS + 114
Symbolic:	bccFlashFolderDelete

This command will request to delete an existing flash folder (or a hierarchy of folders in case of recursive flag):

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Delete recursively (all PATH levels) 0x00000002 Prune the folder content (#1)
4	STRZ	PATH	Flash folder absolute path (0-terminated, using / as folder separator)

(#1) The prune flag will request to remove the entire content of the folder (or any interested folder in case of recursive flag).

On success, a [bccAck](#)_[164] is received with no data.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Flags 2=Folder path
nackFolderNotExist (nackNotFound)	Folder path not found	
nackReadOnly	The folder (or the flash itself) is read-only and cannot be deleted.	
nackNotEmpty	The folder is not empty (and no prune flag specified) and cannot be deleted.	

Delete files from a flash folder

Code:	AS + 103
Symbolic:	bccFlashFileDelete

This command will delete one or more files from a flash folder on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Delete recursive 0x00000002 Delete empty folders
4	STRZ	FILEMASK	Filename mask (with jollies)

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U16	COUNT	No. of deleted files

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Folder 2=Filename 3=Flags
nackReadOnly	One or more file (or the flash or the device itself) is read-only and cannot be deleted.	

Format a flash

Code:	AS + 108
Symbolic:	bccFlashFormat

This command will begin a format (erase) a flash of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags:
4	U16	DEVID	Flash device ID

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What flags: 1=Flags

NACK code	Description	Extra data
		2=Flash device ID
nackReadOnly	The flash (or the device itself) is read-only and cannot be erased.	

On success, a [bccAck](#)^[164] is received with no data. The format operation will begin as soon as possible: when operation normally end you will receive a [bccCompleted](#)^[167] message (with no data).

If operation is aborted from the device, you will receive a [bccAborted](#)^[165].

Notes:

- Operation cannot be aborted by command, but only from remote.
- Both [bccCompleted](#)^[167] and [bccAborted](#)^[165] will have PID as request command PID.

During the operation you can optionally request the operation status by sending a [bccCheck](#)^[166] message (with no parameters): as DCH value you must use the SCH value from first [bccAck](#)^[164] message. It's [bccAck](#)^[164] answer will contain following data:

Offset	Type	Label	Description
0	U8	PERC	Operation progress (%)
1	STRZ	MSG	Optional status message

Load a file to a flash folder

Code:	AS + 100
Symbolic:	bccFlashFileLoad

This command will load a generic file from the local disk to a flash folder of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Overwrite target file 0x00000002 Recover load, if needed 0x00000004 No BIF16 handling 0x00000008 Special load as BOOT (#1) 0x00000010 Special load as BIOS (#1) 0x00000020 Special load as MAIN (#1)

Offset	Type	Label	Description
			0x00000040 Special load as CFG (#1) 0x00000080 (reserved) 0x00000100 (reserved)
4	U32	FILESIZE	Effective file length [bytes]
8	U32	ALLOADDR	Allocation address (only BIF16)
12	U32	STARTADDR	Start address (only BIF16)
16	U8	HH	Time (hour) of the file
17	U8	MM	Time (minute) of the file
18	U8	SS	Time (second) of the file
19	U8	DD	Date (day) of the file
20	U8	MO	Date (month) of the file
21	U8	YY	Date (year - 2000) of the file
22	U32	ATTR	Attributes: 0x00000002 Reserved file 0x00000004 System file 0x00000020 Hidden file 0x00000040 Read-only file 0xF0000000 File type mask: - 0x0 Generic - 0x2 OS standard - 0x3 OS test - 0x4 OS standard flat - 0x5 Old language (without OS) - 0x6 FIFO (queue)

Offset	Type	Label	Description
26	STRZ	FILENAME	Filename

(#1) Special load flags are all exclusive.

Notes:

- to load a file in a specific folder, without specifying a new filename, you need to specify the final / character: /FOLDER is the folder itself and will be not accepted (illegal arguments) while the correct target filename is /FOLDER/.

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors

NACK code	Description	Extra data
nackOpenError	Error opening flash file	
nackMemoryFull	Out of memory (or flash full)	
nackWriteError	Error writing flash file	
nackDataOverflow	Received more data than required	
nackDataUnderflow	Received less data than required	
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Folder 2=Filename 3=File size 4=Flags 5=Alloc address 6=Start address 7=Date/time 8=Attribute
nackFileExist	The flash file exist and overwrite flags has not been specified	
nackIllegalFile	The file is illegal or not recognized for special type loading	
nackReadOnly	The existing file (or the flash itself) file is read-only and cannot be created/overwritten.	

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	OFFSET	Starting load offset

Offset	Type	Label	Description
4	U16	CRC16	CRC16 of file portion from 0 to (OFFSET-1)
6	U8	BSIZE	Required data size (0=default), excluding leading U32 offset data.

NOTE: the SCH value of this bccAck will be the DCH for all subsequent commands/message for this data transfer.

At this time the device will wait a for a [bccCheck](#)₁₆₆ command to confirm operation and effective start offset. If OFFSET is greater than zero, the file CRC16 should be verified for the partial file from offset 0 to (OFFSET-1): if match, we are loading the same file and load can be recovered: otherwise, we well restart from offset 0.

The [bccCheck](#)₁₆₆ command data is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Remove temporary files
4	U32	OFFSET	Effective offset to be used

On failure, a [bccNack](#)₁₆₄ is received and the operation is aborted:

On success, a [bccAck](#)₁₆₄ is received and the transfer process can begin.

The device will send you periodically a [bccIBlock](#)₁₆₆ message, indicating how many data packet you can send to the device itself: after you transmit the required packet (or less i transmitting a [bccEndData](#)₁₆₆), you will receive a new [bccIBlock](#)₁₆₆.

The [bccIBlock](#)₁₆₆ has the following data:

Offset	Type	Label	Description
0	U8	COUNT	N. of binary packet to send

Notes:

- If you don't receive the [bccIBlock](#)₁₆₆ after an amount of time, you can decide to abort the transfer, using the [bccAbort](#)₁₆₅ command.
- On slower communication lines use 1 as COUNT value; on standard RS232 use 3 as COUNT value.

File raw content (data) will be send with the [bccData](#)₁₆₅ message (for last packet use [bccEndData](#)₁₆₅ to indicate end of data), with the following format:

Offset	Type	Label	Description
0	U32	OFFSET	Data offset
4	B[251]	DATA	Binary data to send

Notes:

- [bccData](#)_[165] (or [bccEndData](#)_[165]) messages will begin with pid = 0 and will be incremented by 1 at each message.
- DATA size is 251 (or BSIZE if specified on first request [bccAck](#)_[164]) for each [bccData](#)_[165] message but could be less for [bccEndData](#)_[165] message.
- The data transfer could be aborted at any time with a [bccAbort](#)_[165] command.
- If required file is empty, send directly [bccNoData](#)_[166] message and wait for end of transfer.

After you send the [bccEndData](#)_[165] (or [bccNoData](#)_[166]) message you should wait the [bccCompleted](#)_[167] message that report what the device has received, with the following format:

Offset	Type	Label	Description
0	U32	SIZE	Total file size [bytes]
4	U32	COUNT	No. of total binary packet received

Notes:

- If this message is not received in an amount of time, you can consider the transfer aborted (not confirmed). If the remote device need more time in order to complete the operation, it can periodically send the [bccWait](#)_[167] message, until the operation completes.
- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)_[167] messages: when the limit is reached, the operation will expire.
- The [bccCompleted](#)_[167] indicate that also all pending operations (for examples reload in memory or restarting) has been successfully completed.

At any time the device can decide to abort the transfer (not when you already are in abort condition). In this case you will receive the notification via the [bccAborted](#)_[165] message.

Manage flash volumes

Code:	AS + 117
Symbolic:	bccFlashDisk

This command will manage flash volumes (for special devices, like compact flashes) for the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: (none)
4	U16	CMD	Command: 1=Create a new flash 2=Delete an existing flash 3=Clear the physical device, by deleting all contained flashes (or partitions)

Offset	Type	Label	Description
			4=Backup an existing flash 5=Restore an existing flash
6	U8[32]	DATA	Command specific data
38	STRZ	NAME	Flash name (0-terminated) or physical device name.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U8[32]	DATA	Command specific reply data

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Flags 2=Command 3=Flash/device name
nackReadOnly	The flash (or the device itself) is read-only and cannot be writtten.	
nackUnformattedDevice	The flash (or the device itself) is not correctly formatted.	

For command 1 (Create a new flash) request data is the following:

Offset	Type	Label	Description
+0	U32	FLAGS	Command flags: 0x00000001 Query only effective configuration 0x00000002 Overwrite existing flash
+4	U32	SIZE	Required flash size [kbyte]

For command 1 (Create a new flash) reply data is the following:

Offset	Type	Label	Description
+0	U32	SIZE	Effective flash size [kbyte]

After the [bccAck](#)^[164] is received, the initialization operation will begin as soon as possible: when operation normally end you will receive a [bccCompleted](#)^[167] message (with no data). If operation is aborted from the device, you will receive a [bccAborted](#)^[165].

Notes:

- Operation cannot be aborted by command, but only from remote.
- Both [bccCompleted](#)^[167] and [bccAborted](#)^[165] will have PID as request command PID.

During the operation you can optionally request the operation status by sending a [bccCheck](#)^[166] message (with no parameters): as DCH value you must use the SCH value from first [bccAck](#)^[164] message. The [bccAck](#)^[164] answer for a [bccCheck](#)^[166], will contain following data:

Offset	Type	Label	Description
0	U8	PERC	Operation progress (%)
1	STRZ	MSG	Optional status message

Query contents from a flash folder

Code:	AS + 102
Symbolic:	bccFlashDir

This command will collect information and content of a flash folder of the connected device: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Absolute paths 0x00000002 Recursive 0x00000004 List folders 0x00000008 List files 0x00000030 Mask for date/time type: - 0x0 (default) - 0x1 Creation - 0x2 Last modification

Offset	Type	Label	Description
			- 0x3 Last access 0x00000040 Fully qualified paths (implies absolute paths flag) 0x00000080 List hidden files/folders 0x00000100 Dereference symbolic links (#1)
4	STRZ	FILEMASK	Folder and/or filename mask (with jollies)

Note (#1): when showing file information for a symbolic link, show information for the file the link references rather than for the link itself.

If initial request fails, [bccNack](#)₁₆₄ is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Folder 2=Filename 3=Flags

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U8	HH	Time (hour) of the item
1	U8	MM	Time (minute) of the item
2	U8	SS	Time (second) of the item
3	U8	DD	Date (day) of the item
4	U8	MO	Data (month) of the item
5	U8	YY	Data (year - 2000) of the item
6	U32	ATTRIB	Attributes: 0x00000001 Absolute folder paths 0x00000002 Reserved file

Offset	Type	Label	Description
			0x00000004 System file 0x00000008 Illegal file 0x00000010 File in use (left open) 0x00000020 Hidden file 0x00000040 Read-only file 0x00000080 Symbolic link 0xF0000000 File type - 0x0 Generic - 0x1 Folder (special file) - 0x2 OS standard - 0x3 OS test - 0x4 OS standard flat - 0x5 Old language (without OS) - 0x6 FIFO (queue)
10	U16	DEVID	Flash device ID
12	U16	CRC16	CRC16 file value (0 no information available)
14	U16	-	Reserved
16	U32	SIZE	Effective item size [bytes]
20	STRZ	FILENAME	Folder or filename

Notes:

- [bccData](#)_[165] messages will begin with pid = 0 and will be incremented by 1 at each message.
- [bccEndData](#)_[165] contain last item and after it the transfer is completed
- The data transfer could be aborted at any time with a [bccAbort](#)_[165] command.

Query information of a file in a flash folder

Code:	AS + 110
Symbolic:	bccFlashFileInfo

This command will request information about a flash file of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Generate absolute paths 0x00000006 Mask for date/time type: - 0x0 (default) - 0x1 Creation - 0x2 Last modification - 0x3 Last access 0x00000008 Fully qualified filename (implies absolute paths flag) 0x00000010 Dereference symbolic links (#1)
4	STRZ	FILENAME	Filename (with opt. folder)

Note (#1): when showing file information for a symbolic link, show information for the file the link references rather than for the link itself.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U8	HH	Time (hour) of the item
1	U8	MM	Time (minute) of the item
2	U8	SS	Time (second) of the item
3	U8	DD	Date (day) of the item
4	U8	MO	Date (month) of the item
5	U8	YY	Date (year - 2000) of the item
6	U32	ATTRIB	Attributes: 0x00000001 Absolute path 0x00000002 Reserved file

Offset	Type	Label	Description
			0x00000004 System file 0x00000008 Illegal file 0x00000010 File in use (left open) 0x00000020 Hidden file 0x00000040 Read-only file 0x00000080 Symbolic link 0xF0000000 File type mask: - 0x0 Generic - 0x1 Folder (special file) - 0x2 OS standard - 0x3 OS test - 0x4 OS standard flat - 0x5 Old language (without OSF) - 0x6 FIFO (queue)
10	U16	DEVID	Flash device ID
12	U16	CRC16	CRC16 file value (0=no info available)
14	U16	-	Reserved
16	U32	SIZE	Effective file size [bytes]
20	STRZ	FILENAME	Filename

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackFileNotExist (nackNotFound)	File not existing	
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Folder 3=Filename

Query information of a flash

Code:	AS + 109
Symbolic:	bccFlashInfo

This command will request information about a flash of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Search by name 0x00000002 Query underlying physical device name
4	U16	DEVID	Flash device ID
6	STRZ	NAME	Flash name (if search by name required)

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	TYPE	Device type 0x0000 Generic 0x0001 Standard flash (Robox) 0x0002 Limited flash (Robox) 0x0003 Compact flash/SD/MicroSD (< 4GiB) 0x0004 Ramdisk (Robox) 0x0005 Larger SD/MicroSD (>= 4GiB)
2	U16	DEVID	Flash device ID
4	B[32]	DATA	Data according TYPE
36	STRZ	NAME	Flash name (if required) or physical device name

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags

NACK code	Description	Extra data
		2=Flash device ID 3=Flash name

For standard flash (type 0x0001) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For limited flash (type 0x0002) data are following:

Offset	Type	Label	Description
+0	U32	FREE	Flash free space [bytes]
+4	B[28]		(reserved)

For compact flash/SD/MicroSD (< 4GiB; type 0x0003) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For ramdisk (type 0x0004) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For larger SD/MicroSD (>= 4Gib; type 0x0005) data are following:

Offset	Type	Label	Description
+0	U64	SIZE	Flash size [bytes]
+8	U64	FREE	Flash free space [bytes]
+16	B[16]		(reserved)

Query information of a flash by folder

Code:	AS + 116
Symbolic:	bccFlashInfoByFolder

This command will request information about the corresponding flash device of the specified folder. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: (none)
4	STRZ	PATH	Flash folder path (0-terminated, absolute, using / as folder separator)

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	TYPE	Flash device type 0x0000 Generic 0x0001 Standard flash (Robox) 0x0002 Limited flash (Robox) 0x0003 Compact flash/SD/MicroSD (< 4GiB) 0x0004 Ramdisk (Robox) 0x0005 Larger SD/MicroSD (>= 4GiB)
2	U16	DEVID	Flash device ID
4	B[32]	DATA	Data according TYPE
36	STRZ	NAME	Fully qualified flash name

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags 2=Flash device ID 3=Flash name

For standard flash (type 0x0001) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For limited flash (type 0x0002) data are following:

Offset	Type	Label	Description
+0	U32	FREE	Flash free space [bytes]
+4	B[28]		(reserved)

For compact flash/SD/MicroSD (< 4GiB; type 0x0003) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For ramdisk (type 0x0004) data are following:

Offset	Type	Label	Description
+0	U32	SIZE	Flash size [bytes]
+4	U32	FREE	Flash free space [bytes]
+8	B[24]		(reserved)

For larger SD/MicroSD (>= 4Gib; type 0x0005) data are following:

Offset	Type	Label	Description
+0	U64	SIZE	Flash size [bytes]
+8	U64	FREE	Flash free space [bytes]
+16	B[16]		(reserved)

Query information of a flash folder

Code:	AS + 115
Symbolic:	bccFlashFolderInfo

This command will request information about an existing flash folder. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000003 Mask for date/time type: - 0x0 (default) - 0x1 Creation - 0x2 Last modification - 0x3 Last access
4	STRZ	PATH	Flash folder absolute path (0-terminated, using / as folder separator)

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U8	HH	Time (hour) of the folder
1	U8	MM	Time (minute) of the folder
2	U8	SS	Time (second) of the folder
3	U8	DD	Date (da) of the folder
4	U8	MO	Date (month) of the folder
5	U8	YY	Date (year - 2000) of the folder
6	U32	ATTRIB	Folder attributes: 0x00000002 Reserved folder 0x00000004 System folder 0x00000008 Illegal folder 0x00000020 Hidden folder 0x00000040 Read only folder
10	U16	DEVID	Flash device ID

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackFolderNotExist (nackNotFound)	Folder path not found	
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Flags 2=Folder path

Query list of flashes

Code:	AS + 111
Symbolic:	bccFlashList

This command will request a list of available flashes of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: (none)

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U8	COUNT	N. of device
1	U16	DEVID0	Flash device 0 ID
3	U16	DEVID1	Flash device 1 ID
...

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Flags

Query tree of flash folders

Code:	AS + 107
Symbolic:	bccFlashTree

This message is obsolete, use [bccFlashDir](#)^[148] with appropriate settings flag.

Rename a file in a flash folder

Code:	AS + 104
Symbolic:	bccFlashFileRename

This command will rename a file in a flash folder of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Overwrite target file
4	U8	SLEN	Source length (+1)
5	U8	TLEN	Target length (+1)
6	STRZ	SOURCE	Source filename (with opt. folder)
...	STRZ	TARGET	Target filename (with no folder)

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackFileNotExist (nackNotFound)	Source folder and/or filename not existing	
nackFileExist	Target filename already existing	
nackIllegalArgs	Illegal parameters	U16 What flags: 1=Flags 2=Source folder 3=Source filename 4=Target folder 5=Target filename
nackReadOnly	The target file (or the flash itself) is read-only and cannot be overwritten.	

Save a file from a flash folder

Code:	AS + 101
Symbolic:	bccFlashFileSave

This command will save a generic file from a flash folder of the connected device to the local disk. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Overwrite target file 0x00000002 Recover save, if needed 0x00000004 No BIF16 handling
4	U32	OFFSET	Starting required offset
8	U16	CRC16	Calculated CRC16 (from 0 to OFFSET-1), if OFFSET greater than zero.
10	STRZ	FILENAME	Filename

On failure, a [bccNack](#)₁₆₄ is received with following errors:

NACK code	Description	Extra data
nackFileNotExist (nackNotFound)	Flash file not exist	
nackOpenError	Error opening the flash file	
nackReadError	Error reading the flash file	
nackMemoryFull	Out of memory (or general memory problem)	
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Folder 2=Filename 3=Flags 4=Offset 5=Crc16

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U32	OFFSET	Applied starting load offset
4	U32	FILESIZE	Effective file size [bytes]
8	U32	ALLOCADDR	Allocation address (only BIF16)

Offset	Type	Label	Description
12	U32	STARTADDR	Start address (only BIF16)
16	U8	HH	Time (hour) of the file
17	U8	MM	Time (minute) of the file
18	U8	SS	Time (second) of the file
19	U8	DD	Date (day) of the file
20	U8	MO	Date (month) of the file
21	U8	YY	Data (year - 2000) of the file
22	U32	ATTR	Attributes: 0x00000002 Reserved file 0x00000004 System file 0x00000020 Hidden file 0x00000040 Read-only file 0xF0000000 File type mask: - 0x0 Generic - 0x2 OS standard - 0x3 OS test - 0x4 OS standard flat - 0x5 Old language (without OS) - 0x6 FIFO (queue)

If required starting offset cannot be applied (or CRC16 does not match), returned applied offset is zero.

NOTE: the SCH value of this bccAck will be the DCH for all subsequent commands/message for this data transfer.

In order to receive data from the device, you should send periodically a [bccIBlock](#)₁₆₆₁ message, indicating how many data packet the device can send you. After you received required number of packet (or less if receiving a [bccEndData](#)₁₆₆₁), you can transmit a new [bccIBlock](#)₁₆₆₁.

The [bccIBlock](#)₁₆₆₁ has the following data:

Offset	Type	Label	Description
0	U8	COUNT	N. of binary packet to send

Notes:

- If you send correctly [bccIBlock](#)^[165] but, after an amount of time, you don't receive any data, you can decide to abort the transfer, using the [bccAbort](#)^[165] command.
- On slower communication lines use 1 as COUNT value; on standard RS232 use 3 as COUNT value.

File raw content (data) will be trasmitte by the device with the [bccData](#)^[165] message (for last packet use [bccEndData](#)^[165] to indicate end of data), with the following format:

Offset	Type	Label	Description
0	U32	OFFSET	Data offset
4	B[]	DATA	Binary data to send

Notes:

- [bccData](#)^[165] (or [bccEndData](#)^[165]) messages will begin with pid = 0 and will be incremented by 1 at each message.
- DATA size is in range 1-251 for each [bccData](#)^[165] message but could be less for [bccEndData](#)^[165] message.
- The data transfer could be aborted at any time with a [bccAbort](#)^[165] command.
- If required file is empty, you will receive directly [bccNoData](#)^[165] message.

After you receive [bccEndData](#)^[165] (or [bccNoData](#)^[165]) message you should send a [bccCompleted](#)^[167] message (if all things are right) to report that the file transfer has been completed: if you need more time before sending this message, please periodically send a [bccWait](#)^[167] message to inform the connected device about the delay. The [bccCompleted](#)^[167] data are:

Offset	Type	Label	Description
0	U32	SIZE	Total file size [bytes]
4	U32	COUNT	No. of total binary packet received

Notes:

- Some BCC/31 implementation have limit on the total delay time for [bccWait](#)^[167] messages: when the limit is reached, the operation will expire.

At any time the device can decide to abort the transfer. In this case you will receive the notification via the [bccAborted](#)^[165] message.

Set attributes in a flash folder

Code:	AS + 118
Symbolic:	bccFlashSetAttributes

This command will set multiple flash object (file/folder) attributes on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operating flags: 0x00000001 Absolute paths 0x00000002 Recursive 0x00000004 Set folders 0x00000008 Set files 0x00000010 Set hidden files/folders
4	U32	ATTRIB	Attributes to set: 0x00000004 System file 0x00000020 Hidden file 0x00000040 Read-only file
8	U32	ATTRIBM	Attributes mask (see ATTRIB)
12	STRZ	FILEMASK	Folder and/or filename mask (with jollies)

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U32	COUNT	Number of changed objects (file/folders)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal argument	U16 What illegal: 1=Flags 2=Attributes 3=Attributes mask 4=Filemask
nackReadOnly	The flash (or the device itself) is read-only and cannot be writtten.	

General handling

These message are used for general purpose in the BCC protocol.

- [bccAck](#)^[164], General acknowledge
- [bccNack](#)^[164], General NOT acknowledge
- [bccData](#)^[165], Binary data
- [bccAbort](#)^[165], Abort command
- [bccAborted](#)^[165], Abort event
- [bccEndData](#)^[166], End of data
- [bccNoData](#)^[166], No data
- [bccIBlock](#)^[166], Software inter-block
- [bccCheck](#)^[166], Check-point command
- [bccBusy](#)^[166], System busy event
- [bccCompleted](#)^[167], Completion event
- [bccWait](#)^[167], Wait more
- [bccReady](#)^[167], Ready

General acknowledge

Code:	1
Symbolic:	bccAck

General acknowledge message, usually used as reply message of another request message: some time can contain data.

General NOT acknowledge

Code:	2
Symbolic:	bccNack

General not acknowledge message, usually used as negative reply message of another request message. Data area is the following:

Off	Type	Description
0	U16	Original request message code, to wich the error is referred
2	U16	Nack error code ^[283]
4	U16	Nack extra data (optional, not for all messages)

NOTE: to validate a NACK message, you should even consider area data greater/equal to 4 bytes. If a NACK provide extra information and they are not received (but a least four initial bytes) the message is valid too, but extra information are declared 'not provided' (for example from old BCC implementation).

Some common NACK error codes:

NACK code	Description	Extra data
nackWrongCommand	The command in not implemented	
nackMissingArgs	One or more parameter missing	

NACK code	Description	Extra data
nackNotAuthorized	The command could be valid but current authentication does not allow it (see network interfaces ^[272]).	
nackIllegalMode	The command is valid and authorized, but current mode/status does not permit it to be processed.	
nackResourceBusy	Required resource is not available at this moment.	

Binary data

Code:	3
Symbolic:	bccData

General message to transport binary / general purpose data.

Abort command

Code:	AS+4
Symbolic:	bccAbort

General command to request abort of current operation, in a specific usage context. If while waiting bccAck/bccNack for the command, you received the [bccAborted](#)^[165] notification you should even wait for bccAck or bccNack, but result o abort will be bccAck.

Request data area is the following:

Off	Type	Description
0	U16	Abort code, usually a standard nack error code ^[283]
2	U16	Nack extra data: value depends on context usage.
4	STRZ	Optional error string (0 terminated)

Abort event

Code:	5
Symbolic:	bccAborted

General information about an abort event: data area is the following, but can have additional information depending the usage context:

Off	Type	Description
0	U16	Abort code, usually a standard nack error code ^[283]
2	U16	Nack extra data: value depends on context usage.
4	STRZ	Optional error string (0 terminated)

End of data

Code:	6
Symbolic:	bccEndData

When this message is used instead of [bccData](#)^[165], it mean that there are no more data that can be received.

No data

Code:	7
Symbolic:	bccEndData

When this message is used instead of [bccData](#)^[165], it mean that there is not data to be received.

Software inter-block

Code:	8
Symbolic:	bccIBlock

This message is usually used as software inter-block during data transfer to specifying how many [bccData](#)^[165] can be sent to the receiver in a single shoot. It's data area depend on usage context.

Check-point command

Code:	AS+9
Symbolic:	bccCheck

This command is used as general purpose check command, to verify something.
Optional data area on command, answer data and nack codes , depend on context usage.

System busy event

Code:	10
Symbolic:	bccBusy

This message is usually used instead of [bccNack](#), indicating that the receiver cannot handle the command because is busy to do something else: user can way an amount of time and retry the command or operation.

Completion event

Code:	11
Symbolic:	bccCompleted

This message is usually used to indicate that an operation or request has been completed: it's data area depend on usage context.

Wait more

Code:	12
Symbolic:	bccWait

This message is usually used to indicate that an operation or request need more time in order to completed: it's data area depend on usage context.

Ready

Code:	13
Symbolic:	bccReady

This message is usually used to indicate that something is ready in an operation: it's data area depend on usage context.

I/O handling

NOTE: although defined and used on some hardware Robox device, these command are declared deprecated and are not supported by RTE: we suggest to use [variables](#) ^[249] instead.

These message are used getting and setting I/O value to a connected device.

Get commands:

- [bccGetIC](#) ^[171], get input channel
- [bccGetIW16](#) ^[171], get input word 16bit
- [bccGetOC](#) ^[172], get output channel
- [bccGetOW16](#) ^[173], get output word 16bit

Set commands:

- [bccSetOC](#) ^[177], set output channel
- [bccSetOW16](#) ^[178], set output word 16bit

Force commands:

- [bccForceIC](#) ^[168], force input channel
- [bccForceIW16](#) ^[168], force input word 16bit
- [bccForceOC](#) ^[169], force output channel
- [bccForceOW16](#) ^[170], force output word16bit

Release commands

- [bccReleaseIC](#) ^[175], release input channel
- [bccReleaseIW16](#) ^[175], release input word 16bit
- [bccReleaseOC](#) ^[176], release output channel
- [bccReleaseOW16](#) ^[176], release output word16bit
- [bccReleaseAllIC](#) ^[174], release all input channels

- [bccReleaseAllOC](#)^[174], release all output channels

Force input channel

Code:	AS + 206
Symbolic:	bccForceIC

NOTE: this command is **not supported** in RTE firmware.

This command will force the state of an input channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)
2	U8	STATE	Required channel logical state (0=Off/Open, otherwise On/Close)

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackReadOnly	Channel is not writable	

Force input word 16bit

Code:	AS + 207
Symbolic:	bccForceIW16

NOTE: this command is **not supported** in RTE firmware.

This command will force the state of multiple 16 bit input word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive input words (#1)

Offset	Type	Label	Description
1	U16	INDEX	Required input word first index (see your hardware documentation for valid index values)
3	U16	VAL0	Value for input word 0
5	U16	VAL1	Value for input word 1
7

Note (#1): data size is calulated as $3 + (2 \text{ bytes} * \text{REP})$. Max number of REP is 126.

On success, a [bccAck](#)₁₆₄₁ is received with no data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of input words 2=Iiput word first index
nackReadOnly	Input word is not writable	U16 Index of input word in error

Force output channel

Code:	AS + 208
Symbolic:	bccForceOC

NOTE: this command is **not supported** in RTE firmware.

This command will force the state of an ouput channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)
2	U8	STATE	Requeired channel logical state (0=Off/Open, otherwise On/Close)

On success, a [bccAck](#)₁₆₄₁ is received with no data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackReadOnly	Channel is not writable	

Force output word 16bit

Code:	AS + 209
Symbolic:	bccForceOW16

NOTE: this command is **not supported** in RTE firmware.

This command will force the state of multiple 16 bit output word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive output words (#1)
1	U16	INDEX	Required output word first index (see your hardware documentation for valid index values)
3	U16	VAL0	Value for output word 0
5	U16	VAL1	Value for output word 1
7

*Note (#1): data size is calculated as 3 + (2 bytes * REP). Max number of REP is 126.*

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of output words 2=Output word first index
nackReadOnly	Output word is not writable	U16 Index of output word in error

Get input channel

Code:	AS + 200
Symbolic:	bccGetIC

NOTE: this command is **not supported** in RTE firmware.

This command will get value for a logical input channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)

On success, a [bccAck](#)₁₆₄₁ is received with following data:

Offset	Type	Label	Description
0	U8	PSTATE	Channel physical state (0=Off/Open, otherwise On/Close)
1	U8	LSTATE	Channel logical state (0=Off/Open, otherwise On/Close)

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackWriteOnly	Channel is not readable	

Get input word 16bit

Code:	AS + 201
Symbolic:	bccGetIW16

NOTE: this command is **not supported** in RTE firmware.

This command will get value for multiple 16 bit input word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive input words (#1)

Offset	Type	Label	Description
1	U16	INDEX	Index of first input word required (see your hardware documentation for valid index values)

Note (#1): maximum value for REP in BCC/31 is 63.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U16	PSTATE0	Physical state of input word 0
2	U16	LSTATE0	Logical state of input word 0
4	U16	PSTATE1	Physical state of input word 1
6	U16	LSTATE1	Logical state of input word 1
8

Notes:

- data size is calculated as 4 bytes * REP.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of input words 2=Input words first index
nackWriteOnly	Input word is not readable	U16 Index of input word in error
nackNotAuthorized		

Get output channel

- (deprecated)

Code:	AS + 202
Symbolic:	bccGetOC

NOTE: this command is **not supported** in RTE firmware.

This command will get value for a logical output channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U8	PSTATE	Channel physical state (0=Off/Open, otherwise O/Close)
1	U8	LSTATE	Channel logical state (0=Off/Open, otherwise O/Close)

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackWriteOnly	Output channel is not readable	

Get output word 16bit

Code:	AS + 203
Symbolic:	bccGetOW16

NOTE: this command is **not supported** in RTE firmware.

This command will get value for multiple 16 bit output word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive output words (#1)
1	U16	INDEX	Index of first output word required (see your hardware documentation for valid index values)

Note (#1): maximum value for REP in BCC3 is 63.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U16	PSTATE0	Physical state of output word 0
2	U16	LSTATE0	Logical state of output word 0
4	U16	PSTATE1	Physical state of output word 1
6	U16	LSTATE1	Logical state of output word 1
8

Notes:

- data size is calculated as 4 bytes * REP.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of output words 2=Output word first index
nackWriteOnly	Output word not readable	U16 Index of output word in error

Release all input channel

Code:	AS + 214
Symbolic:	bccReleaseAllIC

NOTE: this command is **not supported** in RTE firmware.

This command will release ALL forced logical input channel. Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received.

Release all output channel

Code:	AS + 215
Symbolic:	bccReleaseAllOC

NOTE: this command is **not supported** in RTE firmware.

This command will release ALL forced logical output channel. Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received.

Release input channel

Code:	AS + 210
Symbolic:	bccReleaseIC

NOTE: this command is **not supported** in RTE firmware.

This command will release a forced logical input channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackReadOnly	Channel is not writable	

Release input word 16bit

Code:	AS + 211
Symbolic:	bccReleaseIW16

NOTE: this command is **not supported** in RTE firmware.

This command will release the state of multiple 16 bit input word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive input words (#1)
1	U16	INDEX	Index of first required input word (see your hardware documentation for valid index values)

Note (#2): maximum number of REP is 126.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of input words 2=Input word first index
nackReadOnly	Input word is not writable	U16 Index of output word in error

Release output channel

Code:	AS + 212
Symbolic:	bccReleaseOC

NOTE: this command is **not supported** in RTE firmware.

This command will release a forced logical output channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index
nackReadOnly	Channel is not writable	

Release output word 16bit

Code:	AS + 213
Symbolic:	bccReleaseOW16

NOTE: this command is **not supported** in RTE firmware.

This command will release the state of multiple 16 bit output word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive output words (#1)
1	U16	INDEX	Index of first required output word (see your hardware documentation for valid index values)

Note (#1): maximum number of REP is 126.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of output words 2=Output word first index
nackReadOnly	Output word is not writable	U16 Index of output word in error

Set output channel

Code:	AS + 204
Symbolic:	bccSetOC

NOTE: this command is **not supported** in RTE firmware.

This command will set value for a output channel. Request parameters are the following:

Offset	Type	Label	Description
0	U16	CH	Required channel index (see your hardware documentation for valid channel values)
2	U8	STATE	Required state (0=Off/Open, otherwise On/Close)

On success, a [bccAck](#)₁₆₄ is received without data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Channel index

NACK code	Description	Extra data
		2=Channel state
nackReadOnly	Channel is not writable	

Set output word 16bit

Code:	AS + 205
Symbolic:	bccSetOW16

NOTE: this command is **not supported** in RTE firmware.

This command will set value for multiple 16 bit output word. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	No. of consecutive output words (#1)
1	U16	INDEX	Index of first output word required (see your hardware documentation for valid index values)
3	U16	MASK0	Mask of settable bits for output word 0
5	U16	VAL0	Value for output word 0
7	U16	MASK1	Mask of settable bits for output word 1
9	U16	VAL1	Value for output word 1
11

Note (#1): maximum value for REP in BCC3 is 63.

Note (#2): data size is calculated as 3 bytes + (4 bytes * REP).

When setting a value for output word X, the effective value is written as (considering ACT_VALUE as current output word value):

```
// Pseudo code with C syntax
VALUE = (ACT_VALUE & ~MASKx) | (VALx & MASKx)
```

On success, a [bccAck](#)¹⁶⁴ will be received with no data.

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal

NACK code	Description	Extra data
		1=No. of output words 2=Output word first index 3=Value U16 Bad value index, from 0 to REP-1
nackReadOnly	Output word is not writable	U16 Index of output word in error

Ladder diagram handling

These messages are used to handle ladder diagram activities, for a connected device.

Task handling:

- [bccLadTaskLoad](#)^[180], load ladder task to memory
- [bccLadTaskSave](#)^[184], save ladder task from memory

Live changes handling:

- [bccLadLiveLoad](#)^[181], load live changes
- [bccLadLiveTest](#)^[187], start live changes testing
- [bccLadLiveConfirm](#)^[180], confirm live changes
- [bccLadLiveCancel](#)^[179], cancel live changes
- [bccLadLiveWd](#)^[189], watch-dog for Live changes

Monitor handling:

- [bccLadMonStart](#)^[184], start ladder monitor
- [bccLadMonRestart](#)^[183], restart ladder monitor
- [bccLadMonStop](#)^[188], stop ladder monitor
- [bccLadMonWd](#)^[188], watch-dog for Ladder monitor
- [bccLadMonStatus](#)^[182], Query ladder monitor status

Cancel live changes

Code:	AS + 805
Symbolic:	bccLadLiveCancel

This command will cancel live changes testing for a specific session. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal:

NACK code	Description	Extra data
		1=Session ID
nackNotFound	Live changes session not found	

Confirm live changes

Code:	AS + 804
Symbolic:	bccLadLiveConfirm

This command will confirm and apply live changes for a specific session. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID
nackNotFound	Live changes session not found	

Load a ladder task to memory

Code:	AS + 800
Symbolic:	bccLadTaskLoad

This command will load the ladder task (the .LAD file itself) directly to device memory, replacing the existing one: this is a standard [data load sequence](#)₁₈.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	SIZE	Ladder task size [byte]
4	U8[12]	-	(riservati)
16	U16	PID	Process ID
18	U32	FLAGS	Operating flags: (none)

If initial request fails, [bccNack](#)₁₆₄ is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Task size 2=Process ID 3=Flags
nackNotFound	Process not found	

NOTE: to have more information about the transferred data, see the .LAD File Specifications in the ladder diagram documentation.

Load live changes

Code:	AS + 802
Symbolic:	bccLadLiveLoad

This command will load live changes for a program directly to device memory: this is a standard standard [data load sequence](#)^[18].

If the command is accepted, the device will open a live change session that can be started with the [bccLadLiveTest](#)^[187] message and kept alive with the [bccLadLiveWd](#)^[189] message: this session must be stopped with [bccLadLiveConfirm](#)^[180] or [bccLadLiveCancel](#)^[179] before it expire.

For a single PID, while its live change session is in testing (active) it is not possible to load any other changes: otherwise any further load will replace current changes by creating a new live changes session. Therefore, any PID can have only one live changes session.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	SIZE	Live changes size [byte]
4	U8[12]	-	(riservati)
16	U16	PID	Process ID
18	U32	FLAGS	Operating flags: (none)

If initial request fails, [bccNack](#)^[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Live changes size 2=Process ID 3=Flags
nackNotFound	Process not found	

ACKDATA structure is the following:

Offset	Type	Label	Description
0	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data.
1	U8[15]		(riservati)
16	U32	SESSID	Live changes session ID

NOTE: to have more information about the transferred data, see the [.LAD File Specifications](#) in the ladder diagram documentation.

Query ladder monitor status

Code:	AS + 814
Symbolic:	bccLadMonStatus

This command will query current status of specified ladder monitor and miscellaneous info for the monitored process itself. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	MONID	Monitor ID

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U8	STATUS	Current monitor status: 0=Monitor not defined 1=Monitor active (with data) 2=Monitor active (with no data) 3=Monitor expired (timer WD denied)
1	U16	FREQ	Data stream frequency (0=data sent when possible) [hz]
3	U8	PTYPE	Process type: 0=Synchronous 1=High priority 2=Normal priority 3=Low priority

Offset	Type	Label	Description
4	U16	PFREQ	Process frequency [hz]
6	U32	PSTS	Process status: 0x00000001 Process is running 0x00000002 Process has runtime error(s)
10	DBL	PLENGTH	Process length [us]

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Owner ID 2=Process ID
nackNotFound	Monitor not found	

Restart ladder monitor

Code:	AS + 811
Symbolic:	bccLadMonRestart

This command will restart the specified ladder monitor with different settings. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	MONID	Monitor ID
8	U32	RUNGID	First rung ID
12	U16	COUNT	No. of consecutive rungs

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=No. of consecutive rungs 2=Owner ID 3=Monitor ID 4=First rung ID

NACK code	Description	Extra data
nackNotFound	Monitor not found	

Save a ladder task from memory

Code:	AS + 801
Symbolic:	bccLadTaskSave

This command will save the ladder task (as a .LAD file) directly from device memory: this is a standard [data save sequence](#)^[16].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U8	BSIZE	Required item size (0=default), excluding leading U32 offset data.
1	U8[15]	-	(riservati)
16	U16	PID	Process ID
18	U32	FLAGS	Operating flags: (none)

If initial request fails, [bccNack](#)^[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Process ID 2=Flags
nackNotFound	Process not found	

NOTE: to have more information about the transferred data, see the .LAD File Specifications in the ladder diagram documentation.

Start ladder monitor

Code:	AS + 810
Symbolic:	bccLadMonStart

This command will create and start a special monitor for a specific ladder process. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID

Offset	Type	Label	Description
4	U16	PID	Process ID
6	U32	RUNGID	First rung ID
10	U16	COUNT	No. of consecutive rungs
12	U32	TIMEWD	Initial watchdog time [ms]

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	MONID	Monitor ID

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Process ID 2=No. of rungs 3=Watchdog time 4=Owner ID 5=First rung ID
nackNotFound	Process not found	
nackOutOfResource	No more resource to create and start a new ladder monitor	

Once started, the monitor will continue to send [bccData](#)^[165] messages, containing data for each of requested rungs (or a single [bccNoData](#)^[166] if no rungs or data to send): transmission frequency is decided by the device.

Each [bccData](#)^[165] message has have a progressive PID, starting from 0 at first data message and has the following structure (overall cannot exceed 255 bytes total):

Offset	Type	Label	Description
0	...	RUNG0	Data for rung 0
...	...	RUNG1	Data for rung 1
...

Each RUNGx field is structured as:

Offset	Type	Label	Description
+0	U8	SIZE	Rung data size (SIZE included)
+1	U32	ID	Rung ID

Offset	Type	Label	Description
+2	U8	RTF	Runtime Flags 0x01 Rung not executed 0x02 Rung with error(s)
+3	U8	NBF	No. of BITS fields (min 1)
+6	U8	BITS0	Boolean data 0 bit0 = rung general state
+7	U8	BITS1	Boolean data 1 (optional)
...
...	...	VAL0	Value data 0 (optional)
...	...	VAL1	Value data 1 (optional)
...

Items inside a rung have a natural order: monitor value for these items follow the same order, split between boolean values (represented with consecutive bit in BITSx fields) and numeric values (represented with consecutive VALx fields).

Bit no. 0 of BITS0 is used to represent the general boolean state of the rung itself; starting from bit no. 1 there are the user boolean values.

Each VALx field is structured as:

Offset	Type	Label	Description
+0	U8	FLAGS	bit0-bit3 = Data type (TYPE) bit4-bit7 = Reserved
+1	...	DATA	Raw data

while the DATA field is structured as (depending on data type value):

Type	Offset	Type	Description
0x0		-	(reserved)
0x1	+1	I8	Signed integer 8bit
0x2	+1	U8	Unsigned integer 8bit
0x3	+1	I16	Signed integer 16bit
0x4	+1	U16	Unsigned integer 16bit

Type	Offset	Type	Description
0x5	+1	I32	Signed integer 32bit
0x6	+1	U32	Unsigned integer 32bit
0x7	+1	I64	Signed integer 64bit
0x8	+1	U64	Unsigned integer 64bit
0x9	+1	DBL	Floating point 64bit
0xA	+1	FLT	Floating point 32bit
0xB	+1	-	Boolean (TRUE): has no raw data.
0xC	+1	-	Boolean (FALSE): has no raw data.
0xD		-	(reserved)
0xE		-	(reserved)
0xF		-	(reserved)

NOTE: to have more informations about the composition (and the order) of data sets, see [Ladder Monitor Data Types](#) in the ladder diagram documentation.

Start live changes testing

Code:	AS + 803
Symbolic:	bccLadLiveTest

This command will start testing for live changes of a specific session. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Live changes session ID
4	U32	TIMEWD	Initial watch dog time [ms]

On success, a [bccAck](#)_[164] is received with no data.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=SESSID 2=TIMEWD

NACK code	Description	Extra data
nackNotFound	Live changes session not found	

Stop ladder monitor

Code:	AS + 812
Symbolic:	bccLadMonStop

This command will stop the specified ladder monitor (or all monitor for same owner). Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	MONID	Monitor ID (or 0xFFFFFFFF for all monitor for same owner)

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 0=Owner ID 1=Monitor ID
nackNotFound	Monitor not found	

Watchdog for ladder monitor

Code:	813
Symbolic:	bccLadMonWd

This command will refresh the grant time (watchdog) for specified ladder monitor. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	MONID	Monitor ID
8	U32	TIMEWD	Watchdog time [ms]

The message has no reply.

Watchdog for live changes

Code:	806
Symbolic:	bccLadLiveWd

This command will refresh the grant time (watchdog) for specified live changes session. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Live changes session ID
4	U32	TIMEWD	Watchdog time [ms]

The message has no reply.

Monitor handling

These messages are used for handling real-time variable monitor for a connected device.

Monitor handling:

- [bccMonCreate](#)^[189], create a monitor
- [bccMonDestroy](#)^[191], destroy a monitor
- [bccMonStart](#)^[194], start a monitor
- [bccMonStop](#)^[195], stop a monitor
- [bccMonStatus](#)^[191], query a monitor status
- [bccMonList](#)^[192], query list of monitors
- [bccMonQuick](#)^[193], quick monitor
- [bccMonWd](#)^[196], watchdog for a monitor
- [bccMonWrite](#)^[196], write a monitor
- [bccMonStatInfo](#)^[193], query monitors statistics

Related arguments:

- [Monitor specification](#)^[11]
- [Using monitor](#)^[11]
- [Using multiple monitor](#)^[12]

Create a monitor

Code:	AS + 400
Symbolic:	bccMonCreate

This command will try to create a new variable monitor on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U8	N	Number of variables (1-25)
5	VAR ^[9]	VAR1	Data for variable 1

Offset	Type	Label	Description
...	VAR ₉	VAR2	Data for variable 2
...
...	VAR ₉	VAR{N}	Data for variable {N}

Notes:

- maximum value for N depend on maximum data size (no more than 251 bytes) with a limit of 25 variables.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U32	ID	Monitor ID
4	U8	SIZE	Total data size, calculated by device (must match your request resulting data size).

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackOutOfResource	No more resource to create a new monitor,	
nackDataOverflow	Data for monitor is exceeding the maximum limit (for BCC3 is 255 bytes)	
nackIllegalArgs	Illegal arguments	U16 What illegal 1=(unused) 2=Number of variable 10+x=Bad variable {x} (x is 1-based) U16 Variable error code 1=Unkown type 2=Bad index 3=Bad address 4=Bad repeat counter 5=Variable data too long

Notes:

- Don't make any assumption about monitor ID assignment: they are strictly depending to a specific implementation of the BCC3 protocol and can be both in sequential or random order.
- Don't make any assumption of monitor definition persistence: if monitor structures - on a device - run out of space, the specific implementation of the BCC3 protocol can act in different way:

1. Reply with a `nackOutOfResource` error.
 2. Reuse monitor entries not yet started (older or in an arbitrary order).
- Monitor can use not initialized variables in reading, but they always have conventional 0 value.

Destroy a monitor

Code:	AS + 401
Symbolic:	bccMonDestroy

This command will try to destroy an existing variable monitor (or all monitor of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	ID	Monitor ID, or 0xFFFFFFFF for all owner's monitor.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Monitor not found	

NOTE: if monitor is running (data stream active), it will be automatically [stopped](#)^[195].

Query a monitor status

Code:	AS + 404
Symbolic:	bccMonStatus

This command will query current status of a variable monitor on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	ID	Monitor ID

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U8	STATUS	Current monitor status: 0x00 = Monitor not defined

Offset	Type	Label	Description
			0x01 = Monitor inactive (but defined, waiting for start) 0x02 = Monitor active (data stream active) 0x03 = Monitor expired (timer WD denied)
1	U16	FREQ	Applied (real) data stream frequency [hz]

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Monitor not found on device	

Query list of monitors

Code:	AS + 405
Symbolic:	bccMonList

This command will query list of defined/active monitor of the owner on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID

On success, a [bccAck](#)_[164] is received with the following data:

Offset	Type	Label	Description
0	U8	N	Number of monitor ID in list
1	U32	ID0	Monitor ID 0
...
...	U32	ID{N-1}	Monitor ID {N-1}

Notes:

- on protocol BCC3 the maximum number N is 63.

On failure, a [bccNack](#)_[164] is received.

Query monitor statistics

Code:	AS + 409
Symbolic:	bccMonStatInfo

This command will be used to query general statistical information about all device monitors. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Statistical flags: (none)
4	U32	NMAX	No. of maximum monitors (capacity)
8	U32	NDEF	No. of defined monitors
12	U32	NNACT	No. of inactive monitors
16	U32	NACT	No. of active monitors
20	U32	NEXP	No. of expired monitors
24	U16	SFREQ	Suggested data stream frequency [hz]
26	U32	SWDT	Suggested watchdog time [ms]

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags

Quick monitor

Code:	AS + 406
Symbolic:	bccMonQuick

This command will query monitor data quickly, only one-shoot request. Request parameters are the following:

Offset	Type	Label	Description
0	U8	N	Number of variables (1-25)
1	VAR ₉	VAR0	Variable def 0
...	VAR ₉	VAR1	Variable def 1
...
...	VAR ₉	VAR{N-1}	Variable def N-1

Notes:

- maximum value for N have to be calculate dynamically (maximum 25 var BCC3) . For BCC3 you should calculate that sum of resulting variables sizes is less or equal to 255 bytes.

On success, a [bccAck](#)₁₆₄ is received containing request data, according monitor [variable](#)₂₈₉ definition.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackDataOverflow	Data for monitor is exceeding the maximum limit (for BCC3 is 255 bytes)	
nackIllegalArgs	Illegal arguments	U16 What illegal 1=(unused) 2=Number of variable 10+N=Bad variable {N} U16 Variable error code 1=Unkown type 2=Bad index 3=Bad address 4=Bad repeat counter 5=Variable data too long

Notes:

- quick monitor can use not initialized variables in reading, but they always have conventional 0 value.

Start a monitor

Code:	AS + 402
Symbolic:	bccMonStart

This command will start the data stream for an existing variable monitor (or all monitor of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	ID	Monitor ID, or 0xFFFFFFFF for all owner's monitor.
8	U16	FREQ	Data stream frequency [hz]
10	U32	WDT	Watchdog time [ms] (initial)

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U16	FREQ	Applied (real) data stream frequency [hz]

Notes:

- the device can reply with a different frequency than required. In this case you can accept the new frequency or stop the monitor or try again by changing something in the monitor definition.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Monitor not found on device	
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Frequency 2=WD time

After success, you will receive [bccData](#)^[165] message at applied frequency (or less if device overloaded), with the following data:

Type	Label	Description
U32	ID	Monitor ID
B[]	DATA	Data according monitor variable ^[289] definition

Notes:

- the PID field for data message is set to 0 when monitor is created and it is incremented by one after each data transmission (progressive).

Stop a monitor

Code:	AS + 403
Symbolic:	bccMonStop

This command will stop the data stream an existing and running variable monitor (or all monitor of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	ID	MonitorID, or 0xFFFFFFFF for all owner's monitor.

On success, a [bccAck](#)_[164] is received with no data: the protocol will respond ack even in the monitor is not started.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Monitor not found on device	

Watchdog for a monitor

Code:	407
Symbolic:	bccMonWd

This message will refresh transmission grant time (monitor watchdog) for specified variable monitor (or all monitor of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID
4	U32	ID	Monitor ID, or 0xFFFFFFFF for all owner's monitor.
8	U32	WDT	Watchdog time [ms]

The message has no reply.

Notes:

- if a monitor watchdog is expired and it's definition is yet valid, a [bccMonWd](#) will automatically restart the monitor (like [bccMonStart](#)_[194]).

Write a monitor

Code:	AS + 408
Symbolic:	bccMonWrite

This command will be used to write data to an existing monitor. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner ID

Offset	Type	Label	Description
4	U32	ID	Monitor ID
8	...		Monitor data (#1) according monitor variable ^[289] definition

(#1) Due to initial parameter, the maximum data length is reduced to 255-8.

On success, a [bccAck](#)^[164] is received .

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Monitor not found on device	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Data size (mismatch)
nackReadOnly	Cant write data	I32 Item index
nackNotInitialized	Variable is not initialized	I32 Item index

Network handling

These message are used for handling general or specific network interface.

Security handling:

- [bccNetLogin](#)^[200], network login
- [bccNetLogout](#)^[200], network logout

Miscellaneous handling

- [bccNetInfo](#)^[203], query network information
- [bccNetStats](#)^[204], query network statistics

Network users handling:

- [bccNetUserList](#)^[202], query list of network users
- [bccNetUserCreate](#)^[199], create a new network user
- [bccNetUserDelete](#)^[199], delete a network user
- [bccNetUserChange](#)^[198], change a network user

General arguments:

- [Network interfaces](#)^[272]

Notes:

- Commands for network interface handling always have destination value 3 (field [DST](#)^[9] of bcc message), otherwise the command will be routed to the connected device.

Network client handling:

- [bccNetClientList](#)^[202], query list of network clients
- [bccNetClientKill](#)^[200], kill a network client

Keep alive session handling:

- [bccNetClientKasSessionBegin](#)^[205], begin a keep-alive session
- [bccNetClientKasSessionEnd](#)^[206], end a keep-alive session
- [bccNetClientKasSessionInfo](#)^[207], query info for a keep-alive-session

Change a network user

Code:	AS + 605
Symbolic:	bccNetUserChange

NOTE: this command is **preliminary**.

This command will change an existing network user settings on the connected network device: active user must have appropriate permissions in order to perform the request. Request parameter are the following:

Offset	Type	Label	Description
0	STRZ(16)	NAME	User name
16	STRZ(16)	PASS	New user password
32	U8	FLAGS	Settings: 0x01 Change password 0x02 Change MSGPERMS 0x04 Change NETPERMS
33	U32	MSGPERMS	New message permission flags
37	U32	MSGMASK	Message permission mask
41	U32	NETPERMS	New network permission flags
45	U32	NETMASK	Network permission mask

Notes:

- Effective changed message and network permission are respectively (MSGPERMS & MSGMASK) and (NETPERMS & NETMASK), if setting enable them.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	User does not exist in configuration	

Create a new network user

Code:	AS + 603
Symbolic:	bccNetUserCreate

NOTE: this command is **preliminary**.

This command will add a new network user on the connected network device: active user must have appropriate permissions in order to perform the request. Request parameter are the following:

Offset	Type	Label	Description
0	STRZ(16)	NAME	User name
16	STRZ(16)	PASS	User password
32	U32	MSGPERMS	Message permissions
36	U32	NETPERMS	Network permissions

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackExists	User already exist in configuration	

Delete a network user

Code:	AS + 604
Symbolic:	bccNetUserDelete

NOTE: this command is **preliminary**.

This command will delete an existing network user on the connected network device: active user must have appropriate permissions in order to perform the request. Request parameter are the following:

Offset	Type	Label	Description
0	STRZ(16)	NAME	User name

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	User does not exist in configuration	

Kill a network client

Code:	AS + 610
Symbolic:	bccNetClientKill

This command will get try to kill a specific client for the connected network device: active user must have appropriate permission(s) in order to perform the request.

Request parameter are following:

Offset	Type	Label	Description
0	U8	ID	Client ID

On success, a [bccAck](#)_[164] is received with no data.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Client not found or not active	

Network login

Code:	AS + 600
Symbolic:	bccNetLogin

This command will request to authenticate to the current connection (login): according to network interface settings, if authenticated successfully, you will gain some permission to communicate (for more information see [Network interfaces](#)_[272]). Request parameters

Offset	Type	Label	Description
0	STRZ(16)	USER	User name
16	STRZ(16)	PASS	Password

On success, a [bccAck](#)_[164] is received with no data.

On failure, a [bccNack](#)_[164] is received.

Network logout

Code:	AS + 601
Symbolic:	bccNetLogout

This command will release an authentication previously gained ([bccNetLogin](#)^[200]), so the connection return to its original state. This command is provided for commodity only and normally is not used, since when a connection terminate it's authentication is also deleted.

Request has no parameters.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received.

Query a keep alive session information of a network client

Code:	AS + 614
Symbolic:	bccNetClientKasSessionInfo

This command will query information about the keep alive session for current network client. Request has no parameters.

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	STOUT	Keep alive session effective timeout [ms]
4	U32	STLEFT	Session left time before timeout [ms]
8	U32	STATE	State of the session: 0x1 TCP client connected 0x2 Communication active 0x4 Lost activity latched
12	STRZ(16)	SIP	Source IP (client)
28	U16	SPORT	Source port (client)
30	U16	TPORT	Target port (device)
32	U8	TEXTSIZE	Field TEXT size
33	STRZ	TEXT	Session description

Notes:

- The bccNetClientKasSessionInfo command is a session exception and will not renew the keep-alive timeout.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	No keep alive session found	

NACK code	Description	Extra data
nackIllegalContext	Illegal context for a keep-alive-session (e.g. not a network connection)	

Query list of network clients

Code:	AS + 609
Symbolic:	bccNetClientList

This command will get list of currently client connections for the connected network device: active user must have appropriate permission(s) in order to perform the request. This is a standard [download transfer sequence](#)^[14].

REQDATA structure has no data.

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U8	ID	Client ID
1	STRZ(16)	IP	Network address (IP, string)
17	STRZ(16)	NAME	User name
33	U16	PORT	Network port
35	FLT	TL	Time live [s]
39	U32	TXC	Message TX total count
43	U32	RXC	Message RX total count
47	FLT	TXF	Message TX media frequency [hz]
51	FLT	RXF	Message RX media frequency [hz]

Query list of network users

Code:	AS + 602
Symbolic:	bccNetUserList

NOTE: this command is **preliminary**.

This command will get a list of currently configured network users on the connected network device: active user must have appropriate permissions in order to perform the request. This is a standard [download transfer sequence](#)^[14].

REQDATA structure has no data.

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	STRZ(16)	NAME	User name
16	U32	MSGPERMS	Message permissions
20	U32	NETPERMS	Network permissions

Query network information

-

Code:	AS + 608
Symbolic:	bccNetInfo

This command will try to query connected network device information. Request has no parameters.

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	TYPE	Network device type: 0x00000000 Generic network device 0x00000001 NET.INT. expansion board 0x00000002 TCP server/BCC library
4	U32	VERS	Network device version (nvMake format)
...	More optional data according TYPE value.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	User does not exist in configuration	

For type 0x000000001 (NET.INT. expansion board) following extra data area provided:

Offset	Type	Label	Description
8	U32	STSF	Status flags: 0x00000001 running verbose mode 0x00000002 running debug mode

Offset	Type	Label	Description
			0x00000004 running no authorization mode 0x00000008 running RTAI mode 0x00000010 computing statistics 0x00000020 DP interface active 0x00000040 TCP interface active
12	U32	MAXC	Maximum number of TCP clients
16	U32	DPV	DP driver version (nvMake format)

For type 0x00000002 (TCP server/BCC library) following extra data are provided:

Offset	Type	Label	Description
8	U32	STSF	Status flags: 0x00000001 running no authorization mode
12	U32	MAXC	Maximum number of TCP clients
16	U32	HSWV	Host software versione
20	U8[32]	HSWN	Host software name

Query network statistics

Code:	AS + 611
Symbolic:	bccNetStats

This command will try to query connected network device informations. Request has no parameters.

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	FLT	SUP	Statistics update period [s]
4	FLT	MFREQ	Main frequency [hz]

Offset	Type	Label	Description
8	U8	NINTF	No. of following interface data
9	INTF	INT0	Interface 0
39	INTF	INT1	Interface 1
...

Where the INTF (30 bytes) is defined as:

Offset	Type	Label	Description
+0	U32	TXC	Total message transmitted to interface
+4	U32	RXC	Total message received from interface
+8	U32	ERR	Total count of interface errors
+12	FLT	TXF	Interface outgoing media frequency [hz]
+16	FLT	RXF	Interface ingoing media frequency [hz]
+20	STRZ(10)	NAME	Interface name

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	User does not exist in configuration	

Start a keep alive session for a network client

Code:	AS + 612
Symbolic:	bccNetClientKasSessionBegin

This command will start a keep alive session for current network client. Request parameter are the following:

Offset	Type	Label	Description
0	U32	FLAG	Request flags
4	U32	STOUT	Keep alive session timeout [ms] (0=unlimited; #1)

Offset	Type	Label	Description
8	U8	TEXTSIZE	Field TEXT size
9	STRZ	TEXT	Session description

(#1) Unlimited or the maximum value is decided by the software of the connected device. The effective value can be obtained with the [bccNetClientKasSessionInfo](#)^[201] command, in field STOUT.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalContext	Illegal context for a keep-alive-session (e.g. not a network connection)	
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Flags 2=Keep alive timeout 3=Description
nackExists	A keep-alive-session is already active for the client	

Stop a keep alive session for a network client

Code:	AS + 613
Symbolic:	bccNetClientKasSessionEnd

This command will stop a keep alive session for current network client. Request has no parameters.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	No keep alive session found	
nackIllegalContext	Illegal context for a keep-alive-session (e.g. not a network connection)	

Oscilloscope handling

These messages are used for handling real-time variable oscilloscope for a connected device.

Oscilloscope handling:

Related arguments:

- [bccOscCreate](#)^[207], create an oscilloscope
- [Oscilloscope specifications](#)^[13]

- [bccOscDestroy](#)^[208], destroy an oscilloscope
- [bccOscStart](#)^[209], start an oscilloscope
- [bccOscStop](#)^[210], stop an oscilloscope
- [bccOscStatus](#)^[211], query an oscilloscope status
- [bccOscList](#)^[212], query list of oscilloscopes
- [bccOscStatInfo](#)^[213], query oscilloscopes statistics
- [bccOscWd](#)^[213], watchdog for an oscilloscope
- [Using oscilloscope](#)^[13]
- [Using multiple oscilloscope](#)^[14]

Create an oscilloscope

Code:	AS + 420
Symbolic:	bccOscCreate

This command will try to create a new variable oscilloscope on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U8	N	Number of tracks (1-24)
5	VAR ^[9]	VAR1	Data for track 1 variable
15	VAR ^[9]	VAR2	Data for track 2 variable
...
...	VAR ^[9]	VAR{N}	Data for track {N} variable

Notes:

- The maximum value for N is 24: the size is calculated as $5 + N * 10$.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	ID	Oscilloscope identification ID
4	U8	SIZE	Total data size, calculated by device (must match your request resulting data size).

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackOutOfResource	No more resource to create a new oscilloscope,	
nackDataOverflow	Data for monitor is exceeding the maximum limit (for BCC3 is 255 bytes)	
nackIllegalArgs	Illegal parameter	U16 What illegal 1=(unused) 2=Field count 10+I=Bad variable {I} (I is 1-based) U16 Variable error code 1=Unknown type 2=Bad index 3=Bad address 4=Bad repeat counter 5=Variable data too long

Notes:

- Don't make any assumption about oscilloscope ID assignment: they are strictly depending to a specific implementation of the BCC3 protocol and can be both in sequential or random order.
- Don't make any assumption of oscilloscope definition persistence: if oscilloscope structures - on a device - run out of space, the specific implementation of the BCC3 protocol can act in different way:
 1. Reply with a nackOutOfResource error.
 2. Reuse oscilloscope entries not yet started (older or in an arbitrary order).
- Oscilloscope does not accept multiple value variables, but only those one with single value: if you need multiple value, consider using a [monitor](#)^[11] instead.
- All variable value are transmitted as float value so if your query double variables you can lost precision.
- Oscilloscopes can use not initialized variables in reading, but they always have conventional 0 value.

Destroy a oscilloscope

Code:	AS + 421
Symbolic:	bccOscDestroy

This command will try to destroy an existing variable oscilloscope (or all oscilloscope of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID, or 0xFFFFFFFF for all owner's oscilloscope.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Oscilloscope not found	

Notes:

- If the oscilloscope is running (data stream active), it will be automatically stopped.

Start an oscilloscope

Code:	AS + 422
Symbolic:	bccOscStart

This command will try to start an existing variable oscilloscope (or all oscilloscope of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID, or 0xFFFFFFFF for all owner's oscilloscope.
8	U16	FREQ	Data stream frequency [hz]
10	U32	WDT	Watchdog time [ms] (initial)

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U16	FREQ	Applied (real) data stream frequency [hz]

Notes:

- The device can reply with a different frequency than required. In this case you can accept the new frequency or stop the oscilloscope or try again by changing something in the oscilloscope definition.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Oscilloscope not found	
nackIllegalArgs	Illegal arguments	U16 Error field 1=Frequency 2=WD time

After success, you will receive [bccData](#)^[165] message at applied frequency (or less if device overloaded), with the following data:

Offset	Type	Label	Description
0	U32	ID	Oscilloscope ID
4	U8	NDS	N. of DS structure
5	DS	DS0	Data of sample 0
...	... ^[9]

Notes:

- The number of samples (NDS) is decided by the connected device, according with applied frequency. For example, if the frequency is 200HZ the device can decide to send each sample with a message at 200HZ or to send two-samples with a message at 100HZ.

This will allow to have high frequency oscilloscope (with few tracks) even over a standard and limited RS232 device.

Each structure DS (data sample) is defines as follow :

Offset	Type	Label	Description
+0	DBL	TIME	Absolute acquisition time [us]
+8	FLT	VAL0	Track 0 value
+12	FLT	VAL1	Track 1 value (opz)
...
...	FLT ^[9]	VAL{NT-1}	Track NT value (opz)

Notes:

- The number of effective track values is the same of number of tracks (NT) in oscilloscope creation with [bccOscCreate](#)^[207].
- The PID field for data message is set to 0 when oscilloscope is created and it is incremented by one after each data transmission (progressive).

Stop an oscilloscope

Code:	AS + 423
Symbolic:	bccOscStop

This command will try to stop an existing and running variable oscilloscope (or all oscilloscope of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID, or 0xFFFFFFFF for all owner's oscilloscope.

On success, a [bccAck](#)₁₆₄ is received with no data: the protocol will respond ack even in the oscilloscope is not started.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Oscilloscope not found	

Query an oscilloscope status

Code:	AS + 424
Symbolic:	bccOscStatus

This command will query current status of a variable oscilloscope on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	U8	STATUS	Current status: 0x00 = Oscilloscope not defined 0x01 = Oscilloscope inactive (but defined, waiting for start) 0x02 = Oscilloscope active (data stream active) 0x03 = Oscilloscope expired (timer WD denied)

Offset	Type	Label	Description
1	U16	FREQ	Applied (real) data stream frequency [hz]

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Oscilloscope not found	

Query list of oscilloscopes

Code:	AS + 424
Symbolic:	bccOscStatus

This command will query current status of a variable oscilloscope on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U8	STATUS	Current status: 0x00 = Oscilloscope not defined 0x01 = Oscilloscope inactive (but defined, waiting for start) 0x02 = Oscilloscope active (data stream active) 0x03 = Oscilloscope expired (timer WD denied)
1	U16	FREQ	Applied (real) data stream frequency [hz]

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackNotFound	Oscilloscope not found	

Query oscilloscopes statistics

Code:	AS + 426
Symbolic:	bccOscStatInfo

This command will be used to query statistical information about all device oscilloscopes. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)

On success, a [bccAck](#)¹⁶⁴ is received with the following data:

Offset	Type	Label	Description
0	U32	FLAGS	Statistical flags: (none)
4	U32	NMAX	No. of maximum oscilloscopes (capacity)
8	U32	NDEF	No. of defined oscilloscopes
12	U32	NNACT	No. of inactive oscilloscopes
16	U32	NACT	No. of active oscilloscopes
20	U32	NEXP	No. of expired oscilloscopes
24	U16	SFREQ	Suggested data stream frequency [hz]
26	U32	SWDT	Suggested watchdog time [ms]

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal param	U16 What illegal: 1=Flags

Watchdog for an oscilloscope

Code:	427
Symbolic:	bccOscWd

This message will refresh transmission grant time (oscilloscope watchdog) for specified variable oscilloscope (or all oscilloscope of the owner) on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNER	Owner identification ID
4	U32	ID	Oscilloscope identification ID, or 0xFFFFFFFF for all owner's oscilloscope.
8	U32	WDT	Watchdog time [ms]

The message has no reply.

Notes:

- If an oscilloscope watchdog is expired and it's definition is yet valid, a `bccOscWd` will automatically restart the oscilloscope (like [bccOscStart](#)^[209]).

Protocol handling

These messages are used for general protocol handling and protocol diagnostics.

Ping/pong testing:

- [bccPing](#)^[215], Ping command
- [bccPong](#)^[215], Ping answer

Related arguments:

[Routing counters](#)^[215]
[Communication timings](#)^[216].

Protocol debug:

- [bccDebugCmd](#)^[214], Debug command

Debug command

Code:	AS + 35
Symbolic:	bccDebugCmd

WARNING: this command is intended only for diagnostic use by Robox SpA.

Send a debug command to the connected device. Request data area is the following:

Type	Label	Description
U32	CMD	Debug command
B[]	DATA	Extra data (optional) can be appended, according CMD value and its implementation.

On success, a [bccAck](#)^[164] is received with following (opt) data:

Type	Label	Description
B[]	DATA	Data (optional) return as command result

On failure, a [bccNack](#)^[164] is received.

Ping answer

Code:	39
Symbolic:	bccPong

This message is used as answer to request message [bccPing](#)^[215] and contain reply data as follow:

Type	Label	Description
U32	DATA	Original ping data
U32	TXHOPS	Number of transmission hops
U32	RXHOPS	Number of receiving hops

Notes:

- It is best practice to always initialize RXHOPS with value of 1.

Ping command

Code:	AS + 38
Symbolic:	bccPing

Send a ping request to discover communication timings. Request data area is the following:

Type	Label	Description
U32	DATA	Ping user data
U32	TXHOPS	Number of transmission hops

Notes:

- It is best practice to always initialize TXHOPS with value of 1.

On success, a [bccPong](#)^[215] is received (see below for details).

On failure, a [bccNack](#)^[164] is received.

Routing counters

The hop counter is used to count how many time a message have to be routed before reaching is destination.

By initializing it to value 1, each type that the message is routed, the hop counter is incremented: when message [bccPong](#)^[215] return, the hops counters give you an exact count of transmission cost.

Communication timings

By using appropriate information in DATA field of [bccPing](#)^[215] message, you can evaluate communication timing.

For example if you place current time [ms] in DATA of [bccPing](#)^[215], where you will receive the [bccPong](#)^[215] by the difference of current time [ms] and the value in returned DATA, you will obtain the elapsed time between transmission of [bccPing](#)^[215] and receiving of [bccPong](#)^[215] (this is possible because the DATA field of [bccPing](#)^[215] message is replicated into DATA field of [bccPong](#)^[215]).

Register handling

NOTE: although defined and used on some hardware Robox device, these command are declared deprecated and are not supported by RTE: we suggest to use [variables](#)^[249] instead.

These messages are used getting and setting register value for a connected device.

Get commands:

- [bccGetR16](#)^[216], get 16bit integer register
- [bccGetR32](#)^[217], get 32bit integer register
- [bccGetRR](#)^[219], get real register (double)
- [bccGetRRF](#)^[218], get real register (float)
- [bccGetSR](#)^[220], get string register

Set commands:

- [bccSetR16](#)^[221], set 16bit integer register
- [bccSetR32](#)^[222], set 32bit integer register
- [bccSetRR](#)^[223], set real register (double)
- [bccSetRRF](#)^[222], set real register (float)
- [bccSetSR](#)^[224], set string register

Get 16bit integer register

Code:	AS + 300
Symbolic:	bccGetR16

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will get value for multiple 16bit integer register. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)

Notes:

- The maximum value for REP in BCC3 is 127.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	I16	VALUE0	Value of register (INDEX+0)
2	I16	VALUE1	Value of register (INDEX+1)
4

Notes:

- The data size is calculated as 2 bytes * REP.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Get 32bit integer register

Code:	AS + 302
Symbolic:	bccGetR32

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will get value for multiple 32bit integer register. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)

Notes:

- The maximum value for REP in BCC3 is 63.

On success, a [bccAck](#)₁₆₄ is received with the following data:

Offset	Type	Label	Description
0	I32	VALUE0	Value of register (INDEX+0)
4	I32	VALUE1	Value of register (INDEX+1)
8

Notes:

- The data size is calculated as 4 bytes * REP.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Get float register

Code:	AS + 306
Symbolic:	bccGetRRF

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will get value for multiple real register (in float precision, 32bit). Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)

Notes:

- The maximum value for REP in BCC3 is 63.

On success, a [bccAck](#)_[164] is received with following data:

Offset	Type	Label	Description
0	FLT	VALUE0	Value of register (INDEX+0)

Offset	Type	Label	Description
4	FLT	VALUE1	Value of register (INDEX+1)
8

Notes:

- The data size is calculated as 4 bytes * REP.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Get real register

- (double) (deprecated)

Code:	AS + 304
Symbolic:	bccGetRR

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will get value for multiple real register (in double precision, 64bit). Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/osf documentation for valid register index values)

Notes:

- The maximum value for REP in BCC3 is 31.

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	DBL	VALUE0	Value of register (INDEX+0)

Offset	Type	Label	Description
8	DBL	VALUE1	Value of register (INDEX+1)
16

Notes:

- The data size is calculated as 8 bytes * REP.

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Get string register

Code:	AS + 308
Symbolic:	bccGetSR

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will get value for a single string register. Request parameters are the following:

Offset	Type	Label	Description
0	U16	INDEX	Index of string register (see your hardware/osf documentation for valid register index values)

On success, a [bccAck](#)_[164] is received with following data:

Offset	Type	Label	Description
0	STRZ	VALUE	String value (0 terminated)

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Index

Set 16bit integer register

Code:	AS + 301
Symbolic:	bccSetR16

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will set value for multiple 16bit integer register. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)
3	I16	VALUE0	Value for register (INDEX+0)
5	I16	VALUE1	Value for register (INDEX+1)
7

Notes:

- The maximum value for REP in BCC3 is 126.
- The data size is calculated as 3 bytes + (2 bytes * REP)

On success, a [bccAck](#)₁₆₄ is received without data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Register read-only or predefined	U16 Error value index (from 0 to REP-1)
nackMissingArgs	One or more parameter missing	
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Set 32bit integer register

Code:	AS + 303
Symbolic:	bccSetR32

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will set value for multiple 32bit integer register. Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/osf documentation for valid register index values)
3	I32	VALUE0	Value for register (INDEX+0)
7	I32	VALUE1	Value for register (INDEX+1)
11

Notes:

- The maximum value for REP in BCC3 is 63.
- The data size is calculated as 3 bytes + (4 bytes * REP)

On success, a [bccAck](#)₁₆₄₁ is received without data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Register read-only or predefined	U16 Error value index (from 0 to REP-1)
nackIllegalArgs	Illegal paramter	U16 What illegal 1=Repeat 2=Index

Set float register

Code:	AS + 307
Symbolic:	bccSetRRF

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will set value for multiple real register (in float precision, 32bit). Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)
3	FLT	VALUE0	Value for register (INDEX+0)
7	FLT	VALUE1	Value for register (INDEX+1)
11

Notes:

- The maximum value for REP in BCC3 is 63.
- The data size is calculated as 3 bytes + (4 bytes * REP)

On success, a [bccAck](#)₁₆₄ is received without data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Register read-only or predefined	U16 Error value index (from 0 to REP-1)
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Set real register

Code:	AS + 305
Symbolic:	bccSetRR

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will set value for multiple real register (in double precision, 64bit). Request parameters are the following:

Offset	Type	Label	Description
0	U8	REP	Number of required registers
1	U16	INDEX	Index of first register (see your hardware/OSF documentation for valid register index values)
3	DBL	VALUE0	Value for register (INDEX+0)
11	DBL	VALUE1	Value for register (INDEX+1)
19

Notes:

- The maximum value for REP in BCC3 is 31.
- The data size is calculated as 3 bytes + (8 bytes * REP)

On success, a [bccAck](#)₁₆₄ is received without data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Register read-only or predefined	U16 Error value index (from 0 to REP-1)
nackIllegalArgs	Illegal parameter	U16 What illegal 1=Repeat 2=Index

Set string register

Code:	AS + 309
Symbolic:	bccSetSR

NOTE: this command is **not supported** in RTE firmware.

NOTE: This register is referred to the volatile register set.

This command will set value for a single string register. Request parameters are the following:

Offset	Type	Label	Description
0	U16	INDEX	Index of string register

Offset	Type	Label	Description
2	STRZ	VALUE	String value (0 terminated)

Notes:

- The data size is calculated as 2 bytes + len(VALUE) + 1

On success, a [bccAck](#)^[164] is received without data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Register read-only or predefined	
nackIllegalArgs	Illegal arguments	U16 What illegal 1=Repeat 2=Index

Report handling

These messages are used handling report (standard and system specific) for the connected device.

Standard report handling:

- [bccReportInfo](#)^[230], query report information
- [bccReportList](#)^[227], get report contents
- [bccReportCmd](#)^[225], command for report

System report handling:

- [bccSysReportInfo](#)^[231], query system report information
- [bccSysReportList](#)^[228], get system report contents
- [bccSysReportCmd](#)^[226], command for system report

Command for report

Code:	AS + 732
Symbolic:	bccReportCmd

This command will send a command to report on the remote device. Request has following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Command flags: 0x00000001 Clear report 0x00000002 Set report size 0x00000004 Set 'up to fill' mode

Offset	Type	Label	Description
			0x00000008 Set 'round' mode 0x00000010 Set acquisition mask
4	U32	SIZE	New report size (no. of item), if bit enabled
8	U16	AMSK	New report acquisition mask.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Flags 2=Size 3=Mask
nackReadOnly	Report is readonly and is not modifiable.	

Command for system report

Code:	AS + 735
Symbolic:	bccSysReportCmd

This command will send a command to system report on the remote device. Request has following parameters:

Offset	Type	Label	Description
0	U32	FLAGS	Command flags: 0x00000001 Clear report 0x00000002 Set system report size 0x00000004 Set 'up to fill' mode 0x00000008 Set 'round' mode 0x00000010 Set acquisition mask
4	U32	SIZE	New system report size (no. of item), if bit

Offset	Type	Label	Description
			enabled
8	U16	AMSK	New system report acquisition mask.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Flags 2=Size 3=Mask
nackReadOnly	System report is readonly and is not modifiable.	

Get report contents

Code:	AS + 731
Symbolic:	bccReportList

This command will get complete (or partial) contents of the report from the remote device.: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FROMID	First report item ID required
4	U32	TOID	Last report item ID required
8	U32	NITEM	No. of report items (0=all available)

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U32	ID	Report item ID
4	DBL	TIME	Item generation time [us]
12	U16	SMSK	Source mask: 0x0000 Generic

Offset	Type	Label	Description
			0x0001 RTE/RRT 0x0002 OS 0x0004 Fieldbus 0x0008 (reserved) 0x0010 (reserved) 0x00E0 Item category: 0x00 generic 0x20 information 0x40 warning 0x80 fault/emergency 0x0100 (user) 0x0200 (user) 0x0400 (user) 0x0800 (user) 0x1000 (user) 0x2000 (user) 0x4000 (user) 0x8000 (user)
14	STRZ	TEXT	Item text

As general, you can get items with the following order:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

When using RTE version v33.16.x (or lower) you should consider two different cases:

1. NEWID is greater/equal than OLDID.
2. NEWID is less than OLDID.

In case 1, you can get items with a single request, as:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

In case 2, you must get items with two different request, as:

- from OLDID to 0xFFFFFFFF and from 0 to NEWID for native order.
- from NEWID to 0 and from 0xFFFFFFFF to OLDID for reversed order.

Get system report contents

Code:	AS + 734
-------	-----------------

Symbolic:

bccSysReportList

This command will get complete (or partial) system report history from the remote device.: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FROMID	First system report item ID required
4	U32	TOID	Last system report item ID required
8	U32	NITEM	No. of system report items (0=all available)

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U32	ID	System report item ID
4	DBL	TIME	Item generation time [us]
12	U16	SMSK	Source mask: 0x0000 Generic 0x0001 RTE/RRT 0x0002 OS 0x0004 Fieldbus 0x0008 (reserved) 0x0010 (reserved) 0x00E0 Item category: 0x00 generic 0x20 information 0x40 warning 0x80 fault/emergency 0x0100 (user) 0x0200 (user) 0x0400 (user) 0x0800 (user) 0x1000 (user) 0x2000 (user) 0x4000 (user) 0x8000 (user)

Offset	Type	Label	Description
14	STRZ	TEXT	Item text

As general, you can get items with the following order:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

When using RTE version v33.16.x (or lower), you should consider two different cases:

1. NEWID is greater/equal than OLDID.
2. NEWID is less than OLDID.

In case 1, you can get items with a single request, as:

- from OLDID to NEWID for native order.
- from NEWID to OLDID for reversed order.

In case 2, you must get items with two different request, as:

- from OLDID to 0xFFFFFFFF and from 0 to NEWID for native order.
- from NEWID to 0 and from 0xFFFFFFFF to OLDID for reversed order.

Query report information

Code:	AS + 730
Symbolic:	bccReportInfo

This command will query information about report on the remote device. Request has no parameters.

On success, a [bccAck](#)_[164] is received with following data:

Offset	Type	Label	Description
0	U32	OLDID	Oldest report item ID
4	U32	NEWID	Newest report item ID
8	U32	NITEM	No. of report items (in range from OLDID to NEWID, all included)
12	U32	SIZE	Report size (no. of storable items)
16	U32	REPID	Report content ID

On failure, a [bccNack](#)_[164] is received.

Report handling is based on the following statements:

- Item ID are allocated progressively, with an exception: at value 0xFFFFFFFF, next valid ID will be 0.
- The no. of items from OLDID and NEWID will be calculated as: if NEWID great or equal than OLDID maximum no. is (NEWID - OLDID + 1), otherwise is ((0xFFFFFFFF - OLDID + 1) + NEWID + 1).

Query system report information

Code:	AS + 733
Symbolic:	bccSysReportInfo

This command will query information about system report on the remote device. Request has no parameters.

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	OLDID	Oldest system report item ID
4	U32	NEWID	Newest system report item ID
8	U32	NITEM	No. of system report items (in range from OLDID to NEWID, all included)
12	U32	SIZE	System report size (no. of storable items)
16	U32	REPID	System report content ID

On failure, a [bccNack](#)^[164] is received.

System report handling is based on the following statements:

- Item ID are allocated progressively, with an exception: at value 0xFFFFFFFF, next valid ID will be 0.
- The no. of items from OLDID and NEWID will be calculated as: if NEWID great or equal than OLDID maximum no. is (NEWID - OLDID + 1), otherwise is ((0xFFFFFFFF - OLDID + 1) + NEWID + 1).

RPE handling

These messages are used for general handling of the RPE extension of RTE.

Axes group handling:

- [bccRpeAxesGroupResolve](#)^[244], resolve an axes group
- [bccRpeAxesGroupInfo](#)^[235], get information for an axes group
- [bccRpeAxesGroupList](#)^[239], list available axes groups
- [bccRpeAxesGroupPositions](#)^[237], query positions for an axes group

Group Authority session handling:

- [bccRpeGASessionBegin](#)^[232], begin a Group Authority session
- [bccRpeGASessionEnd](#)^[234], end a Group Authority session

- [bccRpeGASessionWd](#)^[246], send watchdog a Group Authority session
- [bccRpeGASessionObjLoad](#)^[241], load an object to a Group Authority session
- [bccRpeGASessionObjSave](#)^[245], save an object from a Group Authority session
- [bccRpeGASessionObjStatus](#)^[242], query status for a object in a Group Authority session
- [bccRpeGASessionObjCmd](#)^[233], command for an object in a Group Authority session
- [bccRpeGASessionJogCmd](#)^[238], JOG command for an object in a Group Authority session
- [bccRpeGASessionUpdateObjPointP](#)^[247], update positions for an object's point in a Group Authority session
- [bccRpeGASessionUpdateObjStepInlinePointP](#)^[246], update positions for an object's step inline point in a Group Authority session

Begin a group authority session

Code:	AS + 1100
Symbolic:	bccRpeGASessionBegin

This command will begin a new group authority session for a specified axes group. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Session flags: (none)
4	U32	GID	Axes group ID
8	U32	WDT	Watchdog time [ms] (initial value)
12	U8[8]	SAK	Security Authorization Key (contact Robox SPA for more information)

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U32	MAXWDT	Maximum allowed watch dog time [ms] (0=any value)

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Request flags

NACK code	Description	Extra data
		2=Axes group ID
nackNotFound	Axes group not found	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

Command for an object in a group authority session

Code:	AS + 1106
Symbolic:	bccRpeGASessionObjCmd

This command will send a specific command to a object in a Group Authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB
6	U32	CMD	Command: 1=Set current: P0=Path ID, P1=Step ID 2=Start 3=Stop 4=Hold 5=Unhold 6=Step 7=Step to: P0=Path ID, P1=Step ID 8=Move to point: P0=Point ID 9=Move to inline point: P0=Path ID, P1=Step ID, P2=Point index
10	U32	CMDF	Command flags: 0x1 Backward direction (#1) 0x2 Initial position
14	U32	P0	Parametro 0 (opz)

Offset	Type	Label	Description
18	U32	P1	Parametro 1 (opz)
22	U32	p2	Parametro 2 (opz)

(#1) If CMDf flag 0x1 "Backward direction" is not specified, the "Forward direction" is assumed as default.

On success, a [bccAck](#)₁₆₄₁ is received with the no data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID 2=Object type 3=Command 4=Command flags 5=P0 6=P1 7=P2
nackNotFound	Session not found	
nackBusy	Session is busy, cannot execute the command	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

End a group authority session

Code:	AS + 1101
Symbolic:	bccRpeGASessionEnd

This command will end the specified existing group authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID

On success, a [bccAck](#)₁₆₄₁ is received with no data:

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID
nackNotFound	Session not found	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

Get information for an axes group

Code:	AS + 1001
Symbolic:	bccRpeAxesGroupInfo

This command will get information about an existing axes group of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	GID	Axes group ID

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	GID	Axes group ID (#1)
4	U32	FLAGS	Axes group flags (none)
8	U32	MJMODEL	Associated MJ model: 0=None 1=User defined 2=Bridge crane 3=Cross YZ 4=Cross generic 5=Custom 1 6=Custom 2 7=Custom 3 8=Linear parametric
12	U32	JCMODEL	Associated JC model: 0=Cartesian

Offset	Type	Label	Description
			1=User defined 2=Scara XY 3=Scara XYZ 4=Scara XYZW 5=Anthropomorph 1 6=Anthropomorph 2 7=Cylindric 8=Delta Robot 9=Custom 1 10=Custom 2 11=Custom 3 12=Anthropomorph 3
16	U32	PSETID	Associated power set ID
20	U16	ECMASK	Enabled cartesian axes mask: 0x1 X 0x2 Y 0x4 Z 0x8 A 0x10 B 0x20 C
22	U32	EJMASK	Enabled joint axes mask: 0x1 J1 0x2 J2 ... 0x10000000 J29 0x20000000 J30
26	U8	NAMESIZE	Size for field NAME (\0 termination included)
27	STRZ	NAME	Axes group name

(#1) Field GID repeated for compatibility with *bccRpeAxesGroupList* reply message format.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	

NACK code	Description	Extra data
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Axes group ID
nackNotFound	Axes group not found	
nackRpeNotInstalled	RPE firmware is not installed	

Get positions for an axes group

Code:	AS + 1003
Symbolic:	bccRpeAxesGroupPositions

This command will get positions information for the specified axes group of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U32	GID	Axes group ID

On success, a [bccAck](#)¹⁶⁴ is received with following data:

Offset	Type	Label	Description
0	U32	FLAGS	Flags (none)
4	U8	OPMODE	Axes group operating mode: 0 = Disabled 1 = Missing power 2 = Manual 3 = Execution 4 = C0 wait 5 = C0 execution 6 = Manual (before C0) 7 = Passive
5	U8	-	(reserved)
6	U32	STATE	Axes group state: 0x1 Axes in deceleration

Offset	Type	Label	Description
			0x2 Axes in position 0x4 Axes on hold 0x8 Axes on stop 0x10 Axes on emergency stop 0x20 Cartesian movement active
10	FLT	CP[6]	Cartesian positions
34	FLT	JP[30]	Joint positions

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Request flags 2=Axes group ID
nackNotFound	Axes group not found	
nackRpeNotInstalled	RPE firmware is not installed	

Jog command for an object in a group authority session

Code:	AS + 1107
Symbolic:	bccRpeGASessionJogCmd

This command will send a specific JOG command to a object in a Group Authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB
6	U8	TYPE	Jog type: 1=Joint 2=Cartesian (absolute) 3=Cartesian (tool)
7	U8	-	(reserved)

Offset	Type	Label	Description
8	U32	PMASK	Mask of axes for JOG[+]
12	U32	MMASK	Mask of axes for JOG[-]
16	U32	WDT	Comand watchdog time [ms]
20	U32	FLAGS	Flags: 0x1=Slow speed (10%) 0x2=Snap to grid 0x4=Single step

On success, a [bccAck](#)¹⁶⁴ is received with the no data.

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID 2=Object type 3=JOG type 4=JOG+ mask 5=JOG- mask 6=Watchdog time 7=Flags
nackNotFound	Session not found	
nackBusy	Session is busy, cannot execute the command	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

List available axes groups

Code:	AS + 1002
Symbolic:	bccRpeAxesGroupList

This command will list all available axes groups of the connected device: this is a standard [download transfer sequence](#)¹⁴.

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)

If initial request fails, [bccNack](#)₁₆₄ is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameters	U16 What illegal: 1=Flags
nackRpeNotInstalled	RPE firmware is not installed	

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U32	GID	Axes group ID
4	U32	FLAGS	Axes group flags (none)
8	U32	MJMODEL	Associated MJ model: 0=None 1=User defined 2=Bridge crane 3=Cross YZ 4=Cross generic 5=Custom 1 6=Custom 2 7=Custom 3 8=Linear parametric
12	U32	JCMODEL	Associated JC model: 0=Cartesian 1=User defined 2=Scara XY 3=Scara XYZ 4=Scara XYZW 5=Anthropomorph 1 6=Anthropomorph 2 7=Cylindric 8=Delta Robot 9=Custom 1 10=Custom 2

Offset	Type	Label	Description
			11=Custom 3 12=Anthropomorph 3
16	U32	PSETID	Associated power set ID
20	U16	ECMASK	Enabled cartesian axes mask: 0x1 X 0x2 Y 0x4 Z 0x8 A 0x10 B 0x20 C
22	U32	EJMASK	Enabled joint axes mask: 0x1 J1 0x2 J2 ... 0x10000000 J29 0x20000000 J30
26	U8	NAMESIZE	Size for field NAME (\0 termination included)
27	STRZ	NAME	Axes group name

Load an object to a group authority session

Code:	AS + 1103
Symbolic:	bccRpeGASessionObjLoad

This command will load an object to a group authority session on the connected device. this is a standard [data load sequence](#)^[18].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U8[16]	-	(reserved, see data load sequence specifications)
16	U32	FLAGS	Operation flags: (none)
20	U32	SESSID	Session ID

Offset	Type	Label	Description
24	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB

On success, a [bccAck](#)_[164] is received with at least the following data (ACKDATA):

Offset	Type	Label	Description
0	U8[16]	-	(reserved, see data load sequence specifications)

If initial request fails, [bccNack](#)_[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Size 2=Flags 3=Session ID 4=Object type
nackNotFound	Session not found	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

NOTE: to have more informations about the transferred data, see the .PLIB File Specifications in the documentation.

Query status for a object in a group authority session

Code:	AS + 1105
Symbolic:	bccRpeGASessionObjStatus

This command will query status for an object in a Group Authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB
6	U32	WDT	Watchdog time [ms]

Notes:

- This command also contain watchdog information, so if you periodically use this command you have no need to send the [bccRpeGASessionWd](#)^[248] message manually.

On success, a [bccAck](#)^[164] is received with the following data:

Offset	Type	Label	Description
0	U32	STATE	Object state: 0x1 Loaded 0x2 Execution ready 0x4 Execution active 0x8 Execution paused 0x10 Hold active 0x20 Stop request active 0x40 Hold request active 0x80 Step execution active 0x100 Backward execution active 0x200 Initial positioning active
4	U32	ECMD	Enabled object commands: 0x1 Set current 0x2 Start 0x4 Stop 0x8 Step 0x10 Hold 0x20 Unhold 0x40 Backward direction 0x80 Initial position 0x100 Joint JOG 0x200 Cartesian JOG 0x400 Update point quote
8	U32	PATHID	Executing path ID
12	U32	STEPID	Executing step ID
16	U8	PATHCP	Executing path completion %
17	U8	STEPCP	Executing step completion %

Offset	Type	Label	Description
18	FLT	PATHTLEN	Path total length [unit]
22	FLT	PATHELEN	Path executed length [unit]
26	FLT	PATHTTIME	Path total time [s]
30	FLT	PATHETIME	Path executed time [s]
34	FLT	STEPTLEN	Step total length [unit]
38	FLT	STEPELEN	Step executed length [unit]
42	FLT	STEPTTIME	Step total time [s]
46	FLT	STEPETIME	Step executed time [s]
50	FLT	TSPE	Current tangential speed [unit/s]
54	FLT	TACC	Current tangential acceleration [unit/s ²]

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID 2=Object type 3=Watchdog time
nackNotFound	Session not found	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

Resolve an axes group

Code:	AS + 1000
Symbolic:	bccRpeAxesGroupResolve

This command will try to resolve an existing axes group of the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	FLAGS	Request flags: (none)
4	U8	NAMESIZE	Size for field NAME (\0 termination included)
5	STRZ	NAME	Axes group NAME

On success, a [bccAck](#)^[164] is received with following data:

Offset	Type	Label	Description
0	U32	GID	Axes group ID

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Flags 2=Name
nackNotFound	Axes group not found	
nackRpeNotInstalled	RPE firmware is not installed	

Save an object from a group authority session

Code:	AS + 1104
Symbolic:	bccRpeGASessionObjSave

This command will save an object from a group authority session on the connected device. this is a standard [data save sequence](#)^[16].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U8[16]	-	(reserved, see data save sequence specifications)
16	U32	FLAGS	Operation flags: (none)
20	U32	SESSID	Session ID
24	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB

On success, a [bccAck](#)^[164] is received with at least the following data (ACKDATA):

Offset	Type	Label	Description
0	U8[16]	-	(reserved, see data save sequence specifications)

If initial request fails, [bccNack](#)^[164] is received. Specific errors

NACK code	Description	Extra data
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Size 2=Flags 3=Session ID 4=Object type
nackNotFound	Session not found	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

NOTE: to have more informations about the transferred data, see the .PLIB File Specifications in the documentation.

Update positions for an object's step inline point in a group authority session

Code:	AS + 1109
Symbolic:	bccRpeGASessionUpdateObjStepInlinePointP

This command will send a request to update positions for a specific object's inline point of the specified step in a Group Authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB
6	U8	PTYPE	Position type: 1=Joint 2=Cartesian
7	U8	-	(reserved)
8	U32	PATHID	Path ID

Offset	Type	Label	Description
12	U32	STEPID	Step ID
16	U32	POINTIX	Point index
20	U32	PMASK	Mask of updating positions
24	FLT	P[30]	Positions

On success, a [bccAck](#)₁₆₄₁ is received with the no data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID 2=Object type 3=Position type 4=Path ID 5=Step ID 6=Point index 7=Positionmask 10+i=Position[i]
nackNotFound	Not found	U16 What not found: 1=Session 2=Point
nackBusy	Session is busy, cannot execute the command	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

Update positions for an object's point in a group authority session

Code:	AS + 1108
Symbolic:	bccRpeGASessionUpdateObjPointP

This command will send a request to update positions for a specific point of the specified object in a Group Authority session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID

Offset	Type	Label	Description
4	U16	OBJTYPE	Object type: 1=Running PLIB 2=Testing PLIB
6	U8	PTYPE	Positions type: 1=Joint 2=Cartesian
7	U8	-	(reserved)
8	U32	POINTID	Point identifier
12	U32	PMASK	Mask of updating positions
16	FLT	P[30]	Positions

On success, a [bccAck](#)₁₆₄₁ is received with the no data.

On failure, a [bccNack](#)₁₆₄₁ is received. Specific errors:

NACK code	Description	Extra data
nackMissingArgs	Missing arguments	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Session ID 2=Object type 3=Quote type 4=Point ID 5=Quote mask 10+i=Quote Q[i]
nackNotFound	Not found	U16 What not found: 1=Session 2=Point
nackBusy	Session is busy, cannot execute the command	
nackRpeNotInstalled	RPE firmware is not installed	
nackNotAuthorized	Operation not authorized	

Watchdog for group authority session

Code:	1102
Symbolic:	bccRpeGASessionWd

This command will refresh session grant time (watchdog) for the specified testing session on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	SESSID	Session ID
4	U32	WDT	Watchdog time [ms]

Variable handling

These message are used getting and setting variable value for a connected device.

NOTE: all variable of type string are considered to be encoded as UTF-8.

Standard variables handling:

- [bccReadVar](#)^[264], read a variable
- [bccWriteVar](#)^[270], write a variable
- [bccForceVar](#)^[262], force a variable
- [bccReleaseVar](#)^[267], release a variable
- [bccReleaseAllVars](#)^[268], release all variables

All variables handling:

- [bccEnumVar](#)^[249], enumerate variables

Safe variables handling:

- [bccSafeReadVar](#)^[265], read a variable (safe)
- [bccSafeWriteVar](#)^[271], write a variable (safe)
- [bccSafeForceVar](#)^[263], force a variable (safe)
- [bccSafeReleaseVar](#)^[267], release a variable (safe)
- [bccSafeReleaseAllVars](#)^[268], release all variables (safe)

Dynamic variables handling:

- [bccRegisterVar](#)^[266], register a dynamic variable
- [bccUnregisterVar](#)^[269], un-register a dynamic variable
- [bccUnregisterAllVars](#)^[270], un-register all dynamic variables

Enumerate variables

Code:	AS + 314
Symbolic:	bccEnumVar

This command will enumerate variables definition: this is a standard [download transfer sequence](#)^[14].

REQDATA structure is the following:

Offset	Type	Label	Description
0	U32	FLAGS	Operation settings: 0x00000001 Enumerate standard variables

Offset	Type	Label	Description
			0x00000002 Enumerate user task variables 0x00000004 Enumerate logical variables 0x00000008 Force enumeration (even is same ID) 0x00000010 Enumerate dynamic variables
4	U32	VARSETID	Current locale variable set identification: if different, enumeration will occurs.

ACKDATA structure is the following:

Offset	Type	Label	Description
0	U32	COUNT	No. of item that will be received
4	U32	VARSETID	Real variable set identification

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackSameData	Remote variable set is the same of the local set. Enumeration is not needed.	
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Flags 2=Variable set ID

ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U16	PID	Process ID
2	U16	VARID	Variable ID
4	U8	CATEG	Variable category: 0=Not valid (illegal) 10=Ladder diagram alias

Offset	Type	Label	Description
			11=Memory variable (#1) 12=Standard variable 13=Logical structure 14=Logical variable 15=Dynamic structure/class 16=Dynamic variable
5	U8[23]	DATA	Data according variable category
28	STRZ	NAME	Variable name

(#1) Memory variables are considered to be represented in little-endian format.

If filed PID has value of 0xFFFF (group definition), ITEMDATA structure is the following:

Offset	Type	Label	Description
0	U16	PID	Task ID (0xFFFF)
2	U16	NSIZE	Name size (excluded terminal 0)
4	STRZ(NSIZE+1)	NAME	Group name
4+NSIZE+1	STRZ	TEXT	Group description

When a variable group definition is received (PID=0xFFFF), all subsequent received variables are relative to this group definition until a new group definition is received.

Notes:

- [bccData](#)_[165] (or [bccEndData](#)_[165]) messages will begin with pid = 0 and will be incremented by 1 at each message.
- [bccEndData](#)_[165] contain last item and after it the transfer is completed.
- The data transfer could be aborted at any time with a [bccAbort](#)_[165] command.
- The maximum repeat counter is calculated as the product array index size: if none of 3 index are defined, maximum repeat counter is assumed to have a default value of 1.
- Variable definition with childs (referenced by a PPID e PVARID) must be category 11 (Memory variable) and type 11 (Structure).

Specific DATA for category 10 (ladder diagram alias)

Offset	Type	Label	Description
+0	U16	TYPE	Alias type: 0=not valid (illegal) 1=logical input channel

Offset	Type	Label	Description
			2=logical input word 16bit 3=logical input word 16bit, bit access 4=logical input word 32bit 5=logical input word 32bit, bit access 6=logical output channel 7=logical output word 16bit 8=logical output word 16bit, bit access 9=logical output word 32bit 10=logical output word 32bit, bit access 11=integer volatile register 32bit 12=integer volatile register 32bit, bit access 13=integer non volatile register 32bit 14=integer non volatile register 32bit, bit access 15=real volatile register 64bit 16=real non volatile register 64bit 17=alarm mask 18=alarm mask, bit access
+2	U16	IX	Variable index
+4	U16	BIX	Bit index for bit access

Specific DATA for category 11 (memory variable)

Offset	Type	Label	Description
+0	U8	TYPE	Memory x86 variable type: 0=not valid (illegal)

Offset	Type	Label	Description
			1=unsigned 8bit 2=signed 8bit 3=unsigned 16bit 4=signed 16bit 5=unsigned 32bit 6=signed 32bit 7=double (64bit) 8=float (32bit) 9=boolean (#1) 10=char array 11=structure definition 12=string (0 terminated) (#3) 13=bit 14=timer (#2) 15=counter (#2) 16=(reserved) 17=unsigned 64bit 18=signed 64bit
+1	U32	SIZE	Variable base size. In case of string type (12), this field indicate the maximum string buffer size, 0 terminator included.
+5	U16	DIM0	Array index 0 size
+7	U16	DIM1	Array index 1 size
+9	U16	DIM2	Array index 2 size
+11	U32	FLAG	Variable flags: 0x00000001 Has array 0 size 0x00000002 Has array 1 size 0x00000004 Has array 2 size 0x00000008 Has repeat counter 0x00000010 Has parent variable

Offset	Type	Label	Description
+15	U32	ADDR	Base address or parent offset (if has parent flag)
+19	U16	PPID	Parent Process ID
+21	U16	PVARID	Parent Variable ID

(#1) Boolean type (9) is considered having an unsigned 8bit storage.

(#2) Timer type (14) and counter type (15) have their own predefined structure.

(#3) RDE3 does not currently support this data type.

Notes:

- Flag 'Has array 2 size' implies flag 'Has array 1 size' and this implies flag 'Has index 0'.

Specific DATA for category 12 (standard variable)

Offset	Type	Label	Description
+0	U16	TYPE	Variable type: 0=not valid (illegal) 1=logical input channel 2=logical input word 16bit 3=logical input word 32bit 4=logical output channel 5=logical output word 16bit 6=logical output word 32bit 7=integer volatile register 32bit 8=integer non volatile register 32bit 9=real volatile register 64bit 10=real non volatile register 64bit 11=alarm mask 32bit 12=string volatile register (#1) 13=string non volatile register (#1) 14=integer parameter register 16bit

Offset	Type	Label	Description
			15=integer parameter register 32bit 16=real parameter register 64bit 17=float parameter register 32bit 18=integer axis parameter 16bit 19=integer axis parameter 32bit 20=real axis parameter 64bit 21=float axis parameter 32bit 22=alarm code in alarm stack 23=alarm text in alarm stack (#1) 24=integer volatile register 64bit 25=integer non volatile register 64bit 26=integer parameter register 64bit 27=integer axis parameter 64bit
+2	U16	BASE	Base index
+4	U16	DIM	Array size
+6	I16	OFFSET	Array offset
+8	U16	AXIS	Axis count
+10	U32	FLAGS	Variable flags: 0x00000001 Has array size 0x00000002 Has axis count 0x00000004 Has array offset
+14	U16	SSIZE	String size (#1)

(#1) Use SSIZE field to text string size

Notes:

- Usually these variables should be relative to a group named SYS (aka system variables).
- General fields PID and VARID are not used by this category.

Specific DATA for category 13 (logical structure)

Offset	Type	Label	Description
+0	U32	TYPEID	<p>Structure type ID:</p> <p>0x00000000-0x0000FFFF predefined</p> <p>0x00010000-0xFFFFFFFF user defined</p> <p>Predefined types ID are:</p> <p>0x00000000=not valid (illegal)</p> <p>0x00000001=signed 8bit</p> <p>0x00000002=unsigned 8bit</p> <p>0x00000003=signed 16bit</p> <p>0x00000004=unsigned 16bit</p> <p>0x00000005=signed 32bit</p> <p>0x00000006=unsigned 32bit</p> <p>0x00000007=signed 64bit</p> <p>0x00000008=unsigned 64bit</p> <p>0x00000009=string (#1)</p> <p>0x0000000A=float (32bit)</p> <p>0x0000000B=double (64bit)</p> <p>0x0000000C=boolean</p> <p>0x0000000D=timer</p> <p>0x0000000E=counter</p> <p>0x0000000F=alias (ladder)</p> <p>0x00000010=edge (ladder)</p> <p>0x00000011=(reserved, ex power set)</p>

Offset	Type	Label	Description
			0x00000012=(reserved, ex axes group) 0x00000013=(reserved, ex path) 0x00000014=(reserved, ex point_l)
+4	U16	DIM0	Array 0 size (0=none) or string size (#1)
+6	U16	DIM1	Array 1 size (0=none)
+8	U16	DIM2	Array 2 size (0=none)
+10	U16	OFF0	Array 0 index offset
+12	U16	OFF1	Array 1 index offset
+14	U16	OFF2	Array 2 index offset
+16	U32	FLAGS	Flags: 0x00000001 Begin of user defined structure 0x00000002 End of user defined structure 0x00000004 Disable bit access

(#1) DIM0 value 0 assume meaning of default string size

Notes:

- A user defined structure declaration must begin with the "begin of user defined structure" flag and must end with the "end of user defined structure" flag: a void structure declaration (both flags specified) is allowed, but is useless.
- General fields PID and VARID are not used by this category.
- Fields DIM0, DIM1 and DIM2 (and related field OFFx and flags "Has index N") are not used with the "Begin of user defined structure" flag.
- User defined structures are relative to the current group only: when it changes, structures are cleared.
- User defined structures cannot have nested declarations, but can have items with TYPEID related to another user defined structure.
- DIM2 > 0 implies DIM1 > 0 and DIM1 > 0 implies DIM0 > 0.
- Type 0x00000009 (string) has DIM0 as string size (0=default 128 character string): string array indexes are from DIM1 field.
- The disable bit access flag is intended for those TYPEID that normally provide the data bit access (like integer values).

Specific DATA for category 14 (logical variable)

Offset	Type	Label	Description
+0	U32	TYPEID	<p>Structure type ID: 0x00000000- 0x0000FFFF predefined 0x00010000- 0xFFFFFFFF user defined</p> <p>Predefined types ID are: 0x00000000=not valid (illegal) 0x00000001=signed 8bit 0x00000002=unsigne d 8bit 0x00000003=signed 16bit 0x00000004=unsigne d 16bit 0x00000005=signed 32bit 0x00000006=unsigne d 32bit 0x00000007=signed 64bit 0x00000008=unsigne d 64bit 0x00000009=string (#1) 0x0000000A=float (32bit) 0x0000000B=double (64bit) 0x0000000C=boolean 0x0000000D=timer 0x0000000E=counter 0x0000000F=alias (ladder) 0x00000010=edge (ladder) 0x00000011=(reserve d, ex power set) 0x00000012=(reserve d, ex axes group)</p>

Offset	Type	Label	Description
			0x00000013=(reserved, ex path) 0x00000014=(reserved, ex point_l)
+4	U16	DIM0	Array 0 size (0=none) or string size (#1)
+6	U16	DIM1	Array 1 size (0=none)
+8	U16	DIM2	Array 2 size (0=none)
+10	U16	OFF0	Array 0 index offset
+12	U16	OFF1	Array 1 index offset
+14	U16	OFF2	Array 2 index offset
+16	U32	FLAGS	Variable flags: 0x00000001 Disable bit access

(#1) DIM0 value 0 assume meaning of default string size

Notes:

- User structure type ID are defined with category 13 (logical structure) that have to be defined before the category 14 (logical variable).
- General fields PID and VARID are not used by this category.
- DIM1 > 0 implies DIM0 > 0, and DIM2 > 0 implies DIM1 > 0 and DIM0 > 0.
- Type 0x00000009 (string) has DIM0 as string size (0=default 128 character string): string array indexes are from DIM1 field.
- The disable bit access flag is intended for those TYPEID that normally provides the data bit access (like integer values).

Specific DATA for category 15 (dynamic structure)

Offset	Type	Label	Description
+0	U32	TYPEID	Structure type ID: 0x00000000-0x00000FFF predefined 0x00001000-0xFFFFFFFF user defined Predefined types ID are: 0=not valid (illegal) 1=signed 8bit

Offset	Type	Label	Description
			2=signed 16bit 3=signed 32bit 4=signed 64bit 5=unsigned 8bit 6=unsigned 16bit 7=unsigned 32bit 8=unsigned 64bit 9=float (32bit) 10=real (64bit) 11=bool 12=string
+4	U16	DIM0	Array 0 size (0=none) or string size (#1)
+6	U16	DIM1	Array 1 size (0=none)
+8	U16	DIM2	Array 2 size (0=none)
+10	U32	FLAGS	Flags: 0x00000001 Structured begin 0x00000002 Structure end 0x00000004 Disable bit access 0x00000008 Static structure item 0x00000010 Static-instance structure item 0xF0000000 Structure type (mask): 0x0 = structure 0x1 = class 0x2 = namespace 0x3 = interface
+14	U32	PTYPEID	Parent structure ID (0=none)

(#1) DIM0 value 0 assume meaning of default string size

Notes:

- A used defined structure/class declaration must begin with the 0x1 flag (structure begin) and must end with the 0x2 flag (structure end): a void structure declaration (both flags specified) is allowed, but is useless.

- General fields PID and VARID are not used by this category.
- Field TYPEID, if flags 0x1 (structure begin) is set, contains the source type-id of the structure itself: in all other cases, the field refer to the item target type-id.
- Field PTYPEID is used only when the flag 0x1 (structure begin) is set
- Fields DIM0, DIM1 and DIM2 (and related field OFFx and flags "Has index N") are not used when the flag 0x1 (structure begin) is set
- User defined structure are relative to the current group only: when it change, structure are cleared.
- User defined structure cannot have nested declaration, but can have item with TYPEID related to another user defined structure.
- DIM2 > 0 implies DIM1 > 0 and DIM1 > 0 implies DIM0 > 0.
- Type 13 (string) has DIM0 as string size (0=default 128 character string): string array indexes are from DIM1 field.
- The disable bit access flag is intended for those TYPEID that normally provides the data bit access (like integer values).
- The static structure item flag is valid only if the flags 0x1 (structure begin) is not set.
- The structure type value is valid only if the flags 0x1 (structure begin) is set.

Specific DATA for category 16 (dynamic variable)

Offset	Type	Label	Description
+0	U32	TYPEID	<p>Variable type ID:</p> <p>0x00000000-0x00000FFF predefined</p> <p>0x00001000-0xFFFFFFFF user defined</p> <p>Predefined types ID are:</p> <p>0=not valid (illegal)</p> <p>1=signed 8bit</p> <p>2=signed 16bit</p> <p>3=signed 32bit</p> <p>4=signed 64bit</p> <p>5=unsigned 8bit</p> <p>6=unsigned 16bit</p> <p>7=unsigned 32bit</p> <p>8=unsigned 64bit</p> <p>9=float (32bit)</p> <p>10=real (64bit)</p> <p>11=bool</p>

Offset	Type	Label	Description
			12=string
+4	U16	DIM0	Array 0 size (0=none) or string size (#1)
+6	U16	DIM1	Array 1 size (0=none)
+8	U16	DIM2	Array 2 size (0=none)
+10	U32	FLAGS	Flags: 0x00000001 (reserved) 0x00000002 (reserved) 0x00000004 Disable bit access 0x00000008 Static variable 0x00000010 Static-instance variable

(#1) DIM0 value 0 assume meaning of default string size

Notes:

- User dynamic structure typeid's are defined with category 15 (dynamic structure): the dynamic variable can be defined before its declaration.
- General fields PID and VARID are not used by this category.
- DIM1 > 0 implies DIM0 > 0, and DIM2 > 0 implies DIM1 > 0 and DIM0 > 0.
- Type 13 (string) has DIM0 as string size (0=default 128 character string): string array indexes are from DIM1 field.
- The disable bit access flag is intended for those TYPEID that normally provides the data bit access (like integer values).

Force a variable

Code:	AS + 312
Symbolic:	bccForceVar

This command will force the value of a variable (for example force the state of input channel). Request parameters are the following:

Offset	Type	Label	Description
0	VAR ₉	VARID	Variable identification
10	U8[]	DATA	Variable data, according VARID field.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only or predefined	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackCantForce	Variable value cannot be forced	
nackDataUnderflow	Variable data too short	
nackNotInitialized	Variable is not initialized	

Force a variable (safe)

Code:	AS + 318
Symbolic:	bccSafeForceVar

This command will force the value of a variable in a safe way (for example force the state of input channel). Request parameters are the following:

Offset	Type	Label	Description
0	U32	VARSETID	Variable set identification
4	VAR ₉	VARID	Variable identification
14	U8[]	DATA	Variable data, according VARID field.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only or predefined	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter

NACK code	Description	Extra data
		5=Bit index 6=Size
nackCantForce	Variable value cannot be forced	
nackDataUnderflow	Variable data too short	
nackIllegalVarsetId	Illegal variable set identification	
nackNotInitialized	Variable is not initialized	

Read a variable

Code:	AS + 310
Symbolic:	bccReadVar

This command will read value for a variable. Request parameters are the following:

Offset	Type	Label	Description
0	VAR ₉	VARID	Variable identification

On success, a [bccAck](#)₁₆₄ is received with following data:

Offset	Type	Label	Description
0	U8[]	DATA	Variable data, according VARID field.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadError	Error reading variable	
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackDataOverflow	Variable data too long (exceed 255)	
nackWriteOnly	Variable is write only, cannot read.	

NACK code	Description	Extra data
nackNotInitialized	Variable is not initialized	

For more information about variable types, see table of [standard variables](#)^[289].

Read a variable (safe)

Code:	AS + 316
Symbolic:	bccSafeReadVar

This command will read value for a variable in a safe way. Request parameters are the following:

Offset	Type	Label	Description
0	U32	VARSETID	Variable set identification
4	VAR ^[9]	VARID	Variable identification

On success, a [bccAck](#)^[164] is received with following data:

Type	Label	Description
U8[]	DATA	Variable data, according VARID field.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackReadError	Error reading variable	
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackDataOverflow	Variable data too long (exceed 255)	
nackWriteOnly	Variable is write only, cannot read	
nackIllegaleVarsetId	Illegal variable set identification	
nackNotInitialized	Variable is not initialized	

For more information about variable types, see table of [standard variables](#)^[289].

Register a dynamic variable

Code:	AS + 321
Symbolic:	bccRegisterVar

This command will register a dynamic variable on the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNERID	Owner identification
4	U32	VARSETID	Variable set identification
8	U16	FLAGS	Registration flags: 0x1=Single use registration
10	STRZ	KEY	Variable key

On success, a [bccAck](#)_[164] is received with the following data:

Offset	Type	Label	Description
0	U8[6]	VIDD	Variable identification data
6	U16	FLAGS	Registration reply flags: 0x1=Unlimited TTL
8	U32	TTL	Variable time to live [ms]

On failure, a [bccNack](#)_[164] is received. Specific errors:

NACK code	Description	Extra data
nackIllegalVarsetId	Illegal variable set identification	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=OwnerId 2=Key
nackNotFound	Variable key not found	
nackIllegalDataType	Variable data type not handled	

The dynamic variables always have a duration (or expiration): this duration is to be considered as the maximum time of said inactivity (TTL).

Format of the KEY field

The format of the KEY field is the following:

```
[<prefix>] [; [<repeat>]]=<field0>[;<field1>[...]]
```

where <prefix> is the name of the RVAR prefix (or * in case of no prefix) , <repeat> is the number of the consecutive data, where possible (default value is 1).

The format of the <fieldX> data is the following:

```
[<prefix>] [; [<repeat>]]=<field0>[;<field1>[...]]
```

where <name> is the name of the field and <indexX> is the optional X array index. Note that: value <index2> implies <index1> and <index1> implies <index0>.

Release a variable

Code:	AS + 313
Symbolic:	bccReleaseVar

This command will release the value of a variable (for example release the state of an input channel). Request parameters are the following:

Offset	Type	Label	Description
0	VAR ¹⁶⁴	VARID	Variable identification

On success, a [bccAck](#)¹⁶⁴ is received with no data.

On failure, a [bccNack](#)¹⁶⁴ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only or predefined	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackCantRelease	Variable value cannot be released	
nackNotInitialized	Variable is not initialized	

Release a variable (safe)

Code:	AS + 319
Symbolic:	bccSafeReleaseVar

This command will release the value of a variable (for example release the state of an input channel). Request parameters are the following:

Offset	Type	Label	Description
0	U32	VAR SETID	Variable set identification
4	VAR ₁₆₄	VARID	Variable identification

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only or predefined	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackCantRelease	Variable value cannot be released	
nackIllegalVarsetId	Illegal variable set identification	
nackNotInitialized	Variable is not initialized	

Release all variables

Code:	AS + 315
Symbolic:	bccReleaseAllVars

This command will release the value for all forced variables. Request has no parameters.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. There are no specific errors.

Release all variables (safe)

Code:	AS + 320
Symbolic:	bccSafeReleaseAllVars

This command will release the value for all forced variables in a safe way. Request parameters are the following:

Offset	Type	Label	Description
0	U32	VARSETID	Variable set identification

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalVarsetId	Illegal variable set identification	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size

Un-register a dynamic variable

Code:	AS + 322
Symbolic:	bccUnregisterVar

This command will un-register a dynamic variable from the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNERID	Owner identification
4	U32	VARSETID	Variable set identification
8	U8[6]	VIDD	Variable identification data

On success, a [bccAck](#)₁₆₄ is received with no data:

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalVarsetId	Illegal variable set identification	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=OwnerId 2=Key
nackNotFound	Variable key not found	

Un-register all dynamic variables

Code:	AS + 323
Symbolic:	bccUnregisterAllVars

This command will un-register all variables for a specific owner from the connected device. Request parameters are the following:

Offset	Type	Label	Description
0	U32	OWNERID	Owner identification
4	U32	VARSETID	Variable set identification

On success, a [bccAck](#)₁₆₄ is received with no data:

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackIllegalVarsetId	Illegal variable set identification	
nackIllegalArgs	Illegal arguments	U16 What illegal: 1=OwnerId

Write a variable

Code:	AS + 311
Symbolic:	bccWriteVar

This command will write value for a variable. Request parameters are the following:

Offset	Type	Label	Description
0	VAR ₉	VARID	Variable identification
10	U8[]	DATA	Variable data, according VARID field.

On success, a [bccAck](#)₁₆₄ is received with no data.

On failure, a [bccNack](#)₁₆₄ is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only (cannot write)	
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Type 2=Index 3=Address

NACK code	Description	Extra data
		4=Repeat counter 5=Bit index 6=Size
nackDataUnderflow	Variable data too short	
nackNotInitialized	Variable is not initialized	

For more information about variable types, see table of [standard variables](#)^[289].

Write a variable (safe)

Code:	AS + 317
Symbolic:	bccSafeWriteVar

This command will write value for a variable in a safe way. Request parameters are the following:

Offset	Type	Label	Description
0	U32	VARSETID	Variable set identification
4	VAR ^[9]	VARID	Variable identification
14	U8[]	DATA	Variable data, according VARID field.

On success, a [bccAck](#)^[164] is received with no data.

On failure, a [bccNack](#)^[164] is received. Specific errors:

NACK code	Description	Extra data
nackReadOnly	Variable read-only (cannot write)	
nackIllegalArgs	Illegal parameter	U16 What illegal: 1=Type 2=Index 3=Address 4=Repeat counter 5=Bit index 6=Size
nackDataUnderflow	Variable data too short	
nackIllegalVarsetId	Illegal variable set identification	
nackNotInitialized	Variable is not initialized	

For more information about variable types, see table of [standard variables](#)^[289].

Network interfaces

Network interfaces

Network interface are defined as general software application or hardware device that give the possibility to communicate with Robox Devices with different criteria than local connection (e.g. RS232 connections).

Interface types

Mainly there are three types of network interfaces:

1. Direct TCP/IP network connection, into a local area network (LAN) or more general wide area network (WAN), like Internet, via ethernet connection (see [Ethernet network example](#)^[272]).
2. Indirect TCP/IP via a Dialup Modem connection, using the standard PPP protocol (see [Modem network example](#)^[273]).
3. Direct Dialup Modem connection, using a raw protocol on the RS232 connector.

Integrated ETH port

Newer generation of RBXM (CPU586 and CPUG2) devices have an integrated ETH port that allow direct TCP/IP communication.

The integrated ETH interface allow TCP connection through connection to port 8000, with BCC/31 over DLE/CRC16 transport.

Note: if configured, any TCP connection must authenticate before BCC/31 message can be processed : for more information look at [bccNetLogin](#)^[200] command. Message not allowed (even after authentication), are denied with a [bccNack](#)^[164] response with nackNotAuthorized error code.

NET.INT. expansion board

The NET.INT. is no more in use since newer RBXM devices have their own integrated ETH port.

Ethernet network example

This example explain how can be created a classical local area network (LAN) with some RBXM and PC connected, and how can be reached from various external connection.

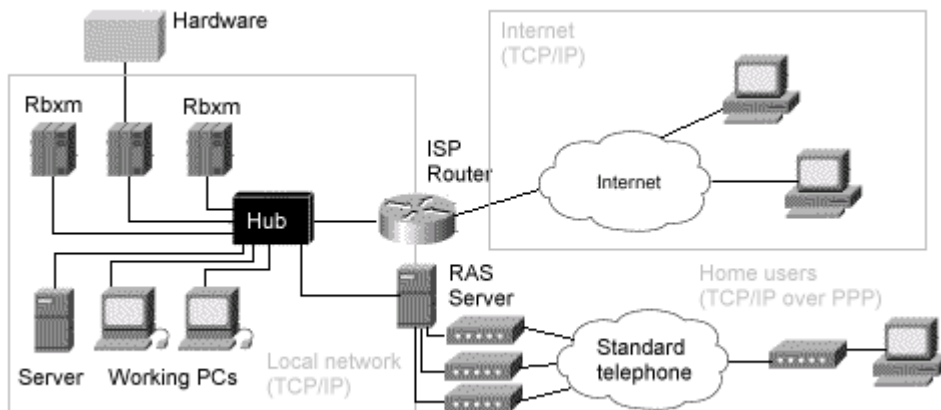
All device in the LAN area are connected to an Hub via RJ-45 Ethernet cable: the protocol used in this network is TCP/IP over ethernet.

Connecting a properly configured ISP router to the HUB (via RJ-45 Ethernet) you can allow connection from the Internet.

The Internet connection, provided by your ISP, can be established with various type of transport or physical devices (like ADSL, HDSL, CDN, ATM, etc): all these will transport the TCP/IP protocol to the router with their own physical transport protocol. The router will communicate with local network via TCP/IP over ethernet.

By having a RAS (remote access) server access, the network can be configured to allow dial-in access from a standard telephone line. A remote authorized user with a standard modem can open a remote access connection to your telephone number and have access to your local network.

In this case, the remote PC will communicate up to the RAS server in TCP/IP over PPP; next the remote network transport will be applied.



Modem network example

In this example we have a RBXM with a NET.INT. board installed: a standard modem is connected to the RS232 port of the NET.INT. board itself.









From your PC, using remote access (PPP protocol), you can activate the modem connection on a standard RTG line (default telephone line): at this point your PC will communicate with the RBXM via TCP/IP over PPP, allowing to access any TCP application on the NET.INT.



Miscellaneous

Messages map

Here a map of all basic BCC/31 actually defined messages:

Symbolic	Code	Description
	0	(reserved for internal use)
bccAck 	1	General acknowledge
bccNack 	2	General not acknowledge
bccData 	3	Binary data
bccAbort 	AS+4	Abort command
bccAborted 	5	Abort event
bccEndData 	6	End of data
bccNoData 	7	No data
bccIBlock 	8	Software interblock

Symbolic	Code	Description
bccCheck ^[166]	AS+9	Check point command
bccBusy ^[166]	10	System busy event
bccCompleted ^[167]	11	Completion event
bccWait ^[167]	12	Wait more
bccReady ^[167]	13	Ready
	14-37	(reserved)
bccPing ^[215]	AS+38	Ping command
bccPong ^[215]	39	Ping answer
	40-99	(reserved for old BCC/30 transfer commands)
bccFlashFileLoad ^[142]	AS+100	Load a file to a flash folder
bccFlashFileSave ^[159]	AS+101	Save file from a flash folder
bccFlashDir ^[148]	AS+102	Query flash folder contents
bccFlashFileDelete ^[140]	AS+103	Delete flash files
bccFlashFileRename ^[159]	AS+104	Rename a flash file
	105-106	(reserved)
bccFlashTree ^[158]	AS+107	Query flash folder tree
bccFlashFormat ^[141]	AS+108	Format a flash
bccFlashInfo ^[153]	AS+109	Query flash information
bccFlashFileInfo ^[150]	AS+110	Query flash file information
bccFlashList ^[158]	AS+111	Query flashes list
bccFlashInit ^[138]	AS+112	Define and initialize flashes
bccFlashFolderCreate ^[137]	AS+113	Create a flash folder
bccFlashFolderDelete ^[140]	AS+114	Delete a flash folder
bccFlashFolderInfo ^[156]	AS+115	Query flash folder information
bccFlashInfoByFolder ^[155]	AS+116	Query flash information by folder
bccFlashDisk ^[146]	AS+117	Handle disk flashes
bccFlashSetAttributes ^[162]	AS+118	Set flash file/folder attributes
	119-199	(reserved)

Symbolic	Code	Description
bccGetIC 	AS+200	Get input channel (deprecated)
bccGetIW16 	AS+201	Get input word 16bit (deprecated)
bccGetOC 	AS+202	Get output channel (deprecated)
bccGetOW16 	AS+203	Get output word 16bit (deprecated)
bccSetOC 	AS+204	Set output channel (deprecated)
bccSetOW16 	AS+205	Set output word 16bit (deprecated)
bccForceIC 	AS+206	Force input channel (deprecated)
bccForceIW16 	AS+207	Force input word 16bit (deprecated)
bccForceOC 	AS+208	Force output channel (deprecated)
bccForceOW16 	AS+209	Force output word 16bit (deprecated)
bccReleaseIC 	AS+210	Release input channel (deprecated)
bccReleaseIW16 	AS+211	Release input word 16bit (deprecated)
bccReleaseOC 	AS+212	Release output channel (deprecated)
bccReleaseOW16 	AS+213	Release output word 16bit (deprecated)
bccReleaseAllIC 	AS+214	Release all input channel (deprecated)
bccReleaseAllOC 	AS+215	Release all output channel (deprecated)
	216-299	(reserved)
bccGetR16 	AS+300	Get 16bit integer register (deprecated)
bccSetR16 	AS+301	Set 16bit integer register (deprecated)
bccGetR32 	AS+302	Get 32bit integer register (deprecated)

Symbolic	Code	Description
bccSetR32 ^[222]	AS+303	Set 32bit integer register (deprecated)
bccGetRR ^[219]	AS+304	Get real register (double) (deprecated)
bccSetRR ^[223]	AS+305	Set real register (double) (deprecated)
bccGetRRF ^[218]	AS+306	Get real register (float) (deprecated)
bccSetRRF ^[222]	AS+307	Set real register (float) (deprecated)
bccGetSR ^[220]	AS+308	Get string register (deprecated)
bccSetSR ^[224]	AS+309	Set string register (deprecated)
bccReadVar ^[264]	AS+310	Read value of a variable
bccWriteVar ^[270]	AS+311	Write value of a variable
bccForceVar ^[262]	AS+312	Force value of a variable
bccReleaseVar ^[267]	AS+313	Release value of a variable
bccEnumVar ^[249]	AS+314	Enumerate variables
bccReleaseAllVars ^[268]	AS+315	Release value of all forced variables
bccSafeReadVar ^[265]	AS+316	Safe read value of a variable
bccSafeWriteVar ^[271]	AS+317	Safe write value of a variable
bccSafeForceVar ^[263]	AS+318	Safe force value of a variable
bccSafeReleaseVar ^[267]	AS+319	Safe release value of a variable
bccSaveReleaseAllVars ^[268]	AS+320	Safe release value of all forced variables
bccRegisterVar ^[266]	AS+321	Register a dynamic variable
bccUnregisterVar ^[269]	AS+322	Un-register a dynamic variable
bccUnregisterAllVars ^[270]	AS+323	Un-register all dynamic variables
	324-399	(reserved)
bccMonCreate ^[189]	AS+400	Create monitor
bccMonDestroy ^[191]	AS+401	Destroy monitor

Symbolic	Code	Description
bccMonStart 	AS+402	Start monitor data stream
bccMonStop 	AS+403	Stop monitor data stream
bccMonStatus 	AS+404	Query monitor status
bccMonList 	AS+405	Query monitor list
bccMonQuick 	AS+406	Quick monitor
bccMonWd 	407	Monitor watchdog
bccMonWrite 	AS+408	Monitor write
bccMonStatInfo 	AS+409	Query monitors statistical information
	410-419	(reserved)
bccOscCreate 	AS+420	Create oscilloscope
bccOscDestroy 	AS+421	Destroy oscilloscope
bccOscStart 	AS+422	Start oscilloscope data stream
bccOscStop 	AS+423	Stop oscilloscope data stream
bccOscStatus 	AS+424	Query oscilloscope status
bccOscList 	AS+425	Query oscilloscope list
bccOscStatInfo 	AS+426	Query oscilloscopes statistical information
bccOscWd 	427	Oscilloscope watchdog
	428-499	(reserved)
bccGetAlarm 	AS+500	Get alarm stack
bccSetAlarm 	AS+501	Set user alarm
bccResetAlarm 	AS+502	Reset alarm stack
bccGetDateTime 	AS+503	Get current date and time
bccSetDateTime 	AS+504	Set current date and time
bccSoftwareReset 	AS+505	Request a software reset
bccHardwareReset 	AS+506	Request a hardware reset
bccSysInfo 	AS+507	Query system information
bccGetMode 	AS+508	Get current mode
bccSetMode 	AS+509	Set current mode

Symbolic	Code	Description
	AS+510	(reserved)
bccAutoConfig 	AS+511	Request device (self) auto configuration
bccCMosReset 	AS+512	Request a CMOS Ram reset
bccSysProcessList	AS+513	Query device system process list
bccProductInfo	AS+514	Query product specific information
bccAlarmHInfo 	AS+515	Query alarm history
bccAlarmHList 	AS+516	Get alarm history
bccAlarmHCmd 	AS+517	Command for alarm history
bccAsciiCmd 	AS+518	Execute a generic ASCII command
bccAlarmSInfo 	AS+519	Query alarm stack information
bccAlarmSGet 	AS+520	Query single alarm stack entry
bccAlarmSList 	AS+521	Query all alarm stack entries
bccAlarmSCmd 	AS+522	Command for alarm stack
bccResolveProcObject 	AS+523	Resolve /proc object.
bccAlarmHListE 	AS+524	Get enhanced alarm history
	525-529	(reserved)
bccOOWSessionBegin 	AS+530	Begin an OOW session
bccOOWSessionEnd 	AS+531	End an OOW session
bccOOWSessionQueryInfo 	AS+532	Query information for an OOW session
	533-599	(reserved)
bccNetLogin 	AS+600	Network login
bccNetLogout 	AS+601	Network logout
bccNetUserList 	AS+602	Query user list
bccNetUserCreate 	AS+603	Create new user
bccNetUserDelete 	AS+604	Delete existing user
bccNetUserChange 	AS+605	Change existing user settings

Symbolic	Code	Description
	AS+606	(reserved; ex bccNetMptSave)
	AS+607	(reserved; ex bccNetMptLoad)
bccNetInfo 	AS+608	Query network information
bccNetClientList 	AS+609	Query client list
bccNetClientKill 	AS+610	Kill a specific client
bccNetStats 	AS+611	Query network statistics
bccNetClientKasSessionBegin 	AS+612	Start a keep alive session for the client
bccNetClientKasSessionEnd 	AS+613	Stop a keep alive session for the client
bccNetClientKasSessionInfo 	AS+614	Query info for a keep alive session for the client
	615-699	(reserved)
bccProcessList 	AS+700	List available remote process
bccProcessInfo 	AS+701	Query process specific information
bccProcessDbgCmd 	AS+702	Execute a process debug command
bccProcessGetTrace 	AS+703	Ask trace information for a process
bccProcessStatus 	AS+704	Ask R/T status for a process
bccProcessCmd 	AS+705	Execute a process command
bccProcessFlashInfo 	AS+706	Query process flash information
bccProcessInspect 	AS+707	Inspect contents for a process
bccProgressGetDebugContext 	AS+708	Query debug context for a process
	709	(reserved)
bccDebugStart 	AS+710	Start a debug session
bccDebugStop 	AS+711	Stop a debug session
bccDebugWd 	712	Debug session watchdog
	713-719	(reserved)
bccBreakpAdd 	AS+720	Add a breakpoint

Symbolic	Code	Description
bccBreakpDel 	AS+721	Delete multiple breakpoint
bccBreakpList 	AS+722	List defined breakpoints
bccBreakpInfo 	AS+723	Ask information for a breakpoint
bccBreakpStatus 	AS+724	Ask status for a breakpoint
	725-729	(reserved)
bccReportInfo 	AS+730	Query report information
bccReportList 	AS+731	Get report
bccReportCmd 	AS+732	Command for report
bccSysReportInfo 	AS+733	Query system report information
bccSysReportList 	AS+734	Get system report
bccSysReportCmd 	AS+735	Command for system report
bccOsAttachedFList 	AS+736	List OS attached functions
	737-749	(reserved)
bccCanObjRead 	AS+750	Read a CANopen object
bccCanObjWrite 	AS+751	Write a CANopen object
bccCanNmtRead 	AS+752	Read one or more CANopen NMT
bccCanNmtWrite 	AS+753	Write one or more CANopen NMT
bccCanEmcyRead 	AS+754	Read a CANopen EMCY message
bccCanEmcyInfo 	AS+755	Query CANopen EMCY information
bccCanRbxChDiag 	AS+756	Query Robox CANopen channel diagnostic
bccCanRbxWsDiag 	AS+757	Query Robox CANopen workstation diagnostic
bccCanPdoRead 	AS+758	Read element from Tx/Rx CANopen PDO
bccCanC402Info 	AS+759	Query CANopen C402 information
bccCoeObjRead 	AS+760	Read a COE object
bccCoeObjWrite 	AS+761	Write a COE object

Symbolic	Code	Description
bccEcatNmtRead 	AS+762	Read an EtherCAT NMT
bccEcatNmtWrite 	AS+763	Write an EtherCAT NMT
	764-769	(reserved)
bccObjBlockList 	AS+770	List object blocks
	771-799	(reserved)
bccLadTaskLoad 	AS+800	Load ladder task to device
bccLadTaskSave 	AS+801	Save ladder task from device
bccLadLiveLoad 	AS+802	Load live changes
bccLadLiveTest 	AS+803	Start live changes testing
bccLadLiveConfirm 	AS+804	Confirm live changes
bccLadLiveCancel 	AS+805	Cancel live changes
bccLadLiveWd 	806	Live changes watchdog
	807-809	(reserved)
bccLadMonStart 	AS + 810	Start ladder monitor
bccLadMonRestart 	AS + 811	Restart ladder monitor
bccLadMonStop 	AS + 812	Stop ladder monitor
bccLadMonWd 	813	Ladder monitor watchdog
bccLadMonStatus 	AS + 814	Ladder monitor status
	815-899	(reserved)
bccFbReadLocalEntry 	AS + 900	Read entry (field bus, Local)
bccFbWriteLocalEntry 	AS + 901	Write entry (field bus, Local)
bccFbReadLocalNmt 	AS + 902	Read NMT status (field bus, Local)
bccFbWriteLocalNmt 	AS + 903	Write NMT command (field bus, Local)
bccFbWriteLocalEntryE 	AS + 904	Write extended entry (field bus, Local)
	905-909	(reserved field bus)
bccFbReadCoeEntry 	AS + 910	Read entry (field bus, EtherCAT/COE)
bccFbWriteCoeEntry 	AS + 911	Write entry (field bus, EtherCAT/COE)

Symbolic	Code	Description
bccFbReadEcatNmt ^[97]	AS + 912	Read NMT status (field bus, EtherCAT)
bccFbWriteEcatNmt ^[86]	AS + 913	Force NMT status (field bus, EtherCAT)
bccFbWriteCoeEntryE ^[109]	AS + 914	Write extended entry (field bus, EtherCAT/COE)
	915-919	(reserved field bus)
bccFbReadCanEntry ^[87]	AS + 920	Read entry (field bus, CANopen)
bccFbWriteCanEntry ^[99]	AS + 921	Write entry (field bus, CANopen)
bccFbReadCanNmt ^[96]	AS + 922	Read NMT status (field bus, CANopen)
bccFbWriteCanNmt ^[115]	AS + 923	Write NMT command (field bus, CANopen)
bccFbWriteCanEntryE ^[106]	AS + 924	Write extended entry (field bus, CANopen)
	925-949	(reserved field bus)
bccFbReadIF ^[95]	AS + 950	Read interface information
bccFbWriteIF ^[114]	AS + 951	Write interface information
bccFbResolveProcObj	AS + 952	Resolve /proc object.
	961-999	(reserved field bus)
bccRpeAxesGroupResolve ^[244]	AS + 1000	Resolve an axes group
bccRpeAxesGroupInfo ^[235]	AS + 1001	Get information for an axes group
bccRpeAxesGroupList ^[239]	AS + 1002	List available axes groups
bccRpeAxesGroupPositions ^[237]	AS + 1003	Query positions for an axes group
	1003-1099	(reserved for RPE)
bccRpeGASessionBegin ^[232]	AS + 1100	Begin a group authority session
bccRpeGASessionEnd ^[234]	AS + 1101	End a group authority session
bccRpeGASessionWd ^[248]	1102	Send a group authority session watchdog
bccRpeGASessionObjLoad ^[241]	AS + 1103	Load an object to a group authority session

Symbolic	Code	Description
bccRpeGASessionObjSave ^[245]	AS + 1104	Save an object from a Group Authority session
bccRpeGASessionObjStatus ^[242]	AS + 1105	Query status for a object in a group authority session
bccRpeGASessionObjCmd ^[233]	AS + 1106	Command for an object in a group authority session
bccRpeGASessionJogCmd ^[238]	AS + 1107	JOG command for an object in a Group Authority session
bccRpeGASessionUpdateObjPositionP ^[247]	AS + 1108	Update positions for a object's point in a Group Authority session
bccRpeGASessionUpdateObjStepInlinePointP ^[246]	AS + 1109	Update positions for an object's step inline point in a Group Authority session
	1011-1499	(reserved for RPE)
	1500-1999	(reserved for XPL)
	2000-2099	(reserved for Recipe Manager)
	2100-7999	(reserved)
	8000-19999	(available for user messages)
	20000-32767	(reserved)

NACK error codes

General use and standard error codes, related to message [bccNack](#)^[164].

Some NACK code, in specific data, can have some extra informations/data or different meaning.

Label	Value	Description	Extra (#1)
	0-99	Reserved ROBOX SPA	
nackMissingArgs	100	[Remote] One or more missing arguments	Y
nackIllegalArgs	101	[Remote] One or more illegal arguments	Y
nackWrongCommand	102	[Remote] Command not recognized or not supported	

Label	Value	Description	Extra (#1)
nackInvalidChannel	103	[Remote] Invalid channel	
nackSystemBusy	104	[Remote] System busy in order to handle the command	
nackIllegalMode	105	[Remote] Current mode does not allow the command	
nackMemoryFull	106	[Remote] Memory full in handling the command	
nackResourceBusy	107	[Remote] Required resource is not available (could be in use)	
nackTimeout	108	[Remote] Operation/command execution expired	
nackNotImplemented	109	[Remote] Command recognized but not yet implemented (usually in a under development software / device).	
nackFileExist	110	[Remote] File already exist	
nackWriteError	111	[Remote] Error writing	
nackAborted	112	[Remote] Operation/command execution aborted	
nackFileNotExist	113	[Remote] File does not exist	
nackIllegalLength	114	[Remote] --obsolete -- yse nackIllegalArgs	
nackIllegalDevice	115	[Remote] Illegal device	
nackUnformattedDevice	116	[Remote] Device is not correctly formatted	
nackSameFile	117	[Remote] File is the same	
nackOpenError	118	[Remote] Open error	

Label	Value	Description	Extra (#1)
nackCoffMissingRelTab	119	[Remote] Missing relocation table in COFF	
nackCoffNo386Code	120	[Remote] COFF is not for Intel 386 platform	
nackCoffUnresolvedSym	121	[Remote] COFF contains unresolved symbols	
nackBadTaskInit	122	[Remote] Error in task initialization	
nackReadOnly	123	[Remote] Read only	
nackWriteOnly	124	[Remote] Write only	
nackCloseError	125	[Remote] Close error	
nackBackupError	126	[Remote] Backup error	
nackReadError	127	[Remote] Error reading	
nackTooMany	128	[Remote] Too many error	
nackTooFew	129	[Remote] Too few error	
nackOutOfResource	130	[Remote] Out of resource	
nackDataOverflow	131	[Remote] Data overflow	
nackDataUnderflow	132	[Remote] Data underflow	
nackNotFound	133	[Remote] Not found (subject)	
nackSizeMismatch	134	[Remote] Size mismatch	
nackOffsetMismatch	135	[Remote] Offset mismatch	
nackTxError	136	[Remote] General transmission error	
nackRxError	137	[Remote] General reception error	
nackBadBif16	138	[Remote] Bad/malformed BIF16	

Label	Value	Description	Extra (#1)
nackTooNested	139	[Remote] Too many nested	
nackNotProgrammed	140	[Remote] Not programmed	
nackObsolete	141	[Remote] Obsolete command	
nackSameData	142	[Remote] Same data	
nackDifferentData	143	[Remote] Different data	
nackCantForce	144	[Remote] Cant force	
nackCantRelease	145	[Remote] Can release	
nackNotAuthorized	146	[Remote] Not authorized	
nackExists	147	[Remote] Already exist	
nackIllegalDebug	148	[Remote] Illegal debug (session)	
nackNotActive	149	[Remote] Not active	
nackAlreadyActive	150	[Remote] Already active	
nackGeneralError	151	[Remote] General error	Y
nackUserRequest	152	[Remote] Request by user	Y
nackInternalError	153	[Remote] Internal error	Y
nackRequestError	154	[Remote] Request error	Y
nackSeqMismatch	155	[Remote] Sequence mismatch	
nackDeviceError	156	[Remote] Device (generic) error	
nackUnreachDestination	157	[Remote] Unreachable destination	
nackUnreachDestinationCH	158	[Remote] Unreachable destination channel	
nackDeviceFull	159	[Remote] Device full	

Label	Value	Description	Extra (#1)
nackIllegalFile	160	[Remote] Illegal file	
nackDisconnected	161	[Remote] Disconnected	
nackConnected	162	[Remote] Connected	
nackEmpty	163	[Remote] Empty	
nackNotEmpty	164	[Remote] Not empty	
nackFolderExist	165	[Remote] Folder already exist	
nackFolderNotExist	166	[Remote] Folder does not exist	
nackIllegalFolder	167	[Remote] Illegal folder	
nackDeviceNotFound	168	[Remote] Device not found	
nackDeviceNotReady	169	[Remote] Device not ready	
nackIsAFolder	170	[Remote] Is a folder	
nackIsNotAFolder	171	[Remote] Is not a folder	
nackBitAccessDenied	172	[Remote] Bit access denied	
nackBitAccessNeeded	173	[Remote] Bit access needed	
nackInterfaceNotFound	174	[Remote] Interface not found	
nackInterfaceNotReady	175	[Remote] Interface not ready	
nackKeyMismatch	176	[Remote] Key mismatch	
nackDataMismatch	177	[Remote] Data mismatch	
nackDeleteError	178	[Remote] Delete error	
nackIllegalVarsetId	179	[Remote] Illegal variable set ID	
nackRpeNotInstalled	180	[Remote] RPE firmware is not installed	
nackBusy	181	[Remote] Busy	
nackNotBusy	182	[Remote] Not busy	

Label	Value	Description	Extra (#1)
nackExpired	183	[Remote] Expired	
nackIllegalDataType	184	[Remote] Illegal data type	
nackNotInitialized	185	[Remote] Not initialized	
nackAlreadyInitialized	186	[Remote] Already initialized	
nackIllegalContext	187	[Remote] Illegal context	
nackXplNotInstalled	188	[Remote] XPL firmware is not installed	
	189-1999	Reserved Robox SpA (remote usage)	
	2000-2999	Reserved Robox SpA (remote firmware usage)	
nackLocalFileNotExist	3000	[Local] File does not exist	
nackLocalFileExist	3001	[Local] File exist	
nackLocalFileOpenError	3002	[Local] Error opening the file	
nackLocalFileReadError	3003	[Local] Error reading the file	
nackLocalFileWriteError	3004	[Local] Error writing the file	
nackLocalFileBackupError	3005	[Local] Error in file backup	
nackLocalMemoryFull	3006	[Local] Memory full	
nackLocalFileSeekError	3007	[Local] Error accessing the file	
nackLocalFolderExist	3008	[Local] Folder exist	
nackLocalFolderNotExist	3009	[Local] Folder does not exist	
nackLocalDisconnected	3010	[Local] Disconnected	
nackLocalRxError	3011	[Local] General RX error	
nackLocalTxError	3012	[Local] General TX error	

Label	Value	Description	Extra (#1)
nackLocalTimeout	3013	[Local] Timeout	
nackLocalAborted	3014	[Local] Aborted	
nackLocalIllegalArgs	3015	[Local] Illegal arguments	
nackLocalSeqMismatch	3016	[Local] Sequence mismatch	
nackLocalDataUnderflow	3017	[Local] Data underflow	
nackLocalDataOverflow	3018	[Local] Data overflow	
nackLocalOffsetMismatch	3019	[Local] Offset mismatch	
nackLocalBadBif16	3020	[Local] Bad/malformed BIF16	
nackLocalSizeMismatch	3021	[Local] Size mismatch	
nackLocalClientBusy	3022	[Local] Client busy	
nackLocalUnreachDestination	3023	[Local] Unreachable destination	
	3024-4095	Reserved Robox SpA (local usage)	
	4096-0x7FFF	Free for user applications	
	0x8000-0xFFFF	Reserved Robox SpA	

(#1) Extra data is not mandatory for any message and it should be considered as "may be an extra data".

Standard variables

Standard variable that any device/connection using BCC protocol should provide.

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x0000		(not available - special code)		
0x0001-0x0006		(reserved)		
0x0007	U8	Bit value for 16bit integer parameter	U16 Parameter index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x0008	U8	Bit value for 32bit integer parameter	U16 Parameter index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0009	U8	Bit value for 64bit integer parameter	U16 Parameter index U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x000A	U8	Bit value for 16bit integer axis parameter	U16 Parameter index U16 Axis index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x000B	U8	Bit value for 32bit integer axis parameter	U16 Parameter index U16 Axis index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x000C	U8	Bit value for 64bit integer axis parameter	U16 Parameter index U16 Axis index U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x000D	U8	Bit value for non volatile 64bit integer register	U16 Register index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x000E	U8	Bit value for volatile 64bit integer register	U16 Register index U8 Bit index (0-31)	
0x000F	U8	Bit value for 64bit memory	U32 Memory address U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0010	U8	Bit value for logical 16bit input word	U16 Word index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0011	U8	Bit value for logical 32bit input word	U16 Word index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x0012	U8	Bit value for logical 16bit output word	U16 Word index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0013	U8	Bit value for logical 32bit output word	U16 Word index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0014	U8	Bit value for physical 16bit input word	U16 Word index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0015	U8	Bit value for physical 32bit input word	U16 Word index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0016	U8	Bit value for physical 16bit output word	U16 Word index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0017	U8	Bit value for physical 32bit output word	U16 Word index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0018	U8	Bit value for non volatile 16bit integer register	U16 Register index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0019	U8	Bit value for non volatile 32bit integer register	U16 Register index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x001A	U8	Bit value for volatile 16bit integer register	U16 Register index U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x001B	U8	Bit value for volatile 32bit integer register	U16 Register index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x001C	U8	Bit value for 8bit memory	U32 Memory address U8 Bit index (0-7)	U8 Bit value (0=0 N=1)
0x001D	U8	Bit value for 16bit memory	U32 Memory address U8 Bit index (0-15)	U8 Bit value (0=0 N=1)

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x001E	U8	Bit value for 32bit memory	U32 Memory address U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x001F	U8	Bit value for alarm mask	U16 Alarm mask index U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0020	U8	Timer result (Q)	U32 Memory address	U8 Boolean value
0x0021	U8	Timer enable (EN)	U32 Memory address	U8 Boolean value
0x0022	DBL	Timer elapsed time (ET)	U32 Memory address	DBL Time value [ms]
0x0023	DBL	Timer preset (PT)	U32 Memory address	DBL Time value [ms]
0x0024	DBL	Timer preset 2 (PT2)	U32 Memory address	DBL Time value [ms]
0x0025-0x002F		(reserved)		
0x0030	U8	Counter up result (QU)	U32 Memory address	U8 Boolean value
0x0031	U8	Counter down result (QD)	U32 Memory address	U8 Boolean value
0x0032	U8	Counter enable up (CU)	U32 Memory address	U8 Boolean value
0x0033	U8	Counter enable down (CD)	U32 Memory address	U8 Boolean value
0x0034	I32	Counter current value (CV)	U32 Memory address	I32 Counter value
0x0035	I32	Counter preset value (PV)	U32 Memory address	I32 Counter value
0x0036-0x003F		(reserved)		
0x0040	U8 * R	Logical unsigned 8bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	U8 Value 0 U8 Value 1 ...
0x0041	U16 * R	Logical unsigned 16bit	U16 Logical Id U32 Item Id	U16 Value 0 U16 Value 1 ...

Type (code)	Base size	Description	IdData (parameters)	Raw data
			U8 Repeat counter (R)	
0x0042	U32 * R	Logical unsigned 32bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	U32 Value 0 U32 Value 1 ...
0x0043	U64 * R	Logical unsigned 64bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	U64 Value 0 U64 Value 1 ...
0x0044	I8 * R	Logical signed 8bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	I8 Value 0 I8 Value 1 ...
0x0045	I16 * R	Logical signed 16bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	I16 Value 0 I16 Value 1 ...
0x0046	I32 * R	Logical signed 32bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	I32 Value 0 I32 Value 1 ...
0x0047	I64 * R	Logical signed 64bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	I64 Value 0 I64 Value 1 ...
0x0048	DBL * R	Logical double 64bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	DBL Value 0 DBL Value 1 ...
0x0049	FLT * R	Logical float 32bit	U16 Logical Id U32 Item Id U8 Repeat counter (R)	FLT Value 0 FLT Value 1 ...
0x004A	U8	Bit value for logical unsigned 8bit	U16 Logical Id U32 Item Id U8 Bit index (0-7)	U8 Bit value (0=0 N=1)

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x004B	U8	Bit value for logical unsigned 16bit	U16 Logical Id U32 Item Id U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x004C	U8	Bit value for logical unsigned 32bit	U16 Logical Id U32 Item Id U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x004D	U8	Bit value for logical unsigned 64bit	U16 Logical Id U32 Item Id U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x004E	U8	Bit value for logical signed 8bit	U16 Logical Id U32 Item Id U8 Bit index (0-7)	U8 Bit value (0=0 N=1)
0x004F	U8	Bit value for logical signed 16bit	U16 Logical Id U32 Item Id U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0050	U8	Bit value for logical signed 32bit	U16 Logical Id U32 Item Id U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0051	U8	Bit value for logical signed 64bit	U16 Logical Id U32 Item Id U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x0052	STRZ	Logical string (0 terminated)	U16 Logical Id U32 Item Id U16 Size (\0 included)	U8 x size String value
0x0053-0x0063		(reserved)		
0x0064	U16 * R	Logical 16bit input word	U16 Word index U16 Repeat counter (R)	U16 Input word 0 U16 Input word 1
0x0065	U32 * R	Logical 32bit input word	U16 Word index	U32 Input word 0 U32 Input word 1

Type (code)	Base size	Description	IdData (parameters)	Raw data
			U16 Repeat counter (R)
0x0066	U16 * R	Logical 16bit output word	U16 Word index U16 Repeat counter (R)	U16 Output word 0 U16 Output word 1
0x0067	U32 * R	Logical 32bit output word	U16 Word index U16 Repeat counter (R)	U32 Output word 0 U32 Output word 1
0x0068	U8	Logical input channel	U16 Channel index	U8 Channel state (0=off other=on)
0x0069	U8	Logical output channel	U16 Channel index	U8 Channel state (0=off other=on)
0x006A	U16 * R	Physical 16bit input word	U16 Word index U16 Repeat counter (R)	U16 Input word 0 U16 Input word 1
0x006B	U32 * R	Physical 32bit input word	U16 Word index U16 Repeat counter (R)	U32 Input word 0 U32 Input word 1
0x006C	U16 * R	Physical 16bit output word	U16 Word index U16 Repeat counter (R)	U16 Input word 0 U16 Input word 1
0x006D	U32 * R	Physical 32bit output word	U16 Word index U16 Repeat counter (R)	U32 Output word 0 U32 Output word 1
0x006E	U8	Physical input channel	U16 Channel index	U8 Channel state (0=off other=on)
0x006F	U8	Physical output channel	U16 Channel index	U8 Channel state (0=off other=on)
0x0070-0x00C7		(reserved)		
0x00C8	I16 * R	Non volatile 16bit integer register	U16 Register index U16 Repeat counter (R)	I16 Register value 0 I16 Register value 1

Type (code)	Base size	Description	IdData (parameters)	Raw data
				...
0x00C9	I32 * R	Non volatile 32bit integer register	U16 Register index U16 Repeat counter (R)	I32 Register value 0 I32 Register value 1 ...
0x00CA	DBL * R	Non volatile 64bit real register (double)	U16 Register index U16 Repeat counter (R)	DBL Register value 0 DBL Register value 1 ...
0x00CB	FLT * R	Non volatile 32bit real register (float)	U16 Register index U16 Repeat counter (R)	FLT Register value 0 FLT Register value 1 ...
0x00CC	STRZ	Non volatile string register (0 terminated)	U16 Register index U16 Size (\0 included)	U8 x size String value
0x00CD	I16 * R	Volatile 16bit integer register	U16 Register index U16 Repeat counter (R)	I16 Register value 0 I16 Register value 1 ...
0x00CE	I32 * R	Volatile 32bit integer register	U16 Register index U16 Repeat counter (R)	I32 Register value 0 I32 Register value 1 ...
0x00CF	DBL * R	Volatile 64bit real register (double)	U16 Register index U16 Repeat counter (R)	DBL Register value 0 DBL Register value 1 ...
0x00D0	FLT * R	Volatile 32bit real register (float)	U16 Register index U16 Repeat counter (R)	FLT Register value 0 FLT Register value 1 ...

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x00D1	STRZ	Volatile string register (0 terminated)	U16 Register index U16 Size (\0 included)	U8 x size String value
0x00D2	I16 * R	16bit integer parameter	U16 Parameter index U16 Repeat counter (R)	I16 Register value 0 I16 Register value 1 ...
0x00D3	I32 * R	32bit integer parameter	U16 Parameter index U16 Repeat counter (R)	I32 Register value 0 I32 Register value 1 ...
0x00D4	DBL * R	64bit real parameter (double)	U16 Parameter index U16 Repeat counter (R)	DBL Register value 0 DBL Register value 1 ...
0x00D5	FLT * R	32bit real parameter (float)	U16 Parameter index U16 Repeat counter (R)	FLT Register value 0 FLT Register value 1 ...
0x00D6	I16 * R	16bit integer axis parameter	U16 Parameter index U16 Axis first U16 Axis repeat counter (R)	I16 Param value for axes index+0 I16 Param value for axes index+1 ...
0x00D7	I32 * R	32bit integer axis parameter	U16 Parameter index U16 Axis index U16 Axis repeat counter (R)	I32 Param value for axes index+0 I32 Param value for axes index+1 ...
0x00D8	DBL * R	64bit real axis parameter (double)	U16 Parameter index U16 Axis index U16 Axis repeat counter (R)	DBL Param value for axes index+0 DLB Param value for axes index+1 ...
0x00D9	FLT * R	32bit real axis parameter (float)	U16 Parameter index	FLT Param value for axes index+0

Type (code)	Base size	Description	IdData (parameters)	Raw data
			U16 Axis index U16 Axis repeat counter (R)	FLT Param value for axes index+1 ...
0x00DA-0x00E0		(reserved)		
0x00E1	U32 * R	32bit alarm mask	U16 Alarm index U16 Repeat counter (R)	U32 Register value 0 U32 Register value 1 ...
0x00E2	U32 *	Alarm code (alarms stack)	U16 Alarm index U16 Repeat counter (R)	U32 Alarm code at index + 0 U32 Alarm code at index + 1 ...
0x00E3	STRZ	Alarm text (alarms stack)	U16 Alarm index U16 Size (\0 included)	U8 x size String value
0x00E4	I64 * R	Non volatile 64bit integer register	U16 Register index U16 Repeat counter (R)	I64 Register value 0 I64 Register value 1 ...
0x00E5	I64 * R	Volatile 64bit integer register	U16 Register index U16 Repeat counter (R)	I64 Register value 0 I64 Register value 1 ...
0x00E6	I64 * R	64bit integer parameter	U16 Parameter index U16 Repeat counter (R)	I64 Register value 0 I64 Register value 1 ...
0x00E7	I64 * R	64bit integer axis parameter	U16 Parameter index U16 Axis index U16 Axis repeat counter (R)	I64 Param value for axes index+0 I64 Param value for axes index+1 ...
0x00E8-0x012B		(reserved)		
0x012C	DBL	System time (us, from boot)		DBL Time value

Type (code)	Base size	Description	IdData (parameters)	Raw data
0x012D-0x01F3		(reserved)		
0x01F4	U8 * R	Unsigned memory 8bit	U32 Memory address U8 Repeat counter (R)	U8 Byte 0 U8 Byte 1 ...
0x01F5	U16 * R	Unsigned memory 16bit	U32 Memory address U8 Repeat counter (R)	U16 Word 0 U16 Word 1
0x01F6	U32 * R	Unsigned memory 32bit	U32 Memory address U8 Repeat counter (R)	U32 Dword 0 U32 Dword 1 ...
0x01F7	DBL * R	Memory double (64bit)	U32 Memory address U8 Repeat counter (R)	DBL Double 0 DBL Double 1
0x01F8	FLT * R	Memory float (32bit)	U32 Memory address U8 Repeat counter (R)	FLT Float 0 FLT Float 1 ...
0x01F9	U8 * R	I/O port 8bit	U32 Memory address U8 Repeat counter (R)	U8 I/O port 0 state U8 I/O port 1 state
0x01FA	U16 * R	I/O port 16bit	U32 Memory address U8 Repeat counter (R)	U16 I/O port 0 state U16 I/O port 1 state
0x01FB	I8 * R	Signed memory 8bit	U32 Memory address U8 Repeat counter (R)	I8 Byte 0 I8 Byte 1 ...
0x01FC	I16 * R	Signed memory 16bit	U32 Memory address U8 Repeat counter (R)	I16 Word 0 I16 Word 1
0x01FD	I32 * R	Signed memory 32bit	U32 Memory address	I32 Dword 0 I32 Dword 1

Type (code)	Base size	Description	IdData (parameters)	Raw data
			U8 Repeat counter (R)	...
0x01FE	U64 * R	Unsigned memory 64bit	U32 Memory address U8 Repeat counter (R)	U64 Dword 0 U64 Dword 1 ...
0x01FF	I64 * R	Signed memory 64bit	U32 Memory address U8 Repeat counter (R)	I64 Dword 0 I64 Dword 1 ...
0x0200	U8 * R	Dynamic unsigned 8bit	U8[6] VIDD U8 Repeat counter (R)	U8 Value 0 U8 Value 1 ...
0x0201	U16 * R	Dynamic unsigned 16bit	U8[6] VIDD U8 Repeat counter (R)	U16 Value 0 U16 Value1 ...
0x0202	U32 * R	Dynamic unsigned 32bit	U8[6] VIDD U8 Repeat counter (R)	U32 Value 0 U32 Value 1 ...
0x0203	U64 * R	Dynamic unsigned 64bit	U8[6] VIDD U8 Repeat counter (R)	U64 Value 0 U64 Value 1 ...
0x0204	I8 * R	Dynamic signed 8bit	U8[6] VIDD U8 Repeat counter (R)	I8 Value 0 I8 Value 1 ...
0x0205	I16 * R	Dynamic signed 16bit	U8[6] VIDD U8 Repeat counter (R)	I16 Value 0 I16 Value 1 ...
0x0206	I32 * R	Dynamic signed 32bit	U8[6] VIDD U8 Repeat counter (R)	I32 Value 0 I32 Value 1 ...
0x0207	I64 * R	Dynamic signed 64bit	U8[6] VIDD U8 Repeat counter (R)	I64 Value 0 I64 Value 1 ...
0x0208	DBL * R	Dynamic real (64bit)	U8[6] VIDD	DBL Value 0 DBL Value 1

Type (code)	Base size	Description	IdData (parameters)	Raw data
			U8 Repeat counter (R)	...
0x0209	FLT * R	Dynamic float (32bit)	U8[6] VIDD U8 Repeat counter (R)	FLT Value 0 FLT Value 1 ...
0x020A	STRZ	Dynamic string	U8[6] VIDD U16 Size (\0 included)	U8[size] String value
0x020B-0x020F		(reserved)		
0x0210	U8	Bit for dynamic unsigned 8bit	U8[6] VIDD U8 Bit index (0-7)	U8 Bit value (0=0 N=1)
0x0211	U8	Bit for dynamic unsigned 16bit	U8[6] VIDD U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0212	U8	Bit for dynamic unsigned 32bit	U8[6] VIDD U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0213	U8	Bit for dynamic unsigned 64bit	U8[6] VIDD U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x0214	U8	Bit for dynamic signed 8bit	U8[6] VIDD U8 Bit index (0-7)	U8 Bit value (0=0 N=1)
0x0215	U8	Bit for dynamic signed 16bit	U8[6] VIDD U8 Bit index (0-15)	U8 Bit value (0=0 N=1)
0x0216	U8	Bit for dynamic signed 32bit	U8[6] VIDD U8 Bit index (0-31)	U8 Bit value (0=0 N=1)
0x0217	U8	Bit for dynamic signed 64bit	U8[6] VIDD U8 Bit index (0-63)	U8 Bit value (0=0 N=1)
0x0218-0x022F		(reserved)		
0x0230-0xFFFFE		(reserved)		
0xFFFFF		(not available - special code)		

Notes:

- Variables in range from 0x0200 to 0x02xx are to be considered still as PRELIMINARY.

- Contents of the VIDD field is obtained by registering the variable with the [bccRegisterVar](#)²⁶⁶ message and depends on the specific implementation of the connected device.
- The maximum R value for any variable type is dynamically calculated, in order to fit and not overflow the standard message data area size (that is 255 byte in protocol V 3,xx).
- On requesting a not supported variable, the device will reply with a `ackIllegalArgs` error.

Index

- A -

Add a breakpoint 40
Alarm handling 23

- B -

bccAbort 164
bccAborted 164
bccAck 164
bccAlarmHCmd 23
bccAlarmHInfo 30
bccAlarmHList 24
bccAlarmHListE 26
bccAlarmSCmd 24
bccAlarmSGet 34
bccAlarmSInfo 31
bccAlarmSList 31
bccAsciiCmd 73
bccAutoConfig 82
bccBreakpAdd 40
bccBreakpDel 49
bccBreakpInfo 60
bccBreakpList 54
bccBreakpStatus 69
bccBusy 164
bccCanC402Info 118
bccCanEmcyInfo 119
bccCanEmcyRead 124
bccCanNmtRead 130
bccCanNmtWrite 136
bccCanObjRead 125
bccCanObjWrite 132
bccCanPdoRead 129
bccCanRbxChDiag 120
bccCanRbxWsDiag 121
bccCheck 164
bccCMosReset 81
bccCoeObjRead 126
bccCoeObjWrite 133
bccCompleted 164
bccData 164
bccDebugCmd 214
bccDebugStart 70
bccDebugStop 71
bccDebugWd 49
bccEcatNmtRead 128
bccEcatNmtWrite 135
bccEndData 164
bccEnumVar 249
bccFbReadCanEntry 87
bccFbReadCanNmt 96
bccFbReadCoeEntry 90
bccFbReadEcatNmt 97
bccFbReadIF 95
bccFbReadLocalEntry 92
bccFbReadLocalNmt 98
bccFbWriteCanEntry 99
bccFbWriteCanEntryE 106
bccFbWriteCanNmt 115
bccFbWriteCoeEntry 102
bccFbWriteCoeEntryE 109
bccFbWriteEcatNmt 86
bccFbWriteIF 114
bccFbWriteLocalEntry 104
bccFbWriteLocalEntryE 112
bccFbWriteLocalNmt 116
bccFlashDir 148
bccFlashFileDelete 140
bccFlashFileInfo 150
bccFlashFileLoad 142
bccFlashFileRename 159
bccFlashFileSave 159
bccFlashFolderCreate 137
bccFlashFolderDelete 140
bccFlashFolderInfo 156
bccFlashFormat 141
bccFlashInfo 153
bccFlashInfoByFolder 155
bccFlashInit 138
bccFlashList 158
bccFlashSetAttributes 162
bccFlashTree 158
bccForceIC 168
bccForceIW16 168
bccForceOC 169
bccForceOW16 170
bccForceVar 262
bccGetAlarm 26
bccGetDateTime 38
bccGetIC 171
bccGetIW16 171

bccGetMode	74	bccNoData	164
bccGetOC	172	bccObjBlockList	55
bccGetOW16	173	bccOOWSessionBegin	72
bccGetR16	216	bccOOWSessionEnd	73
bccGetR32	217	bccOOWSessionQueryInfo	74
bccGetRR	219	bccOsAttachedFList	57
bccGetRRF	218	bccOscCreate	207
bccGetSR	220	bccOscDestroy	208
bccHardwareReset	81	bccOscStart	209
bccIBlock	164	bccOscStatInfo	213
bccLadLiveCancel	179	bccOscStatus	211, 212
bccLadLiveConfirm	180	bccOscStop	210
bccLadLiveLoad	181	bccOscWd	213
bccLadLiveTest	187	bccPing	215
bccLadLiveWd	189	bccPong	215
bccLadMonRestart	183	bccProcessCmd	50
bccLadMonStart	184	bccProcessDbgCmd	51
bccLadMonStatus	182	bccProcessFlashInfo	62
bccLadMonStop	188	bccProcessGetDebugContext	59
bccLadMonWd	188	bccProcessGetTrace	69
bccLadTaskLoad	180	bccProcessInfo	63
bccLadTaskSave	184	bccProcessInspect	52
bccMonCreate	189	bccProcessList	54
bccMonDestroy	191	bccProcessStatus	67
bccMonList	192	bccReadVar	264
bccMonQuick	193	bccReady	164
bccMonStart	194	bccRegisterVar	266
bccMonStatInfo	193	bccReleaseAllIC	174
bccMonStatus	191	bccReleaseAllIOC	174
bccMonStop	195	bccReleaseAllVars	268
bccMonWd	196	bccReleaseIC	175
bccMonWrite	196	bccReleaseIW16	175
bccNack	164	bccReleaseOC	176
bccNetClientKasSessionBegin	205	bccReleaseOW16	176
bccNetClientKasSessionEnd	206	bccReleaseVar	267
bccNetClientKasSessionInfo	201	bccReportCmd	225
bccNetClientKill	200	bccReportInfo	230
bccNetClientList	202	bccReportList	227
bccNetInfo	203	bccResetAlarm	37
bccNetLogin	200	bccResolveProcObject	82
bccNetLogut	200	bccRpeAxesGroupInfo	235
bccNetStats	204	bccRpeAxesGroupList	239
bccNetUserChange	198	bccRpeAxesGroupPositions	237
bccNetUserCreate	199	bccRpeAxesGroupResolve	244
bccNetUserDelete	199	bccRpeGASessionBegin	232
bccNetUserList	202	bccRpeGASessionEnd	234

- bccRpeGASessionJogCmd 238
 - bccRpeGASessionObjCmd 233
 - bccRpeGASessionObjLoad 241
 - bccRpeGASessionObjSave 245
 - bccRpeGASessionObjStatus 242
 - bccRpeGASessionUpdateObjPointP 247
 - bccRpeGASessionUpdateObjStepInlinePointP 246
 - bccRpeGASessionWd 248
 - bccSafeForceVar 263
 - bccSafeReadVar 265
 - bccSafeReleaseAllVars 268
 - bccSafeReleaseVar 267
 - bccSafeWriteVar 271
 - bccSetAlarm 37
 - bccSetDateTime 38
 - bccSetMode 83
 - bccSetOC 177
 - bccSetOW16 178
 - bccSetR16 221
 - bccSetR32 222
 - bccSetRR 223
 - bccSetRRF 222
 - bccSetSR 224
 - bccSoftwareReset 81
 - bccSysInfo 75
 - bccSysReportCmd 226
 - bccSysReportInfo 231
 - bccSysReportList 228
 - bccUnregisterAllVars 270
 - bccUnregisterVar 269
 - bccWait 164
 - bccWriteVar 270
 - Begin a group authority session 232
 - Begin an OOW session 72
- C -**
- Cancel live changes 179
 - Change a network user 198
 - Command for alarm history 23
 - Command for alarm stack 24
 - Command for an object in a group authority session 233
 - Command for report 225
 - Command for system report 226
 - Confirm live changes 180
 - Create a flash folder 137
 - Create a monitor 189
 - Create a new network user 199
 - Create an oscilloscope 207
 - Create and initialize flashes 138
- D -**
- Data format for process contents inspection 42
 - Data format for process debug context 47
 - Data load sequence 18
 - Data save sequence 16
 - Date/Time handling 38
 - Debug and process handling 39
 - Debug command 214
 - Debug session watch-dog 49
 - Delete a breakpoint 49
 - Delete a flash folder 140
 - Delete a network user 199
 - Delete files from a flash folder 140
 - Destroy a monitor 191
 - Destroy a oscilloscope 208
 - Device handling 71
 - Download transfer sequence 14
- E -**
- End a group authority session 234
 - End an OOW session 73
 - Enumerate variables 249
 - Ethernet network example 272
 - Execute a generic ASCII command 73
 - Execute a process command 50
 - Execute a process debug command 51
- F -**
- Field bus device handling 84
 - Field bus handling 117
 - Field bus supported interface type IDs 85
 - Field bus entry data types 85
 - Flash handling 136
 - Force a variable 262
 - Force a variable (safe) 263
 - Force input channel 168
 - Force input word 16bit 168
 - Force NMT status to EtherCAT Interface 86
 - Force output channel 169
 - Force output word 16bit 170
 - Format a flash 141

- G -

General handling 164
 General messages#bccAbort 165
 General messages#bccAborted 165
 General messages#bccAck 164
 General messages#bccBusy 166
 General messages#bccCheck 166
 General messages#bccCompleted 167
 General messages#bccData 165
 General messages#bccEndData 166
 General messages#bccIBlock 166
 General messages#bccNack 164
 General messages#bccNoData 166
 General messages#bccWait 167
 Get 16bit integer register 216
 Get 32bit integer register 217
 Get alarm history 24
 Get alarm stack 26
 Get current date and time 38
 Get enhanced alarm history 26
 Get float register 218
 Get information for an axes group 235
 Get input channel 171
 Get input word 16bit 171
 Get output channel 172
 Get output word 16bit 173
 Get positions for an axes group 237
 Get real register 219
 Get report contents 227
 Get string register 220
 Get system report contents 228

- I -

I/O handling 167
 Inspect contents of a process 52

- J -

Jog command for an object in a group authority session 238

- K -

Kill a network client 200

- L -

Ladder diagram handling 179
 List available axes groups 239
 List available remote process 54
 List defined breakpoints 54

List object blocks 55
 List OS attached functions 57
 Load a file to a flash folder 142
 Load a ladder task to memory 180
 Load an object to a group authority session 241
 Load live changes 181

- M -

Manage flash volumes 146
 Messages map 273
 Modem network example 273
 Monitor handling 189
 Monitor specifications 11
 Monitor specifications#using 11
 Monitor specifications#usingMultiple 12

- N -

NACK error codes 283
 Network handling 197
 Network interfaces 272
 Network login 200
 Network logout 200

- O -

Oscilloscope handling 206
 Oscilloscope specifications 13
 Oscilloscope specifications#using 13
 Oscilloscope specifications#usingMultiple 14

- P -

Ping answer 215
 Ping command 215
 Protocol conventions 8
 Protocol handling 214
 Protocol specification#dlc 11
 Protocol specification#dst 10
 Protocol specification#header 10
 Protocol specification#len 11
 Protocol specification#msg 11
 Protocol specification#pid 11
 Protocol specification#src 10
 Protocol specifications 9

- Q -

Query a keep alive session information of a network client 201
 Query a monitor status 191
 Query alarm history 30
 Query alarm stack information 31

- Query all alarm stack entries 31
 - Query an oscilloscope status 211
 - Query CANopen C402 information 118
 - Query CANopen EMCY message information 119
 - Query contents from a flash folder 148
 - Query current mode 74
 - Query debug context for a process 59
 - Query info for an OOW session 74
 - Query information for a breakpoint 60
 - Query information of a file in a flash folder 150
 - Query information of a flash 153
 - Query information of a flash by folder 155
 - Query information of a flash folder 156
 - Query ladder monitor status 182
 - Query list of flashes 158
 - Query list of monitors 192
 - Query list of network clients 202
 - Query list of network users 202
 - Query list of oscilloscopes 212
 - Query monitor statistics 193
 - Query network information 203
 - Query network statistics 204
 - Query oscilloscopes statistics 213
 - Query process flash information 62
 - Query process information 63
 - Query report information 230
 - Query Robox CANopen channel diagnostic 120
 - Query Robox CANopen workstation diagnostic 121
 - Query runtime status for process 67
 - Query single alarm stack entry 34
 - Query status for a breakpoint 69
 - Query status for a object in a group authority session 242
 - Query system information 75
 - Query system report information 231
 - Query trace information for process 69
 - Query tree of flash folders 158
 - Quick monitor 193
- R -**
- Read a CANopen EMCY message 124
 - Read a CANopen object 125
 - Read a COE object 126
 - Read a variable 264
 - Read a variable (safe) 265
 - Read an entry from CANopen Interface 87
 - Read an entry from EtherCAT (CoE) Interface 90
 - Read an entry from Local Interface 92
 - Read an EtherCAT NMT 128
 - Read data from Tx/Rx CANopen PDO 129
 - Read interface information 95
 - Read NMT status from CANopen Interface 96
 - Read NMT status from EtherCAT Interface 97
 - Read NMT status from Local Interface 98
 - Read one or more CANopen NMT 130
 - Register a dynamic variable 266
 - Register handling 216
 - Release a variable 267
 - Release a variable (safe) 267
 - Release all input channel 174
 - Release all output channel 174
 - Release all variables 268
 - Release all variables (safe) 268
 - Release input channel 175
 - Release input word 16bit 175
 - Release output channel 176
 - Release output word 16bit 176
 - Rename a file in a flash folder 159
 - Report handling 225
 - Request a CMOS ram reset 81
 - Request a hardware reset 81
 - Request device auto configuration 82
 - Reset alarm stack 37
 - Resolve a /proc object 82
 - Resolve an axes group 244
 - Restart ladder monitor 183
 - RPE handling 231
- S -**
- Save a file from a flash folder 159
 - Save a ladder task from memory 184
 - Save an object from a group authority session 245
 - Set 16bit integer register 221
 - Set 32bit integer register 222
 - Set attributes in a flash folder 162
 - Set current date and time 38
 - Set current mode 83
 - Set float register 222
 - Set output channel 177
 - Set output word 16bit 178
 - Set real register 223
 - Set string register 224

Set user alarm	37	Write interface information	114
Standard variables	289	Write NMT command to CANopen Interface	115
Start a debug session	70	Write NMT command to Local Interface	116
Start a keep alive session for a network client	205	Write one or more CANopen NMT	136
Start a monitor	194		
Start an oscilloscope	209		
Start ladder monitor	184		
Start live changes testing	187		
Stop a debug session	71		
Stop a keep alive session for a network client	206		
Stop a monitor	195		
Stop an oscilloscope	210		
Stop ladder monitor	188		

- T -

Transfer specifications	14
-------------------------	----

- U -

Un-register a dynamic variable	269
Un-register all dynamic variables	270
Update positions for an object's point in a group authority session	247
Update positions for an object's step inline point in a group authority session	246
Upload transfer sequence	15

- V -

Variable handling	249
-------------------	-----

- W -

Watchdog for a monitor	196
Watchdog for an oscilloscope	213
Watchdog for group authority session	248
Watchdog for ladder monitor	188
Watchdog for live changes	189
Write a CANopen object	132
Write a COE object	133
Write a monitor	196
Write a variable	270
Write a variable (safe)	271
Write an entry to CANopen Interface	99
Write an entry to EtherCAT (CoE) Interface	102
Write an entry to Local interface	104
Write an EtherCAT NMT	135
Write an extended entry to CANopen Interface	106
Write an extended entry to EtherCAT (CoE) Interface	109
Write an extended entry to Local interface	112