

C-BUS MODULE INTERFACE SPECIFICATION

Document Number: CBUS-CBMIS

Issue: 3.2

Date: 14 February 2005

Applicability: C-Bus Module version 3

Authorised By: J. A. Gerard

CIS Engineering Manager

Comments on this document should be addressed to:

**Engineering Manager
Clipsal Integrated Systems Pty Ltd
PO Box 103 Hindmarsh
South Australia 5007**

Commercial In Confidence

The following document is issued commercial in confidence and cannot be reproduced or transmitted to unauthorised personnel without the expressed written permission of Clipsal Integrated Systems Pty Ltd.

C-Bus Module Interface Specification**CHANGE HISTORY**

Date	Doc Issue	Comments
20 Aug 02	1	Added Master Security functions / events. Removed the Mode parameter from lighting commands.
20 Aug 02	2	Changed zone number definition.
27 Sep 02	3	Added Scope section. Corrected description of event handling. Added details of network path handling. Added Get Last Source Network Index Function. Added example of making the system non-volatile. Added details of the cbus_time_delay function. Added details of ramp rates. Added CAL Response Function. Added PCI Voltage measurement command and event. Changed formatting.
30 Sep 02	4	Changed cbus_vf_initialise to cbus_bf_initialise. Added details of C-Bus Module memory usage. Added details of how to handle serial transmit characters.
28 Jan 03	5	Corrected names for C-Bus voltage measurement. Added new reason codes for telephony application.
4 Feb 2003	6	Added more information about using CAL commands to program a C-Bus Device. CAL Programming Event Handler has been replaced by CAL Generic Write handler and CAL Generic Read handler. CAL Write Event handlers changed to return a cbus_boolean result. CAL Read Event handlers changed to return a cbus_boolean result and also to pass the read data back via a parameter. Added more details about C-Bus Module customisation. Changed Security commands and event handlers to support new specification for: Tamper, Panic, Low Battery, Arm System. Added new commands and event handlers: Mains Failure, Arm Read/Not Ready, Current Alarm Type.
12 June 03	7	Added new Security messages. Replaced some enumerated types with cbus_boolean. Combined the command, event handler and set/get status functions for Alarm, Panic, Tamper, Low Battery, Mains Failure, Line Cut, Arm Failed, Fire Alarm, Gas Alarm, Other Alarm. Removed Current Alarm Type message. Changed enumerated type naming to avoid problems with name re-use. Added some example code.
3 Sep 03	8	Added cbus_lighting_vf_clear_groups function. Added cbus_lighting_vf_initiate_MMI function. Corrected cbus_lighting_vf_register_event_handler definition.

C-Bus Module Interface Specification

Date	Doc Issue	Comments
18 Dec 03	9	The following definitions have been changed due to name space conflicts with C++ compilers : boolean → cbus_boolean true → cbus_true false → cbus_false CBUS_TIME → CBUS_CBUS_TIME Added Network Discovery Functions. Added delay call-back function. Added Last Source Unit Address function
23 Feb 04	10	Removed cbus_security_vf_status_1_request and cbus_security_vf_status_2_request and replaced with cbus_security_vf_status_request. Fixed cbus_lighting_if_app_index function call name.
6 Apr 04	11	Added error reporting functions and codes
16 Apr 04	12	Changed C-Bus transmission failure event handler
19 Apr 04	13	Recovered lost document sections. Added to discovery process.
14 May 04	14	Adding/changing PCI initialisation functions Removed reference to redundant delay functions.
29 June 04	15	Corrected error reporting and added transmission failure/success reporting.
9 July 04	16	Changed types (eg. byte to int8u)
3 Dec 04	3.0	Re-written for C-Bus Module version 3. Added indicator kill and labelling.
20 Jan 05	3.1	Added flavours for lighting labels
14 Feb 05	3.2	Update for release 3.1. Add queue management. Corrects defects: 4123 - Level MMI reports wrong levels. 4260 - some things declared as "byte", and numerous compiler warnings on some platforms. 4333 - Database change event handler should not report target of a ramp. 4390 - Timeout needs to be increased (in heavy traffic conditions for small embedded systems). 4406 - Generic event handler only called sporadically.

C-Bus Module Interface Specification

CONTENTS

1	Purpose	11
2	Scope.....	11
3	Introduction	11
4	References.....	12
5	Principles of Operation.....	13
5.1	C-Bus Communication Mechanism	13
5.2	Processing of Received Messages	13
5.3	Models and Updates	13
6	Event Handlers	14
6.1	Data Scope / Validity in Event Handlers	15
6.2	Event Handler Duration	15
7	Networks.....	16
8	Retries	16
9	QuickStart Guide.....	17
9.1.1	Customisation	17
9.1.2	Initialisation - suitable for Lighting Message Processing	17
9.1.3	Sending Commands.....	18
9.1.4	Acting on Commands	18
9.1.5	Where next.....	18
10	General Functions.....	19
10.1	Maintenance Functions	19
10.1.1	C-Bus Module Initialisation	19
10.1.2	Update	19
10.1.3	Get C-Bus Module Update / Connect State	20
10.1.4	Register a C-Bus Connect Handler.....	20
10.2	Error Reporting And Handling	22
10.2.1	Get The Last Error Code	22
10.2.2	Set The Last Error Code.....	23
10.2.3	Register an Error Handler.....	23
10.2.4	Register a Transmit Successful Handler.....	23
10.2.5	Register a Transmit Failure Handler	23
10.3	Queue Management	25
10.3.1	Process Receive Queue.....	25
10.3.2	Process Transmit Queue.....	25
10.3.3	Enable / Disable Processing of C-Bus Input	25
10.3.4	Register a Handler For Notification of Output Queue Condition...26	
10.4	Generic Send Message Functions	27
10.4.1	General Purpose Send.....	27
10.4.2	SAL Messages	27
10.5	Other Functions	28
10.5.1	Percent to Decimal Conversion	28
10.5.2	Is This A Lighting Application?.....	28
10.5.3	Process SAL Command Events	28
10.6	Event Handling.....	29
10.6.1	Prioritised Event Mode	29

C-Bus Module Interface Specification

10.6.2	Generic Event Handler	29
10.6.3	Continue Processing	29
10.7	C-Bus PC Interface Properties	31
10.7.1	Get PCI Firmware Version.....	31
10.7.2	Get PCI Unit Address	31
10.7.3	Get PCI Serial Number.....	31
10.7.4	Start Measurement of PCI Voltage	31
10.7.5	Get PCI Voltage	31
10.8	Miscellaneous PCI Events.....	32
10.8.1	Configuration Change Event	32
10.8.2	Voltage Measurement Event	32
10.8.3	Power-Up Event	32
10.9	Transparent Mode.....	33
10.9.1	Enter/Exit Transparent Mode.....	33
10.10	Serial I/O.....	34
10.10.1	Register C-Bus Output Function	34
10.10.2	C-Bus Input	34
10.10.3	Second Serial Port Output.....	34
10.10.4	Second Serial Port Input.....	35
10.11	Registering a Network.....	36
10.12	Get Last Source Network Index	37
10.13	Get Last Source Unit Address Function	37
11	Network Discovery	38
11.1	Functions	38
11.1.1	Set Discovery Parameters.....	38
11.1.2	Initiate Network Discovery	39
11.2	Discovery Events	40
11.2.1	Network Discovery Parameter Event.....	40
11.2.2	Network Discovery Complete Event	41
11.2.3	Discovering Levels	41
12	CAL Services	42
12.1	Functions	42
12.1.1	Generic CAL Transmission.....	42
12.1.2	Transmit CAL Reply	42
12.1.3	Read Response	43
12.2	CAL Events.....	44
12.2.1	CAL Response Handler.....	44
12.3	Device Programming Events.....	45
12.3.1	Virtual Addressing Conventions.....	45
12.3.1.1	Generic Programming Event	45
12.3.1.2	Generic Write Event	46
12.3.1.3	Generic Read Event.....	46
12.3.1.4	Write Event	47
12.3.1.5	Read Event	47
13	Lighting Application.....	49
13.1	C-Bus Lighting Ramp Rates.....	50
13.2	Database Functions	51
13.2.1	Application Registration.....	51
13.2.1.1	Register Application Number.....	51
13.2.1.2	Get Application Index	51

C-Bus Module Interface Specification

13.2.2	Group Registration	51
13.2.2.1	Register Group Address	51
13.2.2.2	Get Group Index	51
13.2.2.3	Clear Registered Groups	52
13.2.3	Database Operations	52
13.2.3.1	Set Database Level	52
13.2.3.2	Get Level	52
13.3	C-Bus Lighting Operations	53
13.3.1	Set Level	53
13.4	Timer Operations	54
13.4.1	Set Timer	54
13.4.2	Get Amount Of Time Left On A Timer	54
13.4.3	Stop Timer and Execute Action	54
13.4.4	Stop Timer with No Action	54
13.5	MMI Operations	56
13.5.1	Initiate MMI	56
13.5.2	Initiate Status MMI	56
13.5.3	Initiate Level MMI	56
13.6	Labels	57
13.6.1	Apply Lighting Label	57
13.6.2	Set Preferred Language for Labels	57
13.7	Events	58
13.7.1	Database Change Event	58
13.7.2	Lighting Event	58
13.7.3	Alternate Lighting Event	59
13.7.4	Additional Alternate Lighting Event	59
13.7.5	MMI Event	60
13.7.6	Label Event	61
14	Trigger Control Application	62
14.1	Network Variables	62
14.2	C-Bus Functions	62
14.2.1	Set Trigger Group	62
14.2.2	Indicator Kill	62
14.3	Labels	63
14.3.1	Apply Trigger Label	63
14.3.2	Set Preferred Language for Labels	63
14.4	Database Functions	64
14.4.1	Set Database Level	64
14.4.2	Get Trigger	64
14.5	Timer Functions	65
14.5.1	Set Timer	65
14.5.2	Get Amount Of Time Left On A Timer	65
14.5.3	Stop Timer and Execute Action	65
14.5.4	Stop Timer with No Action	65
14.6	Events	66
14.6.1	Trigger Events	66
14.6.2	Label Events	66
15	Enable Control Application	67
15.1	Network Variables	67
15.2	C-Bus Functions	67

C-Bus Module Interface Specification

15.2.1	Set Enable Group.....	67
15.3	Database Functions	68
15.3.1	Set Database Level	68
15.3.2	Get Enable	68
15.4	Timer Functions	69
15.4.1	Set Timer.....	69
15.4.2	Get Amount Of Time Left On A Timer.....	69
15.4.3	Stop Timer and Execute Action	69
15.4.4	Stop Timer with No Action	69
15.5	Events.....	70
16	Date and Time Application	71
16.1	Network Variables.....	71
16.2	General Purpose Functions.....	71
16.2.1	Days In Month	71
16.2.2	Day Number	71
16.2.3	Day Number to Day Of Week	71
16.2.4	Day of Week.....	71
16.2.5	Copy Time.....	72
16.2.6	Get Time	72
16.2.7	Set Time.....	72
16.3	C-Bus Functions	73
16.3.1	Send Date	73
16.3.2	Send Time.....	73
16.3.3	Send Date and Time	73
16.4	Events.....	73
17	Telephony application	74
17.1	Network Variables.....	74
17.2	Non-Telephone Interface (Slave) Usage	75
17.2.1	Functions	75
17.2.1.1	Isolate Secondary Output.....	75
17.2.1.2	Recall Last Number.....	75
17.2.1.3	Reject Incoming Calls.....	76
17.2.1.4	Divert	76
17.2.1.5	Clear Diversion.....	76
17.2.2	Events.....	77
17.2.2.1	On/Off Hook Event	77
17.2.2.2	Dial-Out Failure Event.....	78
17.2.2.3	Dial-In Failure Event.....	78
17.2.2.4	Ringing Event.....	79
17.2.2.5	Internet Connection Request Event.....	79
17.3	Telephone Interface (Master) Usage	80
17.3.1	Functions	80
17.3.1.1	Line On Hook	80
17.3.1.2	Line Off Hook	80
17.3.1.3	Dial-Out Failure	81
17.3.1.4	Dial-In Failure.....	81
17.3.1.5	Ringing.....	81
17.3.1.6	Recall Number Response	82
17.3.1.7	Request Internet Connection.....	82
17.3.2	Events	83

C-Bus Module Interface Specification

17.3.2.1	Isolate Secondary Outlet Event.....	83
17.3.2.2	Recall Last Number Request Event.....	83
17.3.2.3	Reject Incoming Call Event	84
17.3.2.4	Divert Event	84
17.3.2.5	Clear Diversion Event.....	84
18	Security application	85
18.1	Network Variables.....	85
18.2	Security Database Functions	86
18.2.1	Get Zone State	86
18.2.2	Set Zone State	86
18.2.3	Get Zone Isolation	86
18.2.4	Set Zone Isolation	86
18.2.5	Get System Armed Status	86
18.2.6	Set System Armed Status	87
18.2.7	Get System State	87
18.2.8	Set System Status.....	87
18.3	Non-Security Panel (Slave) Usage.....	88
18.3.1	Functions	88
18.3.1.1	Status Request.....	88
18.3.1.2	Arm System	88
18.3.1.3	Set / Drop Tamper.....	88
18.3.1.4	Raise Alarm	88
18.3.1.5	Emulate Keypad.....	89
18.3.1.6	Display Message.....	89
18.3.1.7	Request Zone Name	89
18.3.2	Events.....	90
18.3.2.1	System Armed Event.....	90
18.3.2.2	Exit Delay Started Event	90
18.3.2.3	Entry Delay Started Event	90
18.3.2.4	Status Event.....	91
18.3.2.5	Zone State Event	91
18.3.2.6	Battery Charging Event	92
18.3.2.7	Zone Name Event	92
18.3.2.8	Password Entry Event.....	92
18.3.2.9	Arm Ready / Not Ready Event	93
18.4	Security Panel (Master) Usage	94
18.4.1	Functions	94
18.4.1.1	System Arm / Disarm	94
18.4.1.2	Exit Delay Started	94
18.4.1.3	Entry Delay Started	94
18.4.1.4	System State.....	94
18.4.1.5	Zone State	94
18.4.1.6	Zone Isolated	95
18.4.1.7	Battery Charging	95
18.4.1.8	Zone Name	95
18.4.1.9	Status Report 1	95
18.4.1.10	Status Report 2	95
18.4.1.11	Password Entry Status	95
18.4.1.12	Arm Ready / Not Ready	96
18.4.2	Events.....	97

C-Bus Module Interface Specification

	18.4.2.1	Status 1 Request Event.....	97
	18.4.2.2	Status 2 Request Event.....	97
	18.4.2.3	Arm System Event	97
	18.4.2.4	Alarm Event	97
	18.4.2.5	Tamper Event	98
	18.4.2.6	Emulate Keyboard Event.....	98
	18.4.2.7	Display Message Event.....	98
	18.4.2.8	Request Zone Name Event	98
19		Measurement Application.....	99
	19.1	Network Variables.....	99
	19.2	Functions	101
	19.2.1	Send Measurement Data.....	101
	19.3	Events.....	101
20		C-Bus String Functions	102
	20.1	Is Hexadecimal Char.....	102
	20.2	Hexadecimal Char to Number	102
	20.3	Hexadecimal Pair to Number	102
	20.4	Hexadecimal String to Number	102
	20.5	Number to Hexadecimal Char	102
	20.6	Append an Integer to a String	102
	20.7	Append a Hex Number to a String.....	103
	20.8	Percent String	103
	20.9	Append Carriage Return	103
	20.10	Convert a String to a Hex String.....	103
21		Usage Notes	104
	21.1	C-Bus Module Customisation.....	104
	21.1.1	C-Bus Enabled Level.....	104
	21.1.2	Application Selection	105
	21.1.3	Tick Rate.....	105
	21.1.4	C-Bus Module Queues	105
	21.1.5	Memory Usage	106
	21.2	Transparent Mode.....	108
	21.2.1	Entering Transparent Mode	108
	21.2.2	Registering Event Handlers.....	108
	21.2.3	Transmit Data to C-Bus.....	109
	21.2.4	Receive Data From C-Bus.....	109
	21.3	Use With Other C-Bus Applications.....	110
	21.4	Programming Configuration Data	110
	21.4.1	PCI Configuration Data.....	110
	21.4.2	C-Bus Device Specific Configuration Data.....	110
	21.4.2.1	Virtual Memory Addresses	110
	21.4.2.2	Writing the Address Pointer.....	110
	21.4.2.3	Writing to the C-Bus Device	111
	21.4.2.4	Reading from the C-Bus Device	111
	21.4.2.5	Using Other Virtual Addresses	111
	21.4.2.6	Other CAL Commands	111
	21.4.2.7	Examples	112
	21.5	Creating a System with Non-Volatile Behaviour	114
22		Example Application.....	115
	22.1	Introduction	115

C-Bus Module Interface Specification

22.2 Code 115

C-Bus Module Interface Specification

1 PURPOSE

This document is to define the Application Program Interfaces to the C-Bus Module.

APIs are sorted by their applicable C-Bus Application, except universal APIs that are listed separately.

A section at the end of the document describes how to use the APIs for particular purposes.

2 SCOPE

This document applies to version 3 of the C-Bus Module.

3 INTRODUCTION

The C-Bus Module provides a portable, scalable interface between embedded systems and C-Bus, and is used by Clipsal Integrated Systems in many released products.

The C-Bus Module complies with the requirements for C-Bus Enabled, at levels 1, 2, 3 or 4.

At the time of writing, the C-Bus Module has been ported to the following processors:

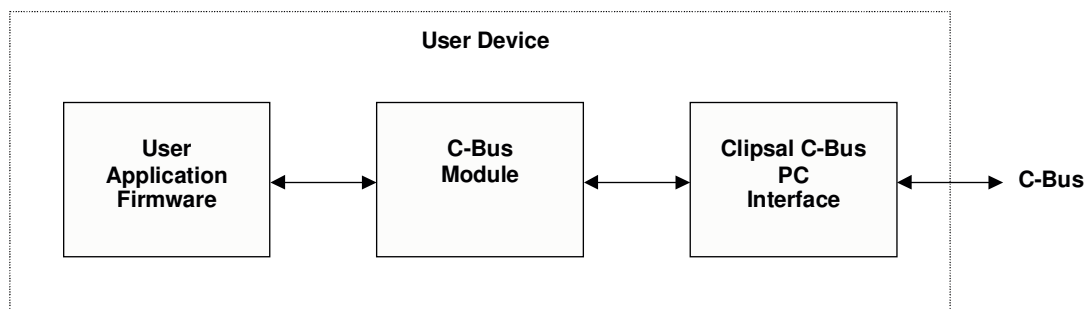
- . Hitachi H8S and H8/300 (with Hitachi HEW)
- . Texas Instruments MSP430 (with IAR EW430 and GCC/MSP430)
- . Intel x86 PC / Windows (with Borland C / C++ Builder)
- . Intel x86 PC / Linux (with GCC 3.3.2)

Porting is aided by careful definition of all data types, and careful adherence to the specified API. All procedures are encapsulated and have a defined C code interface.

The C-Bus Module has been designed for use in embedded systems, communicating with C-Bus via a C-Bus PC Interface as shown below. The C-Bus Module provides a programmer-friendly interface between the user's Application Firmware and the C-Bus PC Interface.

For PC / Windows, a DLL allows the APIs to be called from numerous other programming languages.

For PC / Linux, an object archive allows the code to be statically linked into programs in any language that uses the standard linker.



C-Bus Module Interface Specification

4 REFERENCES

Clipsal Integrated Systems Documents

CBUS-SIUG	C-Bus Serial Interface Users Guide
CBUS-VMA	C-Bus System Virtual Addressing for Compound Devices
CBUS-APP/02	C-Bus Lighting Application
CBUS-APP/05	C-Bus Security Application
CBUS-APP/07	C-Bus Trigger Control Application
CBUS-APP/08	C-Bus Enable Control Application
CBUS-APP/23	C-Bus Clocks and Timekeeping Application
CBUS-APP/24	C-Bus Telephone Status and Control Application
CBUS-APP/28	C-Bus Measurement Application

C-Bus Module Interface Specification

5 PRINCIPLES OF OPERATION

5.1 *C-Bus Communication Mechanism*

The C-Bus Module forms an interface to C-Bus using two queues – a transmit queue and a receive queue. This enables the low-level communication routines to be very short and fast. Time-consuming processing occurs at the higher levels in the software.

To send a C-Bus message, the appropriate function call is used. This generates the appropriate C-Bus message and places it into the transmit queue. When any previous messages have been sent successfully (or the number of retries has been exceeded), then the new message is sent.

As data is received, complete messages are placed onto a receive queue. These are processed in order as time allows.

Direct user software access to the queues is not necessary or encouraged.

5.2 *Processing of Received Messages*

C-Bus messages can arrive at any time. These messages are accumulated and periodically processed by the C-Bus Module. As the messages are decoded, an **event handler** is called.

Each event handler needs to have been registered before the decoded C-Bus messages can be passed to the user application firmware.

If a C-Bus message arrives for which an event handler has not been registered, then that C-Bus message will be discarded.

5.3 *Models and Updates*

When configured for C-Bus Enabled Level 4, the C-Bus Module builds an internal model of the state of each group in its registered Lighting Applications, and tracks all ramp operations. When used for Level 4 interfacing, the C-Bus Module **must** have its update function called at well-defined intervals.

C-Bus Module Interface Specification

6 EVENT HANDLERS

Event Handlers may be registered for a range of C-Bus events as described in the following sections.

Where there is more than one relevant event handler for a particular C-Bus event, there are two alternatives for how the event handlers will be called:

- a. Default Mode – every relevant event handler for a particular event will be executed. They will be executed in order from the most specific to the most general; or
- b. Prioritised Mode – only the most specific of the registered event handlers for a particular event will be executed.

The mode is set using the `cbus_vf_set_prioritised_event_mode` function (see section 10.6.1).

The hierarchy / priority of the event handlers is shown below:

C-Bus Event	Applicable Event Handlers		
	⇐ More General		More Specific ⇒
Any	Generic		
CAL	Generic	CAL Generic Programming	CAL Response CAL Generic Write CAL Generic Read CAL Write CAL Read
Lighting	Generic	Lighting	
Trigger Control	Generic	Trigger Control	
Enable Control	Generic	Enable Control	
Time	Generic	Time	
Telephony	Generic	Telephony	
Security	Generic	Security	

For example, if the CAL Write and CAL Generic Programming Event Handlers are registered and a CAL Write event is received, how the event is handled depends on the mode:

- If the C-Bus Module is in Default Mode, both event handlers will be called.
- If Prioritised Mode is set, only the CAL Write Event Handler will be called.

C-Bus Module Interface Specification

In the above example, if the CAL Read event handler is not registered and a CAL Read event is received, then the CAL Generic Programming Event Handler will be called.

6.1 Data Scope / Validity in Event Handlers

Data that is passed to an event handler is only valid during the life of that event handler.

If a parameter is passed which the user needs to use at a later time (for example a character string), the event handler must copy it elsewhere.

6.2 Event Handler Duration

When an event handler is called, all other processing is suspended.

An event handler should complete and return control as quickly as possible.

C-Bus Module Interface Specification

7 NETWORKS

Section 10.11 provides further details about registering networks.

The C-Bus Module stores registered networks in an array, with a size set in the `cbus_config.h` file by the constant `CBUS_MAX_NETWORK_COUNT + 1`. The last element in the networks table is used when a message is received from an unregistered network.

When a message is received, the source network path is extracted and compared with the registered networks. The index is stored and can be obtained using the function `cbus_if_get_last_source_network_index` (see section 10.12).

If a reply needs to be sent to the source network, this function can be used to retrieve the index of the path to the network.

This approach means that networks only need to be registered once, at initialisation time. Thereafter, all references to networks are by an index into the network table, rather than by direct inclusion of a network route.

8 RETRIES

Almost all function calls to the C-Bus Module allow specification of a number of retries.

Generally, this should be set to 0 (no retries). If a C-Bus network is not reliable and needs a higher number of retries, it usually indicates one of the following problems:

- a. Too many or too few network burdens;
- b. Insufficient voltage at some devices in the network;
- c. Too many devices attached to a single network segment;
- d. Poor wiring connections;
- e. Interference from other electrical devices or wiring - possibly caused by placing C-Bus wiring too close to other wiring.

Use of a number of retries greater than 3 is strongly discouraged.

Any network needing 3 or more retries has other serious defects that should be rectified.

C-Bus Module Interface Specification

9 QUICKSTART GUIDE

There is no excuse! Read this whole document thoroughly and get a good understanding.

Not willing to do that - then this quickstart guide should give enough information about the really important points.

9.1.1 Customisation¹

Take the following steps to customise the C-Bus Module for your use and minimise the memory usage:

- If only the lighting application(s) are used, modify `cbus_config.h` to remove all of the other application definitions.
- Set the maximum number of networks that will be needed (10 by default) in `cbus_config.h`.
- Set the maximum number of lighting applications that will be needed (10 by default) in `cbus_config.h`.
- Set the maximum number of Group Addresses that will be needed (256 by default) per Application in `cbus_config.h`.
- Set the size of the C-Bus Command and Receive Buffers (not the length of the items, just the number of items that are stored in each buffer) in `cbus_config.h`.

9.1.2 Initialisation - suitable for Lighting Message Processing

To initialise the C-Bus Module:

- Register any networks of interest using `cbus_bf_register_network_path`.
- Register any Application Addresses of interest using `cbus_lighting_bf_register_app`.
- If you have less than 256 Group Addresses per Application, then register the Group Addresses of interest using `cbus_lighting_bf_register_group`.
- Register your C-Bus PCI Serial data transmission handler using `cbus_vf_register_serial_transmit_handler`.
- Register a Lighting Event Handler using `cbus_lighting_vf_register_event_handler`.
- If you are interested in being notified that the database has changed (for example, to store the data to non-volatile memory), register an event handler using `cbus_lighting_vf_register_database_event_handler`.

¹ Customisation is not applicable for pre-compiled Windows and Linux versions.

C-Bus Module Interface Specification

- If you want to be notified that there was an MMI discrepancy (which will cause the database to be updated automatically), register an event handler using `cbus_lighting_vf_register_MMI_event_handler`.
- Enable the transmission and reception of data from the PC Interface.
- Call `cbus_bf_initialise`.
- Wait for the state returned by `cbus_ef_get_connect_state` to change to `cbus_ce_module_normal_operation`. Use a timeout. An even better approach is to register a connect event handler, which will raise an event when a successful connection is made, or when it has been determined that a connection cannot be made.
- If any Group Address levels have been stored in non-volatile memory, restore the levels by setting them with `cbus_lighting_vf_set_database_level`.

9.1.3 Sending Commands

To send a lighting command, use `cbus_lighting_vf_set_level`.

The timer functions in section 13.4 can be used to set a Group Address to a level with a timer on it.

9.1.4 Acting on Commands

The registered event handler is called each time a Group Address changes. If the database Change Event Handler or the MMI Event Handler is registered, they will also be called as required.

9.1.5 Where next....

Read the rest of this document!

C-Bus Module Interface Specification

10 GENERAL FUNCTIONS

The following functions are not associated with any particular C-Bus Application.

10.1 Maintenance Functions**10.1.1 C-Bus Module Initialisation**

```
cbus_boolean cbus_bf_initialise(void);
```

This function performs the following operations:

- Initialises the C-Bus Database;
- Initialises the C-Bus Transmit and Receive queues;
- Prepares for a connection to the C-Bus PCI.

This function always returns `cbus_true`.

***This function must always be called before any other C-Bus Module function.
If the C-Bus Module is shut down for any reason, this function must be called before re-starting.***

10.1.2 Update

The update service must be executed on a regular basis (every 5ms to 200ms depending on the build configuration). The update service performs the following operations:

- Starts and initialises the connection to the C-Bus PCI, if required (see section 10.1.1);
- Updates all internal functions, process, databases and models;
- Updates any timers, each second (if a timer expires, the appropriate timeout action is performed);
- Sends messages in the transmit queue to C-Bus; and
- Processes messages from the receive queue.

The transmit and receive actions are performed either on every update call, or every alternate update call depending on the build configuration.

The update function has the interface:

```
cbus_et_module_state cbus_ef_update(void);
```

The function result is the enumerated type `cbus_et_module_state`. The following valid states can be returned:

C-Bus Module Interface Specification

State	Meaning
<code>cbus_ce_module_not_initialised</code>	The C-Bus module has not yet been (or completed) initialisation
<code>cbus_ce_module_pci_reset</code>	The C-Bus Module has reset the PCI as part of the initialisation process
<code>cbus_ce_module_pci_setup</code>	The C-Bus Module is setting up the PCI as part of the initialisation process
<code>cbus_ce_module_normal_operation</code>	The C-Bus Module is operating normally
<code>cbus_ce_module_initilise_fail</code>	The C-Bus Module failed to initialise the C-Bus PCI, normal operation is not possible

Important note: Transmission should not be attempted until the update function (or the function `cbus_ef_get_connect_state`) returns `cbus_ce_module_normal_operation`. If transmission is attempted before this state is returned, information sent using transmit requests could be discarded.

10.1.3 Get C-Bus Module Update / Connect State

```
cbus_et_module_state cbus_ef_get_connect_state(void);
```

This function returns the current C-Bus Module operation / connection state. This state is exactly the same state that is returned by the update function described in section 10.1.1, with the same return values.

If the C-Bus Module is shut down for any reason, the values returned by this function are meaningless.

10.1.4 Register a C-Bus Connect Handler

```
void cbus_vf_register_connect_handler(  
    void (*f)(cbus_et_connect_result));
```

Calling this function registers a connect handler.

The connect handler will be called when the C-Bus Module has made (or failed) a connection to C-Bus.

The registered function shall be of the form:

```
void my_event_handler(cbus_et_connect_result);
```

The function is passed the enumerated type `cbus_et_connect_result`. The following valid results can be passed:

C-Bus Module Interface Specification

Result	Meaning
<code>cbus_ce_connect_failed_no_pci</code>	Connection failed, it appears there is no C-Bus PCI present
<code>cbus_ce_connect_failed_invalid_pci_reply</code>	Connection failed, an unexpected reply was obtained from a C-Bus PCI
<code>cbus_ce_connect_ok</code>	Normal connection was successfully achieved

C-Bus Module Interface Specification

10.2 Error Reporting And Handling

Some functions will set an error code when an invalid operation is requested. The code may be inspected or an error handler may be activated to process these errors.

There is also a mechanism to report transmission failures or success.

10.2.1 Get The Last Error Code

```
int8u cbus_if_get_last_error(void);
```

This function returns the last error code. The error code returned exists until it is overwritten by another error code with a later C-Bus Module function call.

Possible error codes are:

Symbolic Error Name	Code	Description
CBM_SUCCESS	0	No error.
CBM_CMDSTR_TOO_LONG	1	C-Bus command was too long. The command must be less than CBUS_COMMAND_STRING_LENGTH.
CBM_STRING_TOO_LONG	2	The length of the string passed as a parameter was too long.
CBM_TOO_MANY_NETWORKS	3	An attempt to register a network has failed as the space allocated for networks is used up.
CBM_CMD_TOO_LONG	4	An attempt to add a c-bus command to the output queue has failed as the command was too long.
CBM_TOO_MANY_APPLICATIONS	5	An attempt to register another application has failed as there is no more space for applications.
CBM_APPLICATION_NOT_FOUND	6	A C-bus command has failed as the requested application is not registered.
CBM_GRP_EQ_255	7	A group number of 255 is not valid for the requested operation.
CBM_BAD_PCI_VERSION	8	The requested operation requires a higher PCI version.
CBM_GROUP_NOT_FOUND	9	The requested group has not been registered in the database.
CBM_TOO_MANY_GROUPS	10	An attempt to register more groups than the available space allocated.
CBM_INIT_FAILED	11	Connection to the PCI failed.

C-Bus Module Interface Specification

10.2.2 Set The Last Error Code

```
void cbus_vf_set_last_error(int8u error);
```

This sets the C-Bus Module error code, and calls any registered error handler. This can be used to propagate errors back to user code.

10.2.3 Register an Error Handler

```
void cbus_vf_register_error_handler(void (*f) (int8u));
```

This registers an event handler for error reporting. When an error occurs, the registered handler will be called. The parameter passed is the error code.

The registered function shall be of the form:

```
void my_event_handler(int8u number);
```

10.2.4 Register a Transmit Successful Handler

```
void cbus_vf_register_tx_succeed_handler(
    void (*f) (int8u, int8u));
```

This registers a function to be called when transmission succeeds. The registered function shall be of the form:

```
void my_event_handler(int8u Unit_App,
    int8u Msg_Type);
```

10.2.5 Register a Transmit Failure Handler

```
void cbus_vf_register_failure_handler(void (*f) (int8u,
    int8u,
    int8u));
```

In the event of a transmission failure, the handler will be called. The registered function shall be of the form:

```
void my_event_handler(int8u tx_fail_code,
    int8u unit_app,
    int8u msg_type);
```

unit_app is either the unit number or the application number, depending on the message type.

tx_fail_code is one of:

Transmission error Symbolic Name	Value	Description
cbus_ce_tx_no_acknowledge	0	The message transmitted to the Serial Interface with the matching Alpha was not transmitted into the local C-Bus network, due to too many re-transmissions with no acknowledgment.

C-Bus Module Interface Specification

Transmission error Symbolic Name	Value	Description
<code>cbus_ce_tx_checksum_bad</code>	1	The message transmitted to the Serial Interface with the matching Alpha was not transmitted into the local C-Bus network, due to corruption in transmission.
<code>cbus_ce_tx_no_clock</code>	2	The message transmitted to the Serial Interface with the matching Alpha was not transmitted into the local C-Bus network, due to Serial Interface losing C-Bus synchronising clock at any time during an attempted C-Bus transmission.
<code>cbus_ce_tx_buffer_overflow</code>	3	The PCI could not accept the data (eg too much data or checksum error).
<code>cbus_ce_tx_timeout</code>	4	There was no response in the required time.
<code>cbus_ce_tx_application_offline</code>	5	The application has been taken offline because a series of transmissions to it have failed.
<code>cbus_ce_tx_unknown_error</code>	255	Undetermined problem.

`msg_type` is one of:

Message Type Symbolic Name	Value	Description
<code>cbus_ce_mt_bridged</code>	0	Destination was a bridge.
<code>cbus_ce_mt_sal</code>	1	SAL message.
<code>cbus_ce_mt_cal</code>	2	CAL message.
<code>cbus_ce_mt_unknown</code>	255	Unknown message type

C-Bus Module Interface Specification

10.3 Queue Management

In some high traffic conditions, the receive and transmit queues can be manually processed (instead of, or as well as) automatically processed.

10.3.1 Process Receive Queue

```
void cbus_vf_process_receive_queue(void);
```

If anything is waiting in the receive queue (from C-Bus), it will be processed by the appropriate application, including calling back any event handlers if required.

This is not normally needed - automatic queue processing (during the update cycle) is normally sufficient

10.3.2 Process Transmit Queue

```
void cbus_vf_process_transmit_queue(void);
```

If anything is waiting in the transmit queue (to go to C-Bus), it will be transmitted if the C-Bus PCI is available.

This is not normally needed - automatic queue processing (during the update cycle) is normally sufficient

10.3.3 Enable / Disable Processing of C-Bus Input

```
void cbus_vf_stop_ip_processing(cbush_boolean);
```

If the parameter is `cbus_true`, then input received from C-Bus will be discarded.

If the parameter is `cbus_false`, then input received from C-Bus is queued for processing.

Normally, the only reason for disabling input from C-Bus is because the input *and* output queues fill, leading to messages sent to C-Bus being lost.

In this case, it is preferable to ensure that output messages are transmitted, to choke off input messages until the output queues are sufficiently emptied.

C-Bus Module Interface Specification

10.3.4 Register a Handler For Notification of Output Queue Condition

```
cbus_vf_register_tx_queue_flow_ctl_handler(  
    void (*f)(cbus_et_flow_type));
```

When the output queue is either very full, or very empty, the registered event handler will be called. The registered function shall be of the form:

```
void my_event_handler(cbus_et_flow_type e_flow_type);
```

`e_flow_type` is one of:

Flow Control Symbolic Name	Value	Description
<code>cbus_ce_flow_lo</code>	2	The output queue has dropped below the low size threshold.
<code>cbus_ce_flow_hi</code>	3	The output queue has filled above the high size threshold.

C-Bus Module Interface Specification

10.4 Generic Send Message Functions**10.4.1 General Purpose Send**

```
void cbus_vf_send_command(char * s,  
                          int8u retries);
```

This sends a given C-Bus message string through the PC Interface with a given number of retries. The C-Bus Module does no interpretation or other processing of the message.

On success the C-bus error code is `CBM_SUCCESS`. If the message string `s` is too long the C-bus error code is set to `CBM_CMDSTR_TOO_LONG`.

This function should not be required, because functions are provided for sending all defined C-Bus message types.

10.4.2 SAL Messages

```
void cbus_vf_send_SAL_command(int8u network,  
                              int8u application,  
                              char *command,  
                              int8u retries);
```

This builds and sends a C-Bus SAL command. The C-Bus Module incorporates the header, network and application then appends the command.

The C-Bus Module may append this command to the previous command if:

1. The network and application match;
2. The previous command has not been sent;
3. There is still room at the end of the previous message buffer;
4. The previous command was not a Label command; and
5. There is more than one item in the queue OR there is one item in the queue and it has not been sent.

If the command can not be appended, a new message will be created in the output queue.

The `command` parameter is a null terminated string that does not include any C-Bus header (05), Application Address or Network Route. If the `command` parameter is too long the function will fail and set the error code to `CBM_CMD_TOO_LONG`.

This function should not be required, since functions are provided for sending all defined C-Bus message types.

If several commands are concatenated, the number of retries used will be the greater of the retries for each of the commands sent.

C-Bus Module Interface Specification

10.5 Other Functions

The following functions are for general-purpose use.

10.5.1 Percent to Decimal Conversion

C-Bus uses a non-trivial method to convert between decimal level values and percentage levels. These conversion functions are provided as a convenience. These conversions are generally used for the C-Bus Lighting Application.

```
int8u cbus_if_dec_to_percent(int8u n);
```

This converts from a decimal value (0-255) to a percentage (0-100).

```
int8u cbus_if_percent_to_dec(int8u n);
```

This converts from a percentage (0-100) to a decimal value (0-255).

10.5.2 Is This A Lighting Application?

```
cbus_boolean cbus_bf_is_application_lighting(int8u app);
```

Returns whether an application number is a C-Bus Lighting Application (Generally, in the range \$30 - \$5F).

10.5.3 Process SAL Command Events

```
void cbus_vf_process_SAL_commands(cbus_boolean state);
```

When the device is being programmed, or during start-up, it may be desirable to disable the processing of SAL commands and C-Bus MMIs.

Use this function to enable or disable processing of SAL and MMI commands.

C-Bus Module Interface Specification

10.6 Event Handling

Every 5 - 200ms (depending on the configuration), the message(s) at the top of the C-Bus receive queue are processed. The Application number will determine what happens to the message.

For each C-Bus Application, a set of procedures will process the message, and then call an event handler for that Application and message type. Each event handler needs to be registered. If there is no registered event handler for a particular message, then nothing is done. Finally, the generic event handler is called.

The same mechanism is used to respond to any changes made to Network Variables via other function calls. Hence if a function call is made which changes something in the C-Bus model, the event handler will be called.

10.6.1 Prioritised Event Mode

```
void cbus_vf_set_prioritised_event_mode(cbus_boolean prioritised);
```

Use this function to control how events are handled when there is more than one relevant handler.

If `prioritised` is `cbus_true`, then only the most specific of the relevant handlers will be called. If `cbus_false`, then all relevant handlers will be called.

10.6.2 Generic Event Handler

```
void cbus_vf_register_handler(void (*f)(int8u *,
                                         int8u));
```

Use this function to register the generic event handler. A generic event handler will only be called if it has been registered. The event handler is passed the received C-Bus message as an array of bytes, and the length of the message.

The registered function shall be of the form:

```
void my_event_handler(int8u *message,
                      int8u message_length);
```

Use of the generic event handler is strongly discouraged.

Such use requires a detailed knowledge of the C-Bus protocol, which should not be necessary when using the facilities of the C-Bus Module.

10.6.3 Continue Processing

```
void cbus_vf_register_continue_processing_call_back(
    cbus_boolean (*f)(void));
```

Processing of received C-Bus commands may take some time, depending on what the user does in the event handlers.

C-Bus Module Interface Specification

A call-back function can be registered so that at the end of processing each command, the user application is asked whether or not continue with processing more commands. This is done to avoid the situation where processing the whole queue may take more time than is allocated for some task.

If no event handler is registered, processing will continue until the receive queue is empty. The registered function shall be of the form:

```
cbus_boolean my_event_handler(void);
```

The call back function shall return **cbus_true** or **cbus_false**, depending on whether it wants processing of received commands to continue or not.

C-Bus Module Interface Specification

10.7 C-Bus PC Interface Properties

The following functions return details of the PCI being used to make the connection to the C-Bus network.

10.7.1 Get PCI Firmware Version

```
char *cbus_cf_get_PCI_version(void);
```

This returns a pointer to a null terminated string containing the PCI firmware version, as ASCII coded text. The result will be a maximum of 17 characters (including the null terminator).

10.7.2 Get PCI Unit Address

```
int8u cbus_if_get_PCI_unit_address(void);
```

This returns the Unit Address of the C-Bus PCI.

10.7.3 Get PCI Serial Number

```
int32u cbus_if_get_PCI_serial_number(void);
```

This returns the Serial Number of the C-Bus PCI as an unsigned 32 bit integer.

10.7.4 Start Measurement of PCI Voltage

```
void cbus_vf_measure_PCI_voltage(void);
```

This initiates a measurement of the C-Bus voltage at the attached PCI. When the voltage measurement has been returned from the PCI, a Voltage Measurement Event will be raised. The Voltage will then be able to be read using the `cbus_cf_get_PCI_voltage` function.

10.7.5 Get PCI Voltage

```
char *cbus_cf_get_PCI_voltage(void);
```

This returns a pointer to a null terminated string containing the PCI Voltage. The result will be a maximum of 6 characters (including the null terminator).

C-Bus Module Interface Specification

10.8 Miscellaneous PCI Events**10.8.1 Configuration Change Event**

```
void cbus_vf_register_config_change_handler(void (*f)(void));
```

If a configuration parameter (such as unit address etc) of the PCI used to connect to C-Bus is changed (via C-Bus), an event handler can be called. It is up to the event handler to determine what has been changed and act accordingly.

Use this function to register the event handler. If no event handler is registered, this event will be ignored.

The registered function shall be of the form:

```
void my_event_handler(void);
```

10.8.2 Voltage Measurement Event

```
void cbus_vf_register_voltage_measure_handler(void (*f)(void));
```

When a command to measure the C-Bus Voltage is sent (with the function `cbus_vf_measure_PCI_voltage`, the reply from the PCI will trigger calling a Voltage Measure Event Handler. Use this function to register the event handler. If no event handler is registered, this event will be ignored.

The registered function shall be of the form:

```
void my_event_handler(void);
```

10.8.3 Power-Up Event

```
void cbus_vf_register_power_up_handler(void (*f)(void));
```

If the C-Bus power fails and is restored, the C-Bus PCI will be reset. The PCI can notify that power up has occurred, and this in turn can cause an event handler to perform any actions necessary.

The C-Bus Module detects that the PCI has powered up, and automatically restores any prior PCI setup modes or options.

Use this function to register the event handler. If no event handler is registered, this event will be ignored.

The registered function shall be of the form:

```
void my_event_handler(void);
```

C-Bus Module Interface Specification

10.9 Transparent Mode

Transparent mode is used when data is to be sent to and received from the PCI directly with no buffering or interpretation. This is used to provide direct access to the PCI, for example using an additional serial port and some external device.

By convention, all C-Bus software sends “~~~<CR>” to a PCI when it starts up. If the C-Bus Module is used in a configuration where another serial port is used to make a connection (through the C-Bus Module) to the PCI, then this sequence should be used to recognise a new connection request and behave accordingly.

10.9.1 Enter/Exit Transparent Mode

To make the C-Bus module enter or exit Transparent Mode, use the following function.

```
void cbus_vf_set_transparent_mode(cbush_boolean transparent_mode);
```

Set `transparent_mode` parameter to `cbush_true`, to Transparent Mode, and `cbush_false` to clear transparent mode.

When leaving Transparent Mode, it is necessary to re-initialise the C-Bus Module.

C-Bus Module Interface Specification

10.10 Serial I/O

The C-Bus Module does not include software to drive or interface to serial communication ports. Provision of this software external to the C-Bus Module provides the greatest cross-platform code portability.

Two serial ports are supported:

- The PC Interface Serial Port; and
- The Second Serial Port (generally used for communicating with a PC). This is only used when in Transparent Mode. See section 21.2 for more details.

Serial data access is performed using a transmit handler, which needs to be registered, and a character receive function.

10.10.1 Register C-Bus Output Function

```
void cbus_vf_register_serial_transmit_handler(void (*f)(char));
```

This function registers the serial transmission handler for the PCI serial data.

When the C-Bus Module wishes to send a character to the PCI, this event handler will be called. Message strings are sent one byte at a time by calling the registered function.

If no event handler is registered, this event will be ignored, and there will be no C-Bus communications. The registered function shall be of the form:

```
void my_event_handler(char transmit_character);
```

How the character is handled is up to the user. Options include:

- Send the character immediately, and return control when complete (for some types of embedded systems, this may cause a lot of “dead time” where the processor is waiting); or
- Add the character to a buffer, and send the buffer as time permits (usually using an interrupt service routine).

10.10.2 C-Bus Input

```
void cbus_vf_serial_receive_character(char character);
```

This function must to be called when a character is received from the PCI serial port.

Within the C-Bus module, the character will be added to a buffer and dealt with at the appropriate time.

10.10.3 Second Serial Port Output

```
void cbus_vf_register_serial_transmit_handler2(void (*f)(char));
```

Use this function to register the serial transmission handler for the second serial port.

C-Bus Module Interface Specification

When the C-Bus Module wishes to send a character to the second serial port, this event handler will be called. Data strings are sent one byte at a time by calling the registered function.

If no event handler is registered, this event will be ignored, and there will be no C-Bus communications with the second serial port. The registered function shall be of the form:

```
void my_event_handler(char transmit_character);
```

How the character is handled is up to the user. Options include:

- Send the character immediately, and return control when complete (for some types of embedded systems, this may cause a lot of “dead time” where the processor is waiting); or
- Add the character to a buffer, and send the buffer as time permits (usually using an interrupt service routine).

10.10.4 Second Serial Port Input

```
void cbus_vf_serial_receive_character2(char character);
```

This function is called when a character is received from the second serial port. Within the C-Bus module, the character will be sent to the C-Bus PCI if Transparent Mode is set.

C-Bus Module Interface Specification**10.11 Registering a Network**

```
cbus_boolean cbus_bf_register_network_path(int8u network_index,
                                           char * network_path);
```

This registers the network path in the database.

If the index is out of range (\geq CBUS_MAX_NETWORK_COUNT) or if the length of the path is greater than 14 characters plus a null terminator (15 characters total), this will return `cbus_false` and set an error code

If no more networks can be registered, the function will fail and set the error code to `CBM_TOO_MANY_NETWORKS`.

If the network path string is too long, the function will fail and set the error code to `CBM_STRING_TOO_LONG`.

Network index 0 is always used for the local network, and must not be registered.

Note: the network path must contain the full path information. For example, in an installation with a path of bridges with address A1, A2, A3, A4, A5, the paths to each network are:

Index	Network	Path
0	Device's Local Network	"" (does not need to be registered)
1	Network 1	"A109"
2	Network 2	"A112A2"
3	Network 3	"A11BA2A3"
4	Network 4	"A124A2A3A4"
5	Network 5	"A12DA2A3A4A5"

The path to the destination network is relative to the point where the device using the C-Bus Module is installed.

This could be different to the C-Bus "local network" displayed in the C-Bus Toolkit installation software.

For example, in Figure 1, the path from the C-Bus "Local Network" to Network A3 will be "A1A2A3", the path from Device 1 to Network A3 will be "A2A3" and the path from Device 2 to Network A3 will be "A3". The path from Device 2 to Network A2 will be "" and the path from Device 2 to Network A1 will be "A1".

C-Bus Module Interface Specification

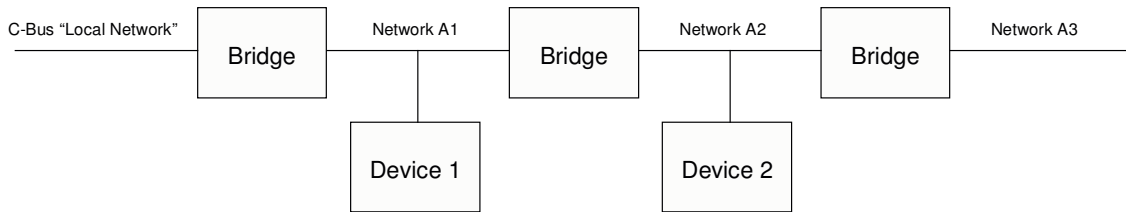


Figure 1 Network Routing Example

C-Bus Network Routing is described in more detail in the C-Bus Serial Interface Users Guide.

10.12 Get Last Source Network Index

```
int8u cbus_if_get_last_source_network_index(void);
```

This returns the index of the network of the most recently processed received message (see registering networks in section 10.11). If this function is used in an event handler, the network index relates to the event being handled.

If the received message is from a network that is not registered, the path to the network will be placed in the "spare" network slot (see section 7) and this index will be returned.

The contents of the spare network slot are only guaranteed to be valid during the life of the event handler. If this spare network slot is used (for example to send a reply), then either:

- ***The reply needs to be sent from within the event handler; or***
- ***The network path needs to be copied for later use, then registered before use.***

10.13 Get Last Source Unit Address Function

```
int8u cbus_if_get_last_source_unit(void);
```

This returns the source unit address of the most recently processed received message.

C-Bus Module Interface Specification**11 NETWORK DISCOVERY**

A C-Bus Network contains one or more units, each identified by their unique Unit Address (0 to 255). The Network Discovery functions can be used to determine which units are installed on a network, and find some details about those units.

The Network Discovery process consists of:

- a. Setting the discovery parameters (section 11.1.1); then
- b. Initiating the Network Discovery (section 11.1.2).

Once started, network discovery causes a series of events that report the results of the discovery (see 11.2.1). These events may take from a few seconds to a minute or more depending on the number of units on the network, and the number of unit parameters requested.

The process can be repeated for each C-Bus Network of interest.

11.1 Functions**11.1.1 Set Discovery Parameters**

The required unit parameters to be returned from the Network Discovery process can be set using the following function:

```
void cbus_discovery_vf_set_parameters(  
    int16u discovery_parameters);
```

discovery_parameters is a bitwise mask, where a “1” bit indicates a piece of information to be discovered, as listed below:

Bit	Parameter Value	Name	Data Returned
0	1	Unit Type	The C-Bus Unit Type (eg “KEY4” or “DIMDN8”)
1	2	Manufacturer	The manufacturer of the unit
2	4	Version	The firmware version of the unit
3	8	Group Addresses	List of Group Addresses used by the unit
4	16	Group Levels	List of Group Address levels
5	32	Applications	List of Application numbers used by the unit
6	64	Serial Number	Serial Number of the unit
7	128	Status	Common Status (see Serial Interface User Guide)
8	256	Area Address	Unit Area Address

C-Bus Module Interface Specification

The following constants are defined for these parameters, and these can be added or bitwise OR'ed to produce the desired discovery parameters.

```
#define CBUS_DISCOVERY_NONE          0x0000
#define CBUS_DISCOVERY_UNIT_TYPE     0x0001
#define CBUS_DISCOVERY_MANUFACTURER  0x0002
#define CBUS_DISCOVERY_VERSION       0x0004
#define CBUS_DISCOVERY_GROUPS        0x0008
#define CBUS_DISCOVERY_LEVELS        0x0010
#define CBUS_DISCOVERY_APPLICATIONS  0x0020
#define CBUS_DISCOVERY_SERIAL_NUMBER 0x0040
#define CBUS_DISCOVERY_STATUS        0x0080
#define CBUS_DISCOVERY_AREA          0x0100
#define CBUS_DISCOVERY_CONFLICT      0x8000
```

For example, the get the Unit Type and Version parameters, use:

```
cbus_discovery_vf_set_parameters(CBUS_DISCOVERY_UNIT_TYPE |
                                CBUS_DISCOVERY_VERSION);
```

11.1.2 Initiate Network Discovery

A Network discovery is initiated using:

```
void cbus_discovery_vf_initiate(int8u network);
```

The discovery will run until all of the requested information has been found and notified (using event handlers):

C-Bus Module Interface Specification**11.2 Discovery Events****11.2.1 Network Discovery Parameter Event**

Following the initiation of the Network Discovery, a series of events will return the discovered units and their parameters.

To register an event handler for these events, use:

```
void cbus_discovery_vf_register_parameter_handler(
    void (*f)(int8u,
              int8u,
              int16u,
              int8u *,
              int8u));
```

The registered function is called for each unit and parameter discovered. If no event handler is registered, discovery events will be ignored.

The event handler is passed the discovery network number, unit address, the discovery parameter (see section 11.1.1), a pointer to the parameter data, and the number of bytes of parameter data. The registered function shall be of the form:

```
void my_event_handler(int8u network,
                     int8u unit_address,
                     int16u parameter,
                     int8u * parameter_data,
                     int8u byte_count);
```

When a unit is first discovered, the event handler is called with the appropriate network and unit_address, but the parameter will be 0 (CBUS_DISCOVERY_NONE, meaning no data), and the data will be NULL.

If there is a unit address conflict (ie two or more C-Bus units at the same address), the parameter will be CBUS_DISCOVERY_CONFLICT.

This first event can be used to record the existence of a unit.

As each parameter of the unit is discovered, the event handler will be called with the appropriate parameter number and data as shown below:

Parameter Value	Name	Data Returned
1	Unit Type	8 bytes of ASCII data (padded with spaces)
2	Manufacturer	8 bytes of ASCII data (padded with spaces)
4	Version	8 bytes of ASCII data (padded with spaces)
8	Group Addresses	0 - 16 Group Addresses (FF means not used)
16	Group Levels	0 - 16 Group Address levels (corresponding to the Group Addresses from above)

C-Bus Module Interface Specification

Parameter Value	Name	Data Returned
32	Applications	2 Application addresses (FF means not used)
64	Serial Number	4 bytes
128	Status	1 byte (see Serial Interface User Guide)
256	Area Address	1 Area Address (FF means not used)

The order in which parameter events appear is:

1. Unit found (no parameter data)
2. Unit Type
3. Manufacturer
4. Version
5. Applications
6. Area Address
7. Serial Number
8. Status
9. Group Addresses
10. Levels

11.2.2 Network Discovery Complete Event

Following the completion of the Network Discovery, a completion event is raised. To register an event handler for this event, use the function:

```
void cbus_discovery_vf_register_complete_handler(void (*f)(void));
```

The registered function shall be of the form:

```
void my_event_handler(void);
```

11.2.3 Discovering Levels

If the options to discover Applications, Group Addresses and Levels are selected, the Lighting Database will be updated with the levels as they are discovered, and an MMI event will be raised if the discovered level is different from that in the database.

C-Bus Module Interface Specification

12 CAL SERVICES

C-Bus CAL commands are used to address individual units on a C-Bus network, usually for loading or extracting configuration or commissioning information.

Refer to section 21.4 for more details on using CAL commands to program a C-Bus Device.

12.1 Functions**12.1.1 Generic CAL Transmission**

```
void cbus_vf_CAL_send(int8u network,
                     int8u unit_address,
                     char *message_string,
                     int8u retries);
```

Generic CAL commands can be sent using this function.

If you do not understand what a CAL command is, or how to use CAL commands, then either read the C-Bus Serial Interface Users Guide, or contact Clipsal Integrated Systems Pty Ltd for more information.

In the event of a failure to transmit, the C-Bus Module uses the specified number of retries before aborting the transmission.

Any reply will take a minimum of 100ms to be received, since the receive buffer is not checked until 100ms after the command has been sent.

If there are many commands in the transmit and/or receive buffers, it could be several seconds between transmitting a command and receiving a reply.

If a reply is received, the CAL Response event handler will be called.

Ensure that the length of the message (header + network path + message string etc) does not exceed the `CBUS_OUTPUT_STRING_LENGTH` configuration parameter (see section 21.1). The `message_string` is a null terminated string that does not include the header (06), destination or network of the command.

On success the C-bus error code is set to `CBM_SUCCESS`.

If the `message_string` is too long the C-bus error code is set to `CBM_CMDSTR_TOO_LONG`.

12.1.2 Transmit CAL Reply

```
extern void cbus_vf_CAL_reply_send(int8u network,
                                   int8u unit_address,
                                   char *message_string,
                                   int8u retries);
```

CAL Reply commands can be sent using this function.

C-Bus Module Interface Specification

In the event of a failure to transmit, the C-Bus Module uses the specified number of retries before aborting the transmission.

The `message_string` is a null terminated string which does not include the header, destination or network of the message.

On success the C-bus error code is set to `CBM_SUCCESS`.

If the `message_string` is too long the C-bus error code is set to `CBM_CMDSTR_TOO_LONG`.

Do not use this function to return a reply to a CAL Read event. Use the special function `cbus_vf_CAL_read_response` instead.

12.1.3 Read Response

```
void cbus_vf_CAL_read_response(int8u network,  
                               int8u unit_address,  
                               int8u *data,  
                               int8u data_count);
```

Use this function to respond to CAL programming read events (see section 12.3).

`data` is a pointer to the data to be returned, `data_count` is the number of bytes to be returned.

On success the C-bus error code is set to `CBM_SUCCESS`.

If the `data` is too long the C-bus error code is set to `CBM_CMDSTR_TOO_LONG`.

C-Bus Module Interface Specification

12.2 CAL Events**12.2.1 CAL Response Handler**

```
void cbus_vf_CAL_register_response_handler(void (*f)(int8u *,
                                                    int8u));
```

Use this function to register a CAL Response event handler. If no event handler is registered, this event will be ignored.

This event handler will be called when the response to a (previously transmitted) CAL message is received. The event handler is passed the received C-Bus command as an array of bytes which includes the message header and network path.

The registered function shall be of the form:

```
void my_event_handler(int8u * command,
                      int8u command_length);
```

C-Bus Module Interface Specification

12.3 Device Programming Events

It is possible to load configuration data into the User Device, or extract configuration data from the User Device, both over C-Bus.

This is achieved by treating the C-Bus PCI as a bridge, and addressing the device on the other side: CAL Commands addressed to the PCI Unit address will be actioned internally by the PCI and will not be passed to the serial port. CAL commands that are addressed to a virtual unit address on the other side of the PCI “bridge” are sent to the serial port, and can be handled by the C-Bus module by calling an event handler.

12.3.1 Virtual Addressing Conventions

The memory that can be addressed through C-Bus **does not** have to reflect the internal memory map of the User Device – instead, any convention the user of the C-Bus Module chooses to apply can be used.

However, some C-Bus addressing conventions are enforced in order to provide a consistent interface to memory read and write processes:

Writing to Virtual Address 0 from C-Bus sets an address pointer. This pointer is the location at which subsequent read and write commands will access data.

Reading and writing at virtual address 1 accesses data at the location of the address pointer.

Accessing virtual address 0 and 1 is handled by the C-Bus module, generates Read and Write events, and causes the Read and Write event handlers to be called.

Accessing other virtual addresses will cause the Generic Read and Write Event Handlers to be called. Use of this method for reading and writing configuration data is STRONGLY discouraged.

Using C-Bus CAL commands other than STORE (to write) and RECALL (to read) will result in the Generic Programming Event Handler being called.

For more details on using CAL commands to program a Device, see section 21.4.2.

12.3.1.1 Generic Programming Event

```
void cbus_vf_CAL_register_generic_programming_handler(
    void (*f)(int8u *,
              int8u));
```

This function registers an event handler for generic CAL programming events.

The event handler is only called when a CAL Programming command is received and no applicable event handler is registered.

C-Bus Module Interface Specification

The event handler is passed the received C-Bus command as an array of bytes which includes the message header and network path. The registered function shall be of the form:

```
void my_event_handler(int8u * command,
                     int8u command_length);
```

12.3.1.2 Generic Write Event

```
void cbus_vf_CAL_register_generic_write_handler(
    cbus_boolean (*f)(int8u,
                      int8u,
                      int8u,
                      int8u *,
                      int8u));
```

This function registers an event handler for generic CAL programming write events.

The event handler will be called when a CAL Programming STORE command to virtual address \$02 - \$FF is received. If no event handler is registered, this event is ignored.

The event handler is passed the originating network number, unit address, the virtual address, the CAL command data string (stripped of the header) and the length of the command. The registered function shall be of the form:

```
cbus_boolean my_event_handler(int8u source_network,
                              int8u source_unit_address,
                              int8u virtual_address,
                              int8u * command,
                              int8u command_length);
```

If the write operation was successful, the event handler shall return `cbus_true`, otherwise `cbus_false`.

If the event handler returns `cbus_true`, the C-Bus Module sends a CAL ACKNOWLEDGE command back to the originator of the write command.

12.3.1.3 Generic Read Event

```
void cbus_vf_CAL_register_generic_read_handler(
    cbus_boolean (*f)(int8u,
                      int8u,
                      int8u,
                      int8u *,
                      int8u));
```

This function registers the event handler for generic CAL programming read events.

The event handler will be called when a CAL Programming RECALL command to virtual address \$02 - \$FF is received. If no event handler is registered, this event is ignored.

The event handler is passed the originating network number, unit address virtual address, and the number of bytes to be read. The event handler needs to copy the correct number of bytes to the result pointer.

C-Bus Module Interface Specification

The registered function shall be of the form:

```

cbus_boolean my_event_handler(int8u source_network,
                              int8u source_unit_address,
                              int8u virtual_address,
                              int8u *result
                              int8u data_count);

```

If the read operation is to succeed, the event handler shall return `cbus_true`, otherwise `cbus_false`.

If the event handler returns `cbus_true`, the C-Bus Module generates and sends a C-Bus CAL REPLY command.

12.3.1.4 Write Event

```

void cbus_vf_CAL_register_write_handler(
    cbus_boolean (*f)(int8u,
                      int8u,
                      int32u,
                      int8u *,
                      int8u));

```

This function registers the event handler for CAL programming write events.

The event handler will be called when a CAL Programming STORE command is received to Virtual Address \$01. If no event handler is registered, this event is ignored.

The event handler is passed the originating network number, unit address, the address of the write, a pointer to the data and the amount of data. The registered function shall be of the form:

```

cbus_boolean my_event_handler(int8u source_network,
                              int8u source_unit_address,
                              int32u address,
                              int8u *data,
                              int8u data_count);

```

If the write operation was successful, the event handler shall return `cbus_true`, otherwise `cbus_false`.

If the event handler returns `cbus_true`, the C-Bus Module sends a CAL ACKNOWLEDGE command back to the originator of the write command.

12.3.1.5 Read Event

```

void cbus_vf_CAL_register_read_handler(
    cbus_boolean (*f)(int8u,
                      int8u,
                      int32u,
                      int8u *,
                      int8u));

```

This function registers the event handler for CAL programming read events.

C-Bus Module Interface Specification

The event handler will be called when a CAL Programming RECALL command is received to Virtual Address \$01. If no event handler is registered, this event is ignored.

The event handler is passed the originating network number, unit address, the address of the read, a pointer to a data buffer and the amount of data. The event handler needs to copy the correct number of bytes to the result pointer. The registered function shall be of the form:

```
cbus_boolean my_event_handler(int8u  source_network,  
                             int8u  source_unit_address,  
                             int32u address,  
                             int8u  *result  
                             int8u  data_count);
```

If the read operation is to succeed, the event handler shall return `cbus_true`, otherwise `cbus_false`.

If the event handler returns `cbus_true`, the C-Bus Module generates and sends a C-Bus CAL REPLY command.

C-Bus Module Interface Specification

13 LIGHTING APPLICATION

This section should be read in conjunction with the C-Bus Lighting Application document, CBUS-APP/02.

When built for C-Bus Enabled Level of 2 or more, the C-Bus Module includes a Lighting Application database. This provides a model of the C-Bus Lighting Application group addresses that can be interrogated to determine levels.

The number of Lighting Applications that can be stored in the database is set in the file `cbus_config.h`². The application number is passed to the various functions, and the index of the Application (0, 1, 2, ...) is looked up by the function before the database records can be accessed.

Each Lighting Application instance includes a set of Group Records. In normal cases, there are 256 Group Address records for each Lighting Application instance. Each Group Address has a record which stores the current level along with details such as whether it is ramping and how fast, timer settings and other parameters needed for the light state machine. The state machine maintains the ramping levels and will automatically handle timer time-out actions.

For some embedded systems with tight memory constraints, the number of Group Records can be reduced. In that case, each Group of interest must be registered before it is available in the database.

The C-Bus lighting application database can only be accessed using the functions defined in the header file.

² Pre-compiled versions of the C-Bus Module (for Windows and Linux) are built for 10 Lighting Applications.

C-Bus Module Interface Specification**13.1 C-Bus Lighting Ramp Rates**

Where functions require a ramp rate, the value used is a ramp rate token in the range 0 – 15 (*not the duration in seconds*).

Ramp rates and their tokens are:

Ramp Rate Token	Ramp Rate
0	Instantaneous
1	4 seconds
2	8 seconds
3	12 seconds
4	20 seconds
5	30 seconds
6	40 seconds
7	1 minute
8	1.5 minutes
9	2 minutes
10	3 minutes
11	5 minutes
12	7 minutes
13	10 minutes
14	15 minutes
15	17 minutes

When a command is used to ramp to level 0 with a non-instant ramp rate, the C-Bus Module issues a ramp to level 1 command. Then, when the level gets to level 1, the C-Bus Module will automatically issue a ramp to level 0 command.

This behaviour is consistent with the C-Bus Scene Master and ensures that old (Mask 1) Key Input Unit LEDs are always on when the Group Address level is non-zero.

C-Bus Module Interface Specification

13.2 Database Functions*13.2.1 Application Registration**13.2.1.1 Register Application Number*

```
cbus_boolean cbus_lighting_bf_register_app(int8u app);
```

This registers the lighting application number in the database.

If the database cannot support any more application numbers (ie. Application count > 10), this returns `cbus_false` and sets the error code to `CBM_TOO_MANY_APPLICATIONS`.

13.2.1.2 Get Application Index

```
int16s cbus_lighting_if_app_index(int8u app);
```

This returns the index of the application in the database. It returns -1 if the Application Number was not found and sets the error code to `CBM_APPLICATION_NOT_FOUND`.

13.2.2 Group Registration

Group address registration is not required if the C-Bus Module is built using 256 groups per Lighting Application (ie. `CBUS_MAX_LIGHTING_GROUPS = 256`).³

13.2.2.1 Register Group Address

```
cbus_boolean cbus_lighting_bf_register_group(int8u app,  
                                             int8u group);
```

This registers the lighting group address in the database. If the database can not support any more group addresses (ie. Group count > `CBUS_MAX_LIGHTING_GROUPS`), it returns `cbus_false` and sets the error code to `CBM_TOO_MANY_GROUPS`.

If the application is not registered, the function will fail and set the error code to `CBM_APPLICATION_NOT_FOUND`.

13.2.2.2 Get Group Index

```
char cbus_lighting_if_group_index(int8u app,  
                                  int8u group);
```

This returns the index of the Group Address in the database.

The data base stores up to `CBUS_MAX_LIGHTING_GROUPS` Group Addresses per Application, with index 0 to `CBUS_MAX_LIGHTING_GROUPS - 1`.

³ Pre-compiled versions of the C-Bus Module (for Windows and Linux) are built for 256 groups per Lighting Application. Consequently, group registration is not required.

C-Bus Module Interface Specification

If the Group Address was not found, this returns -1 and sets the error code to **CBM_GROUP_NOT_FOUND**.

If the application is not registered, the function will fail and set the error code to **CBM_APPLICATION_NOT_FOUND**.

13.2.2.3 *Clear Registered Groups*

```
void cbus_lighting_vf_clear_groups(void);
```

This clears all registered Group Addresses. This is only applicable when there are less than 256 Groups per Application.

13.2.3 Database Operations

13.2.3.1 *Set Database Level*

```
void cbus_lighting_vf_set_database_level(int8u application,  
                                         int8u group,  
                                         int8u level,  
                                         int8u ramp_rate);
```

This sets the level in the database **without** sending a corresponding C-Bus command.

If the application is not registered, the function will fail and set the error code to **CBM_APPLICATION_NOT_FOUND**.

If the group is not registered, the function will fail and set the error code to **CBM_GROUP_NOT_FOUND**.

13.2.3.2 *Get Level*

```
int8u cbus_lighting_if_get_level(int8u network,  
                                 int8u application,  
                                 int8u group);
```

This returns the level of a Group Address from the database (which should reflect the value on C-Bus).

If the application is not registered, the function will fail and set the error code to **CBM_APPLICATION_NOT_FOUND** and return a level of 0.

If the group is not registered, the function will fail and set the error code to **CBM_GROUP_NOT_FOUND** and return a level of 0.

C-Bus Module Interface Specification

13.3 C-Bus Lighting Operations**13.3.1 Set Level**

```
void cbus_lighting_vf_set_level(int8u network,  
                                int8u application,  
                                int8u group,  
                                int8u level,  
                                int8u ramp_rate,  
                                int8u retries,  
                                int8u force);
```

This sets a lighting group address to a level with a particular ramp rate. It makes a specific number of retries to get the message onto C-Bus.

force is used to re-assert the level into the network, in the event that an MMI indicates a discrepancy between the database level and the reported level. **force** is the number of times the level will be copied from the database into the network in the event of a discrepancy (and when that count is expired, the database will not be used to further update the network).

If the application is not registered, the function will fail and set the error code to **CBM_APPLICATION_NOT_FOUND**.

If the group is not registered, the function will fail and set the error code to **CBM_GROUP_NOT_FOUND**.

The function will fail if the group is set to 255. The error code will be set to **CBM_GRP_EQ_255** in this case.

C-Bus Module Interface Specification

13.4 Timer Operations

The following functions are used for timers on the lighting application. All times are in seconds.

13.4.1 Set Timer

```
void cbus_lighting_vf_set_timer(int8u network,  
                                int8u application,  
                                int8u group,  
                                int16s duration,  
                                int16s expiry_level);
```

Set a timer to expire after `duration` seconds. On expiry, send a C-Bus command to set the level to `expiry_level`. If `expiry_level` is `-1`, the current level will be restored on timer expiry.

If the application is not registered, the function will fail and set the error code to `CBM_APPLICATION_NOT_FOUND`.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

13.4.2 Get Amount Of Time Left On A Timer

```
int16s cbus_lighting_if_get_timer(int8u application,  
                                   int8u group);
```

This returns the amount of time left on a Group's timer, in seconds.

13.4.3 Stop Timer and Execute Action

```
void cbus_lighting_vf_terminate_timer(int8u application,  
                                       int8u group);
```

This stops the timer and executes the associated C-Bus action.

If the application is not registered, the function will fail and set the error code to `CBM_APPLICATION_NOT_FOUND`.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

13.4.4 Stop Timer with No Action

```
void cbus_lighting_vf_reset_timer(int8u application,  
                                   int8u group);
```

This stops the timer without executing the C-Bus action.

C-Bus Module Interface Specification

If the application is not registered, the function will fail and set the error code to **CBM_APPLICATION_NOT_FOUND**.

If the group is not registered, the function will fail and set the error code to **CBM_GROUP_NOT_FOUND**.

C-Bus Module Interface Specification

13.5 MMI Operations**13.5.1 Initiate MMI**

```
void cbus_lighting_vf_initiate_MMI(int8u network,  
                                   int8u application);
```

This executes a C-Bus MMI on the selected Network and Application.

If connected through a version 1-3 PCI, it will initiate a status MMI; for a version 4+ PCI, it will initiate a series of level MMI(s), which will synchronise the database to the C-Bus Group Address levels.

This function can be used to re-synchronise the database with C-Bus in exceptional circumstances. During normal operation, the C-Bus Module interacts with C-Bus to maintain synchronisation

It is not necessary, or desirable, to use regular MMIs to synchronise the database to the C-Bus levels.

13.5.2 Initiate Status MMI

```
void cbus_lighting_vf_initiate_status_MMI(int8u network,  
                                           int8u application);
```

This starts a status MMI on the selected Network and Application.

This function is not normally expected to be used.

13.5.3 Initiate Level MMI

```
void cbus_lighting_vf_initiate_level_MMI(int8u network,  
                                          int8u application,  
                                          int8u start_group);
```

This starts a level MMI on the selected Network and Application, at the specified starting group.

The level MMI will cause the database levels to be updated for the block of 32 groups, beginning at `start_group`.

The value of `start_group` must be one of 0x00, 0x20, 0x40, 0x60, 0x80, 0xA0, 0xC0 or 0xE0. Operation of C-Bus equipment with other values is not guaranteed.

C-Bus Module Interface Specification

13.6 Labels**13.6.1 Apply Lighting Label**

```
void cbus_lighting_vf_label(int8u network,
                           int8u application,
                           int8u group,
                           int8u flavour,
                           cbus_et_label_type type,
                           int8u language,
                           int8u length,
                           int8u *text);
```

Labels can be applied to lighting groups and trigger control group / action selectors in DLT switches.

To label a group, set:

application:	lighting application you wish to use
group:	group to be labelled
flavour:	0, 1, 2 or 3 depending on the group button flavour to be labelled (normally 0)
type:	<code>cbus_lighting_ce_label_text</code>
language:	the language code (see <code>cbus_lighting.h</code>)
length:	the length of the label string
text:	pointer to the label string (16 chars max)

More sophisticated labelling using dynamic icons is also possible - refer to the C-Bus lighting application documentation for more details.

13.6.2 Set Preferred Language for Labels

```
void cbus_lighting_vf_set_language(int8u network,
                                   int8u application,
                                   int8u language);
```

Sets the preferred language for label display. If the language has no labels loaded, the English labels will be displayed.

C-Bus Module Interface Specification

13.7 Events

13.7.1 Database Change Event

```
void cbus_lighting_vf_register_database_event_handler(  
    void (*f)(int8u,  
              int8u,  
              int8u));
```

This function registers the event handler for the Lighting Database change events. If the database is changed due to an external event or due to an internal operation, then this event handler will be called (if registered).

This handler may also be compile time enabled to be called every time the level changes due to a database level changing due to a ramping event.

Every time the `cbus_lighting_vf_set_database_level` function is called, this event handler will be called.

The event handler can be used to save the data to non-volatile memory for recall after power failures. The parameters passed are the Application Number, the Group Address and the new target level. The registered function shall be of the form:

```
void my_event_handler(int8u application,  
                      int8u group,  
                      int8u level);
```

13.7.2 Lighting Event

```
void cbus_lighting_vf_register_event_handler(  
    void (*f)(int8u,  
              int8u,  
              int8u,  
              int8u));
```

If a Lighting message is received, the database will be automatically updated and the event handler will be called (if registered).

This registers an event handler for Lighting Application messages. When any event on the Lighting Application is received, the registered event will be called. The parameters passed are the Application Number, the Group Address, the new level and the ramp rate index (see section 13.1).

The registered function shall be of the form:

```
void my_event_handler(int8u application,  
                      int8u group,  
                      int8u level,  
                      int8u ramp_rate);
```

C-Bus Module Interface Specification

13.7.3 Alternate Lighting Event

```
void cbus_lighting_vf_register_event_handler2(  
    void (*f)(int8u,  
              int8u,  
              int8u,  
              int8u,  
              cbus_boolean));
```

If a Lighting message is received, the database will be automatically updated and this event handler will be called (if registered).

This registers an event handler for Lighting Application messages. The parameters passed are the Application Number, the Group Address, the new level, the ramp rate index (see section 13.1), and whether the received command was an ON/OFF command (as opposed to a ramp to level command).

This event handler will not normally be required, the previous one is preferred.

The registered function shall be of the form:

```
void my_event_handler(int8u application,  
                     int8u group,  
                     int8u level,  
                     int8u ramp_rate,  
                     cbus_boolean on_off_command);
```

13.7.4 Additional Alternate Lighting Event

```
void cbus_lighting_vf_register_event_handler3(  
    void (*f)(int8u,  
              int8u,  
              int8u,  
              int8u,  
              cbus_lighting_et_source_type));
```

If a Lighting message is received, the database will be automatically updated and this event handler will be called (if registered).

This registers an event handler for Lighting Application messages. The parameters passed are the Application Number, the Group Address, the new level, the ramp rate index (see section 13.1), and an enumerated type indicating the source of the change to the lighting group.

This event handler will not normally be required, the previous one is preferred.

Possible source types are:

C-Bus Module Interface Specification

Enumerant	Meaning
<code>cbus_lighting_ce_source_cbus</code>	A command was received from C-Bus which caused this event
<code>cbus_lighting_ce_source_MMI</code>	An MMI discrepancy was corrected and caused this event
<code>cbus_lighting_ce_source_ramp</code>	A ramping group caused this event
<code>cbus_lighting_ce_source_self</code>	A function call by the user caused this event

The registered function shall be of the form:

```
void my_event_handler(int8u application,
                      int8u group,
                      int8u level,
                      int8u ramp_rate,
                      cbus_lighting_et_source_type source);
```

13.7.5 MMI Event

```
void cbus_lighting_vf_register_MMI_event_handler(
    void (*f)(int8u,
              int8u,
              int8u,
              int8u));
```

This function registers the event handler for the Lighting Application MMI correction messages.

If there is a discrepancy between the state of a group reported in an MMI and the state in the database, the database will be corrected. When the MMI mechanism corrects a Lighting Group Address level, the registered event will be called. The parameters passed are the Application Number, the Group Address and the new level.

The ramp rate is always zero, but is included to make the interface identical to the lighting event handler, so that the one event handler can be registered for BOTH events if required.

The registered function shall be of the form:

```
void my_event_handler(int8u application,
                      int8u group,
                      int8u level,
                      int8u ramp_rate);
```

C-Bus Module Interface Specification

13.7.6 Label Event

```
cbus_lighting_vf_register_label_handler(  
    void (*f)(int8u,  
              int8u,  
              int8u,  
              cbus_et_label_type,  
              int8u,  
              int8u,  
              int8u *));
```

This function registers the event handler for the received Label commands.

The registered function shall be of the form:

```
void my_event_handler(int8u application,  
                      int8u group,  
                      int8u flavour,  
                      cbus_et_label_type label_type,  
                      int8u language,  
                      int8u label_length,  
                      int8u *text);
```

Meanings of these parameters are described in section 13.6.1.

C-Bus Module Interface Specification

14 TRIGGER CONTROL APPLICATION

This section should be read in conjunction with the Trigger Control Application document, CBUS-APP/07.

14.1 Network Variables

The Trigger Control Application network variables are stored in a database and the values are automatically maintained.

14.2 C-Bus Functions

Note that registering groups is not required if using 256 groups per Application (ie. `CBUS_MAX_LIGHTING_GROUPS = 256`).

If less than 256 groups per Application are used, the Trigger Groups must be registered using the lighting group registration functions, with application `CBUS_TRIGGER_APP`.

14.2.1 Set Trigger Group

```
void cbus_trigger_vf_set_group(int8u network,
                              int8u trigger_group,
                              int8u action_selector,
                              int8u retries);
```

This activates a Trigger Group/Action Selector combination. It makes a specific number of retries to get the message onto C-Bus.

If the `trigger_group` is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

The function will fail if the `trigger_group` is set to 255. The error code will be set to `CBM_GRP_EQ_255` in this case.

14.2.2 Indicator Kill

```
void cbus_trigger_vf_indicator_killgroup(int8u network,
                                         int8u trigger_group,
                                         int8u retries);
```

This transmits a C-Bus Indicator Kill message on the Trigger Group. This is used to switch off the indicators for any units that have a scene active on that Trigger Group.

The Trigger Group does not have to be registered.

The function will fail if the `trigger_group` is set to 255. The error code will be set to `CBM_GRP_EQ_255` in this case.

C-Bus Module Interface Specification

14.3 Labels**14.3.1 Apply Trigger Label**

```
void cbus_trigger_vf_label(int8u network,
                          int8u group,
                          int8u action_selector,
                          int8u flavour,
                          cbus_et_label_type type,
                          int8u language,
                          int8u length,
                          int8u *text);
```

Labels can be applied to scene trigger keys, by trigger group + action selector in DLT switches.

To label a Trigger Control group/action selector pair used for a scene, set:

group:	trigger group to be labelled
action_sel:	action selector to be labelled
flavour:	0
type:	<code>cbus_ce_label_text</code>
language:	the language code (see <code>cbus_lighting.h</code>)
length:	the length of the label string
text:	pointer to the label string (16 chars max)

More sophisticated labelling using dynamic icons is also possible - refer to the C-Bus lighting application documentation for more details.

14.3.2 Set Preferred Language for Labels

```
void cbus_trigger_vf_set_language(int8u network,
                                 int8u language);
```

Sets the preferred language for control trigger label display. If the language has no labels loaded, the English labels will be displayed.

C-Bus Module Interface Specification

14.4 Database Functions

Note that registering groups is not required if using 256 groups per Application (ie. `CBUS_MAX_LIGHTING_GROUPS = 256`).

If less than 256 groups per Application are used, the Trigger Groups must be registered using the lighting group registration functions, with application `CBUS_TRIGGER_APP`.

14.4.1 Set Database Level

```
void cbus_trigger_vf_set_database(int8u trigger_group,  
                                int8u action_selector);
```

This sets the Trigger Group in the database without sending a corresponding C-Bus command.

If the `trigger_group` is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

14.4.2 Get Trigger

```
int8u cbus_trigger_if_get_group(int8u network,  
                                int8u trigger_group);
```

This returns the value of a Trigger Group from the database (which should reflect the value on C-Bus).

C-Bus Module Interface Specification

14.5 Timer Functions

The following functions are used for timers on the trigger control application. All times are in seconds.

14.5.1 Set Timer

```
void cbus_trigger_vf_set_timer(int8u network,
                              int8u trigger_group,
                              int16s duration,
                              int16s expiry_level);
```

Set a timer to expire after `duration` seconds. On expiry, send a C-Bus command to set the level to `expiry_level`. If `expiry_level` is `-1`, the current level will be restored on timer expiry.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

14.5.2 Get Amount Of Time Left On A Timer

```
int16s cbus_trigger_if_get_timer(int8u trigger_group);
```

This returns the amount of time left on a Group's timer, in seconds.

14.5.3 Stop Timer and Execute Action

```
void cbus_trigger_vf_terminate_timer(int8u trigger_group);
```

Stop the timer and execute the C-Bus action.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

14.5.4 Stop Timer with No Action

```
void cbus_trigger_vf_reset_timer(int8u trigger_group);
```

This stops the timer without executing the C-Bus action.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

C-Bus Module Interface Specification

14.6 Events**14.6.1 Trigger Events**

```
void cbus_trigger_vf_register_handler(void (*f)(int8u,  
                                              int8u));
```

This registers the event handler for the Trigger Control Application messages. If a Trigger Control message is received, the database will be automatically updated and the event handler will be called (if registered). The parameters passed are the Trigger Group and the new action selector.

The registered function shall be of the form:

```
void my_event_handler(int8u trigger_group,  
                     int8u action_selector);
```

14.6.2 Label Events

```
cbus_trigger_vf_register_label_handler(  
    void (*f)(int8u,  
              int8u,  
              int8u,  
              cbus_et_label_type,  
              int8u,  
              int8u,  
              int8u *));
```

This function registers the event handler for the received Label commands.

The registered function shall be of the form:

```
void my_event_handler(int8u group,  
                     int8u action_selector,  
                     int8u flavour,  
                     cbus_et_label_type label_type,  
                     int8u language,  
                     int8u label_length,  
                     int8u *text);
```

Meanings of these parameters are described in section 14.3.1.

C-Bus Module Interface Specification

15 ENABLE CONTROL APPLICATION

This section should be read in conjunction with the C-Bus Enable Control Application document, CBUS-APP/08.

15.1 Network Variables

The Enable Control Application network variables are stored in a database and the values are automatically maintained.

15.2 C-Bus Functions

Note that registering groups is not required if using 256 groups per Application (ie. `CBUS_MAX_LIGHTING_GROUPS = 256`).

If less than 256 groups per Application are used, the Enable Groups must be registered using the lighting group registration functions, with application `CBUS_ENABLE_APP`.

15.2.1 Set Enable Group

```
void cbus_enable_vf_set_group(int8u network,  
                             int8u enable_group,  
                             int8u value,  
                             int8u retries);
```

This sets the Enable Group to a particular value. It makes a specific number of retries to get the message onto C-Bus.

If the `enable_group` is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

The function will fail if the `enable_group` is set to 255. The error code will be set to `CBM_GRP_EQ_255` in this case.

C-Bus Module Interface Specification

15.3 Database Functions

Note that registering groups is not required if using 256 groups per Application (ie. `CBUS_MAX_LIGHTING_GROUPS = 256`).

If less than 256 groups per Application are used, the Enable Groups must be registered using the lighting group registration functions, with application `CBUS_ENABLE_APP`.

15.3.1 Set Database Level

```
void cbus_enable_vf_set_database(int8u enable_group,  
                                int8u value);
```

This sets the Enable Group in the database without sending a corresponding C-Bus command.

If the `enable_group` is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

15.3.2 Get Enable

```
int8u cbus_enable_if_get_group(int8u network,  
                                int8u enable_group);
```

This returns the value of an Enable Group from the database (which should reflect the value on C-Bus).

C-Bus Module Interface Specification

15.4 Timer Functions

The following functions are used for timers on the Enable control application. All times are in seconds.

15.4.1 Set Timer

```
void cbus_enable_vf_set_timer(int8u network,  
                             int8u enable_group,  
                             int16s duration,  
                             int16s expiry_level);
```

Set a timer to expire after `duration` seconds. On expiry, send a C-Bus command to set the level to `expiry_level`. If `expiry_level` is `-1`, the current level will be restored on timer expiry.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

15.4.2 Get Amount Of Time Left On A Timer

```
int16s cbus_enable_if_get_timer(int8u enable_group);
```

This returns the amount of time left on a Group's timer, in seconds.

15.4.3 Stop Timer and Execute Action

```
void cbus_enable_vf_terminate_timer(int8u enable_group);
```

Stop the timer and execute the C-Bus action.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

15.4.4 Stop Timer with No Action

```
void cbus_enable_vf_reset_timer(int8u enable_group);
```

This stops the timer without executing the C-Bus action.

If the group is not registered, the function will fail and set the error code to `CBM_GROUP_NOT_FOUND`.

C-Bus Module Interface Specification

15.5 Events

```
void cbus_enable_vf_register_handler(void (*f)(int8u,  
                                              int8u));
```

This registers the event handler for the Enable Control Application messages. If an Enable Control message is received, the database will be automatically updated and the event handler will be called (if registered). The parameters passed are the Enable Group and the value.

The registered function shall be of the form:

```
void my_event_handler(int8u trigger_group,  
                      int8u value);
```

C-Bus Module Interface Specification

16 DATE AND TIME APPLICATION

This section should be read in conjunction with the C-Bus Clocks and Timekeeping Application document, CBUS-APP/23.

16.1 Network Variables

The Date and Time Application has two Network Variables: the Date and the Time.

These variables are automatically updated in response to received C-Bus messages and are maintained in real-time, therefore providing calendar services.

The storage format for the date and time are defined in the header file.

These variables are exported as a structure and can be read directly.

Direct update of these variables is not supported, and could cause unpredictable results.

16.2 General Purpose Functions**16.2.1 Days In Month**

```
int8u cbus_time_if_days_in_month(int8u m,  
                                int16u y);
```

This returns the number of days in month *m* of year *y*.

16.2.2 Day Number

```
int16u cbus_time_if_calculate_day_number(  
    cbus_time_st_time_type *time);
```

This calculates the day number. This is the number of days (inclusive) that have elapsed since 1 January 2000.

16.2.3 Day Number to Day Of Week

```
int8u cbus_time_if_day_number_to_day_of_week(int16u day_number);
```

This calculates the day of the week from the day number. Sunday = 0 ... Saturday = 6.

16.2.4 Day of Week

```
int8u cbus_time_if_calculate_day_of_week(  
    cbus_time_st_time_type *time);
```

This calculates the day of the week from the date. Sunday = 0 ... Saturday = 6.

C-Bus Module Interface Specification

16.2.5 Copy Time

```
void cbus_time_vf_copy_time(cbus_time_st_time_type *source_date,  
                           cbus_time_st_time_type *dest_date);
```

This copies the date and time from the `source_date` structure to the `dest_date` structure.

16.2.6 Get Time

```
void cbus_time_vf_get_time(cbus_time_st_time_type *user_time);
```

This gets the current time, as an alternative to using or copying the exported time / date structure.

16.2.7 Set Time

```
void cbus_time_vf_set_time(cbus_time_st_time_type * new_time);
```

This sets the current time.

This function updates the local copy of the date & time, but does not make a transmission onto C-Bus.

C-Bus Module Interface Specification

16.3 C-Bus Functions**16.3.1 Send Date**

```
void cbus_time_vf_send_date(int8u network);
```

This transmits the current date onto C-Bus.

16.3.2 Send Time

```
void cbus_time_vf_send_time(int8u network);
```

This transmits the current time onto C-Bus.

16.3.3 Send Date and Time

```
void cbus_time_vf_send_date_time(int8u network);
```

This transmits the current date and time onto C-Bus.

16.4 Events

```
void cbus_time_vf_register_handler(  
    void (*f)(cbus_boolean),  
    cbus_time_st_time_type,  
    cbus_time_st_time_type);
```

This registers the event handler for the Date/Time Application messages.

If a Date/Time message is received, the database will be automatically updated and the event handler will be called (if registered).

The parameter passed indicates whether the date changed (otherwise the time has changed).

The registered function shall be of the form:

```
void my_event_handler(cbus_boolean date_changed,  
    cbus_time_st_time_type *initial_date_time,  
    cbus_time_st_time_type *adjusted_date_time);
```

C-Bus Module Interface Specification

17 TELEPHONY APPLICATION

This section should be read in conjunction with the C-Bus Telephone Status and Control Application document, CBUS-APP/24.

The Telephony Application module usage depends on whether the device using the module is a C-Bus Telephone Interface or some other device.

For example, a “Divert” command will be received by a Telephone Interface as an event to trigger some action, but other devices would only send “Divert” commands, and would not care about the event.

Any device can use any of the functions or events, but they are separated in this document for clarity.

17.1 Network Variables

The Telephony Application does not have Network Variables. The only interfaces to the Telephony Application are function calls and events.

C-Bus Module Interface Specification

17.2 Non-Telephone Interface (Slave) Usage**17.2.1 Functions**

The following functions provide the means of transmitting telephony commands (usually to a C-Bus Telephone Interface).

17.2.1.1 Isolate Secondary Output

```
void cbus_telephony_vf_isolate_secondary_outlet (
    int8u network,
    cbus_telephony_et_isolate_secondary_type isolate_status);
```

This sends a command to set the isolation of the secondary outlet of a C-Bus Telephone Interface. The `isolate_status` variable definition is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_normal</code>	Set the secondary outlet to a normal connection.
<code>cbus_telephony_ce_isolated</code>	Isolate the secondary outlet from the incoming telephone line.

17.2.1.2 Recall Last Number

```
void cbus_telephony_vf_recall_last_number (
    int8u network,
    cbus_telephony_et_recall_number_type type);
```

This sends a command to retrieve the last incoming or outgoing telephone number. The returned value indicates if the command was successful. The `type` variable definition is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_outgoing</code>	Ask the telephone interface to retrieve and return the telephone number of the last outgoing call.
<code>cbus_telephony_ce_incoming</code>	Ask the telephone interface to retrieve and return the telephone number of the last incoming call ⁴ .

⁴ Retrieving the telephone numbers of incoming calls from a C-Bus telephone interface requires the customer to have Calling Line Identification enabled on their telephone line, and that the line identification operate in a supported format.

C-Bus Module Interface Specification

17.2.1.3 *Reject Incoming Calls*

```
void cbus_telephony_vf_reject_incoming_call(int8u network);
```

This causes the current incoming call to be rejected.

17.2.1.4 *Divert*

```
void cbus_telephony_vf_divert(int8u network,  
                             char * number);
```

This sends a command the telephone interface, requesting that it divert incoming calls to another telephone number.

17.2.1.5 *Clear Diversion*

```
void cbus_telephony_vf_clear_diversion(int8u network);
```

This sends a command the telephone interface, requesting that it cancel the diversion of phone numbers.

C-Bus Module Interface Specification

17.2.2 Events

The relevant event handler will be called when a Telephony message is received from a C-Bus Telephone interface.

17.2.2.1 On/Off Hook Event

```
void cbus_telephony_vf_hook_handler(void (*f) (
    cbus_boolean,
    cbus_telephony_et_off_hook_type,
    char *));
```

This registers the event handler for the Telephony Application on/off hook messages.

When an On Hook or Off Hook event on the Telephony Application is received, the registered handler will be called. The data passed is whether the event is an on-hook or off-hook event, and if it is an off-hook, the reason and the telephone number (16 characters max). The reason variable definition is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_incoming_voice</code>	Telephone interface has answered an incoming voice call in voice mode
<code>cbus_telephony_ce_incoming_data</code>	Telephone interface has answered an incoming voice call in data mode
<code>cbus_telephony_ce_incoming_other</code>	The incoming call has unknown mode
<code>cbus_telephony_ce_outgoing_voice</code>	The telephone interface is making an outgoing voice call
<code>cbus_telephony_ce_outgoing_data</code>	The telephone interface is making an outgoing data call
<code>cbus_telephony_ce_outgoing_other</code>	An outgoing call has unknown mode
<code>cbus_telephony_ce_setting_diversion</code>	The telephone interface is setting a diversion
<code>cbus_telephony_ce_clearing_diversion</code>	The telephone interface is clearing a diversion

The registered function shall be of the form:

```
void my_event_handler(
    cbus_boolean on_hook,
    cbus_telephony_et_off_hook_type reason,
    char * number);
```

If `on_hook` is `cbus_true`, then the event is an on-hook event, otherwise it is an off-hook event.

C-Bus Module Interface Specification

17.2.2.2 *Dial-Out Failure Event*

```
void cbus_telephony_vf_dial_out_failure_handler(
    void (*f)(cbus_telephony_et_dial_out_failure_type));
```

This registers the event handler for the Telephony Application failure messages.

When a Dial-Out Failure event on the Telephony Application is received, the registered handler will be called. The data passed is the reason for the failure. The **reason** variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_no_dial_tone</code>	Dial out failed because there was no dial tone found
<code>cbus_telephony_ce_no_answer</code>	Dial out failed because there was no answer
<code>cbus_telephony_ce_no_ack</code>	Dial out failed because the called party did not acknowledge the message
<code>cbus_telephony_ce_number_unobtainable</code>	Dial out failed because the number is unobtainable
<code>cbus_telephony_ce_number_busy</code>	Dial out failed because the number is busy
<code>cbus_telephony_ce_internal_failure</code>	Dial out failed for some unknown reason

The registered function shall be of the form:

```
void my_event_handler(
    cbus_telephony_et_dial_out_failure_type reason);
```

17.2.2.3 *Dial-In Failure Event*

```
void cbus_telephony_vf_dial_in_failure_handler(
    void (*f)(cbus_telephony_et_dial_in_failure_type));
```

This registers the event handler for the Telephony Application failure messages. When a Dial-In Failure event on the Telephony Application is received, the registered event will be called. The data passed is the reason for the failure. The **reason** variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_stopped_ringing</code>	An incoming call failed because it stopped ringing before it was answered

C-Bus Module Interface Specification

The registered function shall be of the form:

```
void my_event_handler(  
    cbus_telephony_et_dial_in_failure_type reason);
```

17.2.2.4 *Ringing Event*

```
void cbus_telephony_vf_ringing_handler(void (*f)(char *));
```

This registers an event handler for the Telephony Application Ringing message.

When a Ringing event on the Telephony Application is received, the registered event will be called. The data passed is the calling telephone number.

The registered function shall be of the form:

```
void my_event_handler(char * number);
```

17.2.2.5 *Internet Connection Request Event*

```
void cbus_telephony_vf_internet_connection_handler(void  
    (*f)(void));
```

This registers an event handler for the Telephony Application Internet Connection message.

When an Internet Connection Request event on the Telephony Application is received, the registered event will be called. The registered function shall be of the form:

```
void my_event_handler(void);
```

It is expected that this event could be used by a device to establish an Internet connection, normally after the current telephone call has completed.

C-Bus Module Interface Specification

17.3 Telephone Interface (Master) Usage

The following functions and event handlers are only applicable to the Telephone Interface, and should not be used by any other devices.

17.3.1 Functions

The following functions provide the means of transmitting telephony commands.

17.3.1.1 Line On Hook

```
void cbus_telephony_vf_line_on_hook(int8u network);
```

This sends a Line On Hook telephony command onto C-Bus.

17.3.1.2 Line Off Hook

```
void cbus_telephony_vf_line_off_hook(  
    int8u network,  
    cbus_telephony_et_off_hook_type reason,  
    char * number);
```

This sends a Line Off Hook telephony command onto C-Bus, including the reason code and the telephone number (16 characters max). The reason variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_incoming_voice</code>	Telephone interface has answered an incoming voice call in voice mode
<code>cbus_telephony_ce_incoming_data</code>	Telephone interface has answered an incoming voice call in data mode
<code>cbus_telephony_ce_incoming_other</code>	The incoming call has unknown mode
<code>cbus_telephony_ce_outgoing_voice</code>	The telephone interface is making an outgoing voice call
<code>cbus_telephony_ce_outgoing_data</code>	The telephone interface is making an outgoing data call
<code>cbus_telephony_ce_outgoing_other</code>	An outgoing call has unknown mode
<code>cbus_telephony_ce_setting_diversion</code>	The telephone interface is setting a diversion
<code>cbus_telephony_ce_clearing_diversion</code>	The telephone interface is clearing a diversion

C-Bus Module Interface Specification

17.3.1.3 *Dial-Out Failure*

```
void cbus_telephony_vf_dial_out_failure(
    int8u network,
    cbus_telephony_et_dial_out_failure_type reason);
```

This sends a Telephony dial-out failure command onto C-Bus, including the reason. The reason variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_no_dial_tone</code>	Dial out failed because there was no dial tone found
<code>cbus_telephony_ce_no_answer</code>	Dial out failed because there was no answer
<code>cbus_telephony_ce_no_ack</code>	Dial out failed because the called party did not acknowledge the message
<code>cbus_telephony_ce_number_unobtainable</code>	Dial out failed because the number is unobtainable
<code>cbus_telephony_ce_number_busy</code>	Dial out failed because the number is busy
<code>cbus_telephony_ce_internal_failure</code>	Dial out failed for some unknown reason

17.3.1.4 *Dial-In Failure*

```
void cbus_telephony_vf_dial_in_failure(
    int8u network,
    cbus_telephony_et_dial_in_failure_type reason);
```

This sends a Telephony dial-in failure command onto C-Bus, including the reason. The reason variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_stopped_ringing</code>	An incoming call failed because it stopped ringing before it was answered

17.3.1.5 *Ringing*

```
void cbus_telephony_vf_ringing(int8u network,
                               char * number);
```

This sends a Ringing telephony command onto C-Bus with the telephone number.

C-Bus Module Interface Specification

17.3.1.6 *Recall Number Response*

```
void cbus_telephony_vf_recall_number_response(  
    int8u network,  
    cbus_telephony_et_recall_number_type type,  
    char * number);
```

This sends a response to the telephony Recall Last Number Request command onto C-Bus with the type and the telephone number. The type variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_outgoing</code>	The telephone is returning the number of the last outgoing call.
<code>cbus_telephony_ce_incoming</code>	The telephone interface is returning the number of the last incoming call.

17.3.1.7 *Request Internet Connection*

```
void cbus_telephony_vf_internet_connection_request(int8u network);
```

This sends a command to request that an Internet connection be made (by another device).

C-Bus Module Interface Specification

17.3.2 Events

If a telephony message is received, the appropriate event handler will be called.

17.3.2.1 *Isolate Secondary Outlet Event*

```
void cbus_telephony_vf_isolate_secondary_handler(
    void (*f)(cbus_telephony_et_isolate_secondary_type));
```

This registers an event handler for the Telephony Application Isolate Secondary messages.

When an Isolate Secondary Outlet message on the Telephony Application is received, the registered handler will be called. The parameter passed is the Isolation Status. The status variable is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_normal</code>	Set the secondary outlet to a normal connection.
<code>cbus_telephony_ce_isolated</code>	Isolate the secondary outlet from the incoming telephone line.

The registered function shall be of the form:

```
void my_event_handler(
    cbus_telephony_et_isolate_secondary_type status);
```

17.3.2.2 *Recall Last Number Request Event*

```
void cbus_telephony_vf_recall_last_number_handler(
    void (*f)(cbus_telephony_et_recall_number_type));
```

This registers an event handler for the Telephony Application Request Last Number messages.

When a Recall Last Number message on the Telephony Application is received, the registered handler will be called. The parameter passed is the type. The type variable definition is an enumerated type:

Enumerant	Meaning
<code>cbus_telephony_ce_outgoing</code>	The telephone interface has been asked to retrieve and return the telephone number of the last outgoing call.
<code>cbus_telephony_ce_incoming</code>	The telephone interface has been asked to retrieve and return the telephone number of the last incoming call.

C-Bus Module Interface Specification

The registered function shall be of the form:

```
void my_event_handler(cbus_telephony_et_recall_number_type type);
```

17.3.2.3 *Reject Incoming Call Event*

```
void cbus_telephony_vf_reject_call_handler(void (*f)(void));
```

This registers an event handler for the Telephony Application Reject Incoming Call message.

When a Reject Incoming Call message on the Telephony Application is received, the registered handler will be called.

The registered function shall be of the form:

```
void my_event_handler(void);
```

17.3.2.4 *Divert Event*

```
void cbus_telephony_vf_divert_handler(void (*f)(char *));
```

This registers an event handler for the Telephony Application Divert message.

When a Divert message on the Telephony Application is received, the registered handler will be called. The parameter passed is the telephone number the diversion should be set to.

The registered function shall be of the form:

```
void my_event_handler(char * number);
```

17.3.2.5 *Clear Diversion Event*

```
void cbus_telephony_vf_clear_diversion_handler(void (*f)(void));
```

This registers an event handler for the Telephony Application Clear Diversion message.

When a Clear Diversion message on the Telephony Application is received, the registered handler will be called.

The registered function shall be of the form:

```
void my_event_handler(void);
```

C-Bus Module Interface Specification**18 SECURITY APPLICATION**

This section should be read in conjunction with the C-Bus Security Application document, CBUS-APP/05.

The Security Application module usage depends on whether the device using the module is a Security Panel or some other device.

For example, an "Arm System" command will be transmitted by a Security Panel as a command, but other devices would only receive "Arm System" commands as an event.

Any device can use any of the functions or events, but they are separated in this document for clarity.

18.1 Network Variables

The Security Application maintains the following model of the security system as network variables:

- Zone Isolation (`cbus_true` / `cbus_false`)
- State of Tamper, Panic, Alarm, Low Battery, Main, Line Cut, Arm Fail, Fire Alarm, Gas Alarm, Other Alarm (`cbus_true` / `cbus_false`)
- Zone State, as the enumerated type:

`cbus_security_et_zone_state_type`

Enumerant	Meaning
<code>cbus_security_ce_zone_sealed</code>	Zone is sealed
<code>cbus_security_ce_zone_unsealed</code>	Zone is unsealed
<code>cbus_security_ce_zone_open</code>	Zone is open circuit
<code>cbus_security_ce_zone_short</code>	Zone is short circuit

- System Arm State, as the enumerated type:

`cbus_security_et_arm_code_type`

Enumerant	Meaning
<code>cbus_security_ce_disarmed</code>	Security system is disarmed
<code>cbus_security_ce_fully_armed</code>	Security system is completely armed
<code>cbus_security_ce_partially_armed</code>	Security system is partly armed

C-Bus Module Interface Specification

18.2 Security Database Functions

Values are automatically set when sending or receiving C-Bus Commands.

Setting these values (using a ..._set_... call) does NOT cause a C-Bus message to be sent.

All status variables (except Zone Status) are reset when a System Disarm command is sent or received.

18.2.1 Get Zone State

```
cbus_security_et_zone_state_type  
cbus_security_ef_get_zone_state(int8u zone_number);
```

This returns the state of a zone.

18.2.2 Set Zone State

```
void cbus_security_vf_set_zone_state(  
    int8u zone_number,  
    cbus_security_et_zone_state_type zone_state);
```

This sets the state of a zone.

18.2.3 Get Zone Isolation

```
cbus_boolean cbus_security_bf_get_zone_isolation(  
    int8u zone_number);
```

This returns `cbus_true` if the zone is isolated.

18.2.4 Set Zone Isolation

```
void cbus_security_vf_set_zone_isolation(  
    int8u zone_number,  
    cbus_boolean zone_isolated);
```

This sets zone to isolated if `cbus_true`, or non-isolated otherwise.

18.2.5 Get System Armed Status

```
cbus_security_et_arm_code_type  
cbus_security_ef_get_armed_status(void);
```

This returns the system armed state.

C-Bus Module Interface Specification

18.2.6 Set System Armed Status

```
void cbus_security_vf_set_armed_status(
    cbus_security_et_arm_code_type armed_status);
```

This sets the system armed state.

18.2.7 Get System State

```
cbus_boolean cbus_security_bf_get_status(
    cbus_security_et_status_message_type status_item);
```

This returns the status of *item*, which is an enumerated type. The state returned is **cbus_true** (meaning that item is active) or **cbus_false** (meaning that the item is not active). The possible items to query are:

cbus_security_et_status_message_type

Enumerant	Meaning
cbus_security_ce_status_alarm	If cbus_true , an alarm is active
cbus_security_ce_status_tamper	If cbus_true , tamper is active
cbus_security_ce_status_panic	If cbus_true , panic is active
cbus_security_ce_status_low_battery	If cbus_true , low battery is active
cbus_security_ce_status_mains_failure	If cbus_true , mains has failed
cbus_security_ce_status_line_cut	If cbus_true , a telephone line has been cut
cbus_security_ce_status_arm_failed	If cbus_true , an arm failed
cbus_security_ce_status_fire_alarm	If cbus_true , a fire alarm is active
cbus_security_ce_status_gas_alarm	If cbus_true , a gas alarm is active
cbus_security_ce_status_other_alarm	If cbus_true , some other alarm type is active

18.2.8 Set System Status

```
void cbus_security_vf_set_status(
    cbus_security_et_status_message_type item,
    cbus_boolean state);
```

This sets the state of the parameter in the database. The items are as described above.

C-Bus Module Interface Specification

18.3 Non-Security Panel (Slave) Usage

These functions and events are used to interact with a C-Bus security system.

18.3.1 Functions

18.3.1.1 Status Request

```
void cbus_security_vf_status_request(int8u network);
```

This requests an update of the status of the attached security system, by sending Status 1 & 2 Request messages onto C-Bus. The security panel will reply with Status 1 and Status 2 messages. Any changes to the armed, tamper, panic or zone states will cause the appropriate event handlers to be called.

18.3.1.2 Arm System

```
void cbus_security_vf_arm_system(
    int8u network,
    cbus_security_et_arm_type arm_mode);
```

This sends a command to arm the security system. The Arm Mode is an enumerated type:

`cbus_security_et_arm_type`

Enumerant	Meaning
<code>cbus_security_ce_away_mode</code>	Arm in away mode
<code>cbus_security_ce_night_mode</code>	Arm in night mode
<code>cbus_security_ce_day_mode</code>	Arm in day mode
<code>cbus_security_ce_vacation_mode</code>	Arm in vacation mode
<code>cbus_security_ce_highest_level</code>	Arm to the highest level of protection possible

18.3.1.3 Set / Drop Tamper

```
extern void cbus_security_vf_tamper_status(
    int8u network,
    cbus_boolean tamper_status);
```

This requests that the security system raise (if `tamper_state = cbus_true`) or drop its tamper state.

18.3.1.4 Raise Alarm

```
void cbus_security_vf_raise_alarm(int8u network);
```

This requests that the security system raise an alarm condition.

C-Bus Module Interface Specification

18.3.1.5 *Emulate Keypad*

```
void cbus_security_vf_emulate_keypad(int8u network,  
                                     char key);
```

This sends an Emulate Keypad security command onto C-Bus. The key parameter is an ASCII character (see the Security Application Documentation). This will be interpreted by the security system however is appropriate for its current mode and type of operation (normal entry of a password).

18.3.1.6 *Display Message*

```
void cbus_security_vf_display_message(int8u network,  
                                     char * message);
```

This requests that the security system display the supplied message⁵.

18.3.1.7 *Request Zone Name*

```
void cbus_security_vf_request_zone_name(int8u network,  
                                       int8u zone_number);
```

This requests that the security system return the name of the specified zone⁶.

⁵ Not all security systems support display of messages.

⁶ Not all security systems support returning zone names.

C-Bus Module Interface Specification

18.3.2 Events

If a message is received (from a security system), the relevant event handler will be called.

When a Status Report message is received, it may trigger a Zone Status, System Arm Status, Tamper Status or Panic Status event.

An event handler will only be called if the received message sets a state different to that stored in the database.

18.3.2.1 System Armed Event

```
void cbus_security_vf_system_armed_handler(  
    void (*f) (cbus_security_et_arm_code_type));
```

This registers an event handler for the Security Application Armed message.

When a System Armed or System Disarmed message is received on the Security Application, the registered handler will be called. The Arm Code Type as defined in section 18.1, is passed as a parameter.

The registered function shall be of the form:

```
void my_event_handler(  
    cbus_security_et_arm_code_type arm_code_type);
```

18.3.2.2 Exit Delay Started Event

```
void cbus_security_vf_exit_delay_started_handler(void (*f) (void));
```

This registers an event handler for an Exit Delay Started message on the Security Application. When such a message is received, the registered handler will be called.

The registered function shall be of the form:

```
void my_event_handler(void);
```

18.3.2.3 Entry Delay Started Event

```
void cbus_security_vf_entry_delay_started_handler(void  
    (*f) (void));
```

This registers an event handler for an Entry Delay Started message on the Security Application. When such a message is received, the registered handler will be called.

```
void my_event_handler(void);
```

C-Bus Module Interface Specification

18.3.2.4 Status Event

```
void cbus_security_vf_status_message_handler(  
    void (*f)(cbus_security_et_status_message_type,  
              cbus_boolean));
```

This registers an event handler for Status messages on the Security Application. When status messages are received, the registered handler will be called.

The handler is passed the system state that has changed, and what it has changed to. Possible system states are shown in section 18.2.7.

The registered function shall be of the form:

```
void my_event_handler(  
    cbus_security_et_status_message_type message_type,  
    cbus_boolean status);
```

18.3.2.5 Zone State Event

```
void cbus_security_vf_zone_handler(void (*f)(int8u));
```

This registers an event handler for the Zone state change messages on the Security Application. When a Zone Unsealed, Zone Sealed, Zone Open, Zone Short, Zone Isolated or Status Report message is received on the Security Application, the registered handler will be called. The zone number is passed as a parameter.

Use the `cbus_security_ef_get_zone_status` function to find what the zone state changed to.

The registered function shall be of the form:

```
void my_event_handler(int8u zone_number);
```

Special Cases: If a Status Report 1 is received, the Zone number parameter will be \$FE. If a Status Report 1 is received, the Zone number parameter will be \$FF.

C-Bus Module Interface Specification

18.3.2.6 *Battery Charging Event*

```
void cbus_security_vf_battery_charging_handler(
    void (*f)(cbus_security_et_battery_charging_type));
```

This registers an event handler for Battery Charge messages on the Security Application. When a Battery Charging message on the Security Application is received, the registered handler will be called. The battery charge state is an enumerated type, and is passed as a parameter:

cbus_security_et_battery_charging_type

Enumerant	Meaning
cbus_security_ce_charging_stopped	Battery charging has stopped (was previously charging)
cbus_security_ce_charging_started	Battery charging has started (was previously not charging)

The registered function shall be of the form:

```
void my_event_handler(
    cbus_security_et_battery_charging_type state);
```

18.3.2.7 *Zone Name Event*

```
void cbus_security_vf_zone_name_handler(void (*f)(int8u,
    char *));
```

This registers an event handler for Zone Name Security Application messages. When a Zone Name message on the Security Application is received, the registered handler will be called. The zone number and the name of the zone are passed as parameters. The registered function shall be of the form:

```
void my_event_handler(int8u zone_number,
    char * zone_name);
```

18.3.2.8 *Password Entry Event*

```
void cbus_security_vf_password_entry_status_handler(
    void (*f)(cbus_security_et_password_entry_type));
```

This registers an event handler for the Password Entry Security Application messages. When a Password Entry message on the Security Application is received, the registered handler will be called. The password entry state is passed as a parameter and is an enumerated type:

C-Bus Module Interface Specification

cbus_security_et_password_entry_type

Enumerant	Meaning
cbus_security_ce_password_succeeded	Password entry succeeded
cbus_security_ce_password_failed	Password entry failed
cbus_security_ce_password_disabled	Password entry has been disabled (was previously enabled)
cbus_security_ce_password_enabled	Password entry has been enabled (was previously disabled)

The registered function shall be of the form:

```
void my_event_handler(cbsecurity_et_password_entry_type code);
```

18.3.2.9 *Arm Ready / Not Ready Event*

```
void cbus_security_vf_arm_status_handler(void (*f)(int8u));
```

This registers an event handler for Arm Ready / Not Ready messages on the Security Application. When an Arm Ready / Not Ready message on the Security Application is received, the registered handler will be called.

The applicable zone code is passed as a parameter. The registered function shall be of the form:

```
void my_event_handler(int8u zone_number);
```

C-Bus Module Interface Specification

18.4 Security Panel (Master) Usage

The following functions and event handlers are only applicable to Security Panels, and should not be used by any other devices.

18.4.1 Functions

The following functions provide the means of transmitting Security commands.

18.4.1.1 System Arm / Disarm

```
void cbus_security_vf_system_arm_status(  
    int8u network,  
    cbus_security_et_arm_code_type arm_code);
```

This sends a System Armed or System Disarmed Security Application command onto C-Bus.

18.4.1.2 Exit Delay Started

```
void cbus_security_vf_exit_delay_started(int8u network);
```

This sends an Exit Delay Started Security Application command onto C-Bus.

18.4.1.3 Entry Delay Started

```
void cbus_security_vf_entry_delay_started(int8u network);
```

This sends an Entry Delay Started Security Application command onto C-Bus.

18.4.1.4 System State

```
void cbus_security_vf_status_message(  
    int8u network,  
    cbus_security_et_status_message_type message_type,  
    cbus_boolean status);
```

This sends a Security Application status command onto C-Bus. The `message_type` is described in section 18.2.7.

18.4.1.5 Zone State

```
void cbus_security_vf_zone_state(  
    int8u network,  
    int8u zone_number,  
    cbus_security_et_zone_state_type zone_state);
```

This sends a Zone Unsealed, Zone Sealed, Zone Open or Zone Short Security Application command onto C-Bus. The zone state is described in section 18.1.

C-Bus Module Interface Specification

18.4.1.6 Zone Isolated

```
void cbus_security_vf_zone_isolated(int8u network,  
                                   int8u zone_number);
```

This sends a Zone Isolated Security Application command onto C-Bus.

18.4.1.7 Battery Charging

```
void cbus_security_vf_battery_charging(  
    int8u network,  
    battery_charging_type battery_status);
```

This sends a Battery Charging Security Application command onto C-Bus.

18.4.1.8 Zone Name

```
void cbus_security_vf_zone_name(int8u network,  
                                int8u zone_number,  
                                char * zone_name);
```

This sends a Zone Name Security Application command onto C-Bus.

The zone name shall be an 11 character ASCII string, padded with spaces if necessary, and null terminated (null terminator is the 12th byte).

This shall ONLY be sent in response to a received Request Zone Name command.

18.4.1.9 Status Report 1

```
void cbus_security_vf_status_report_1(int8u network);
```

This sends a Status Report 1 Security Application command onto C-Bus.

This shall only be sent in response to a Status 1 Request command.

18.4.1.10 Status Report 2

```
void cbus_security_vf_status_report_2(int8u network);
```

This sends a Status Report 2 Security Application command onto C-Bus.

This shall only be sent in response to a Status 2 Request command.

18.4.1.11 Password Entry Status

```
void cbus_security_vf_password_entry_status(  
    int8u network,  
    cbus_security_et_password_entry_type password_status);
```

This sends a Password Entry Status Security Application command onto C-Bus.

The password status is described in section 18.3.2.8.

C-Bus Module Interface Specification

18.4.1.12 Arm Ready / Not Ready

```
void cbus_security_vf_arm_ready(int8u network,  
                                int8u zone_number);
```

This sends an Arm Ready or Arm Not Ready Security Application command onto C-Bus. The zone number of any unsealed zone is passed as a parameter.

<p><i>The zone number is 0 if the system has armed correctly.</i></p>
--

C-Bus Module Interface Specification

18.4.2 Events

If a Security message is received, the appropriate event handler will be called (if registered).

18.4.2.1 *Status 1 Request Event*

```
void cbus_security_vf_status_1_request_handler(void (f)(void));
```

This registers an event handler for the Status 1 Request Security Application message. When a Status Request 1 command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(void);
```

18.4.2.2 *Status 2 Request Event*

```
void cbus_security_vf_status_2_request_handler(void (f)(void));
```

This registers an event handler for the Status 2 Request Security Application message. When a Status Request 2 command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(void);
```

18.4.2.3 *Arm System Event*

```
void cbus_security_vf_arm_system_handler(
    void (*f)(cbus_security_et_arm_type));
```

This registers an event handler for the Arm System Security Application message. When an Arm System command on the Security Application is received, the registered handler will be called. The arm type is passed as a parameter, and is described in section 18.3.1.2.

The registered function shall be of the form:

```
void my_event_handler(cbus_security_et_arm_type arm_mode);
```

18.4.2.4 *Alarm Event*

```
void cbus_security_vf_alarm_handler(void (*f)(cbus_boolean));
```

This registers an event handler for the Alarm Security Application message. When an Alarm command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(cbus_boolean alarm_status);
```

C-Bus Module Interface Specification

18.4.2.5 Tamper Event

```
void cbus_security_vf_tamper_handler(void (*f)(cbus_boolean));
```

This registers an event handler for Tamper Security Application messages. When a Tamper command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(cbus_boolean tamper_status);
```

18.4.2.6 Emulate Keyboard Event

```
void cbus_security_vf_emulate_keypad_handler(void (f)(char));
```

This registers an event handler for Emulate Keypad Security Application messages. When an Emulate Keypad event on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(char key);
```

18.4.2.7 Display Message Event

```
void cbus_security_vf_display_message_handler(void (f)(char *));
```

This registers an event handler for the Display Message Security Application command. When a Display Message command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(char * message);
```

18.4.2.8 Request Zone Name Event

```
void cbus_security_vf_request_zone_name_handler(void (f)(int8u));
```

This registers an event handler for the Request Zone Name Security Application command. When a Request Zone Name command on the Security Application is received, the registered handler will be called. The registered function shall be of the form:

```
void my_event_handler(int8u zone_number);
```

C-Bus Module Interface Specification

19 MEASUREMENT APPLICATION

This section should be read in conjunction with the C-Bus Measurement Application document, CBUS-APP/28.

19.1 Network Variables

The Measurement Application has network variables corresponding to the value of each channel of each measurement device.

The number and type of measurement devices is highly flexible and at the discretion of an installer. Consequently, the C-Bus Module does not maintain the value of these variables internally. It just sends and receives the commands.

Measurements have units (of measurement), which are described using the following enumerated type:

`cbus_measurement_et_unit_type`

Enumerant	Meaning
<code>cbus_measurement_ce_centigrade</code>	Temperature
<code>cbus_measurement_ce_amp</code>	Current
<code>cbus_measurement_ce_degree</code>	Angular displacement
<code>cbus_measurement_ce_coulomb</code>	(Electric) charge
<code>cbus_measurement_ce_true_false</code>	Boolean stuff
<code>cbus_measurement_ce_farad</code>	Capacitance
<code>cbus_measurement_ce_henry</code>	Inductance
<code>cbus_measurement_ce_hertz</code>	Frequency
<code>cbus_measurement_ce_joule</code>	Energy
<code>cbus_measurement_ce_katal</code>	Rate of catalytic activity
<code>cbus_measurement_ce_kg_m3</code>	Density
<code>cbus_measurement_ce_kg</code>	Mass
<code>cbus_measurement_ce_litre</code>	Volume
<code>cbus_measurement_ce_litre_h</code>	Very slow flow rates
<code>cbus_measurement_ce_litre_m</code>	Slow flow rate
<code>cbus_measurement_ce_litre_s</code>	Flow rate
<code>cbus_measurement_ce_lux</code>	Light level
<code>cbus_measurement_ce_metre</code>	Distance
<code>cbus_measurement_ce_metre_m</code>	Slow speed
<code>cbus_measurement_ce_metre_s</code>	Speed
<code>cbus_measurement_ce_metre_s2</code>	Acceleration

C-Bus Module Interface Specification

Enumerant	Meaning
cbus_measurement_ce_mole	Quantity of substance
cbus_measurement_ce_newton_metre	Torque
cbus_measurement_ce_newton	Force
cbus_measurement_ce_ohm	Resistance
cbus_measurement_ce_pascal	Pressure
cbus_measurement_ce_percent	Humidity, generic percentages & linear ratios
cbus_measurement_ce_decibel	Logarithmic ratio
cbus_measurement_ce_ppm	Concentrations
cbus_measurement_ce_rpm	Angular speed
cbus_measurement_ce_second	Elapsed Time
cbus_measurement_ce_minute	Elapsed Time
cbus_measurement_ce_hour	Elapsed Time
cbus_measurement_ce_sievert	Radiation
cbus_measurement_ce_steradian	Units of solid angle
cbus_measurement_ce_tesla	Magnetic field strength
cbus_measurement_ce_volt	Voltage
cbus_measurement_ce_watt_hour	Power consumption
cbus_measurement_ce_watt	Power
cbus_measurement_ce_weber	Magnetic Flux
cbus_measurement_ce_no_unit	Unitless quantities
cbus_measurement_ce_custom	User defined

C-Bus Module Interface Specification

19.2 Functions

19.2.1 Send Measurement Data

```
void cbus_measurement_vf_send_data(  
    int8u network,  
    int8u device_id,  
    int8u channel,  
    int32s mantissa,  
    int8s exponent,  
    cbus_measurement_et_unit_type units);
```

This function sends a Measurement Application message, where

channel is the measurement channel number

mantissa is the numeric value of the measured parameter.

exponent is the multiplier (power of ten)

units is the units of the measured parameter, as selected from the table in section 19.1.

19.3 Events

```
void cbus_measurement_vf_register_handler(  
    void (*f)(int8u,  
              int8u,  
              int32s,  
              int8s,  
              cbus_measurement_et_unit_type));
```

This registers the event handler for the Measurement Application messages.

If a Measurement Application message is received, the event handler will be called (if registered). The parameters passed are the Device Identifier, the Channel number, the mantissa and exponent of the measured value and the Unit Type, as defined in section 19.1.

The registered function shall be of the form:

```
void my_event_handler(int8u device_id,  
                      int8u channel,  
                      int32s mantissa,  
                      int8s exponent,  
                      cbus_measurement_et_unit_type units);
```

C-Bus Module Interface Specification

20 C-BUS STRING FUNCTIONS

The following functions are provided in the `cbus_str` unit. They are required for the other parts of the C-Bus Module, and have been provided for general use also.

20.1 Is Hexadecimal Char

```
cbus_boolean cbus_str_is_hex_char(char c);
```

This function returns `cbus_true` if the character is a hexadecimal character.

20.2 Hexadecimal Char to Number

```
int8u cbus_str_hex_char_to_number(char c);
```

This function converts hexadecimal character `c` into a number.

20.3 Hexadecimal Pair to Number

```
int8u cbus_str_hex_pair_to_number(char *s);
```

This function converts hexadecimal pair of characters into a byte.

20.4 Hexadecimal String to Number

```
int32s cbus_str_hex_string_to_number(char *s);
```

This function converts string of hexadecimal characters into a number.

20.5 Number to Hexadecimal Char

```
char cbus_str_number_to_hex_char(int8u n);
```

This function converts a number to a hexadecimal digit.

20.6 Append an Integer to a String

```
void cbus_str_add_integer_to_string(char *s,  
                                   int32s n,  
                                   int8u digits);
```

This function adds an integer number to the end of a string (including sign). If the number of digits is non-zero, it puts in that many digits and pads with zeros. If the number of digits is zero, it leaves off leading zeros.

C-Bus Module Interface Specification

20.7 Append a Hex Number to a String

```
void cbus_str_add_hex_to_string(char * s,  
                               int32s n,  
                               int8u digits);
```

This function adds a hexadecimal number to the end of a string. If the number of digits is non-zero, it puts in that many digits and pads with zeros. If the number of digits is zero, it leaves off leading zeros.

20.8 Percent String

```
void cbus_str_percent_string(char * s,  
                             int8u Level);
```

This function converts a level to a percentage string.

20.9 Append Carriage Return

```
void cbus_str_add_CR_to_string(char * s);
```

This function appends a carriage return to a string.

20.10 Convert a String to a Hex String

```
void cbus_str_add_hex_string_to_string(char * DestString,  
                                       char * SourceString);
```

This function converts each character of an ASCII string to hexadecimal character pairs and appends them to a string. eg. `cbus_str_add_hex_string_to_string(s, "Test")` where `s` is "1234" will convert "Test" to hex pairs ("54657374") and append it, leaving `s` at "123454657374".

C-Bus Module Interface Specification

21 USAGE NOTES**21.1 C-Bus Module Customisation⁷**

The C-Bus Module can be customised to suit user application requirements and processor capability. All customisation is performed in the `cbus_config.h` file using `#define` statements. No changes should be made to any of the other C-Bus Module files.

A default `cbus_config.h` file (`cbus_config_default.h`) is provided and should be copied to the file name `cbus_config.h` and then customised. This is done so that when a new version of the C-Bus Module files is issued, your `cbus_config.h` file will not be over-written.

Users of versions of the C-Bus module prior to version 3 will see that the format of the `cbus_config.h` file has changed very substantially. When upgrading it will be necessary to use the new format, and make tailorings based on the old file.

The `cbus_config.h` file includes extensive comments about which sections can be changed, and which cannot. It also includes consistency checking to ensure that only mutually compatible options are selected.

The optional customisations that can be made are:

- Selection of which C-Bus Applications are to be supported.
- Selection of whether the C-Bus Lighting Database is required (this provides you with the ability to interrogate levels, to support MMIs and to enable C-Bus timers).
- If the C-Bus Database is implemented, you have the options to select how many Applications and Group Addresses you will support within the database.
- Selection of the C-Bus Enabled Level (1 – 5).
- Selection of the “tick” rate.
- Selection of whether the C-Bus queues are processed on every “tick” or on alternating “ticks”.
- Enabling or disabling support for labelling C-Bus DLT units.
- Selection of C-Bus queue lengths.
- Selection of the C-Bus queue item and buffer lengths.

21.1.1 C-Bus Enabled Level

Setting the C-Bus Enabled Level automatically sets several other options, but some can be overridden if desired. The features that are included at each of the levels are

⁷ For customers who receive the C-Bus Module in pre-compiled form, the `cbus_config.h` file used during compilation is supplied. However changes to this file should **not** be made, and will have no effect.

C-Bus Module Interface Specification

shown in the table below. Higher levels include the features of all lower levels. For more details, refer to the C-Bus Enabled Program documentation.

Note that the features do not exactly match the C-Bus Enabled Program features at each level.

C-Bus Enabled Level	Features Introduced at this level
1	Transmit SAL Commands
2	Receive SAL Commands Receive CAL Commands Transmit CAL Responses
3	Transmit CAL Commands C-Bus Database
4	Handle MMI messages and maintain Database

21.1.2 Application Selection

Various C-Bus Applications can be selected, as required. The `cbus_config.h` file should be customised (see section 9.1.1) as appropriate.

21.1.3 Tick Rate

The Tick Rate is set in the `cbus_config.h` file. This is the rate at which the `cbus_ef_update` function gets called (see section 10.1). This must execute an update to the C-Bus Database (if the Lighting Application is being used) every 200ms. Consequently, the Tick Rate must be a divisor of 200ms (for example, 5ms, 10ms, 20ms, 25ms, 50ms, 100ms or 200ms).

The other constraint on the Tick Rate depends on how the C-Bus transmit data is handled:

- If transmit data is sent to a queue and handled by an interrupt handler, then there is no problem;
- Alternatively, it is possible that each time the Transmit Event Handler (see section 10.10.1) is called, the character is sent to a serial port, but control is not returned until transmission is complete. To send a whole string could take up to 45ms (45 characters x 1 ms (approx) / character) to send. This means that the `cbus_ef_update` function should not be called more often than every 100ms (to allow some margin of error), and hence the tick rate should be 100ms or 200ms.

21.1.4 C-Bus Module Queues

The C-Bus Module queue (buffer) lengths are set in the `cbus_config.h` file.

The transmit queue length depends on several factors:

C-Bus Module Interface Specification

- How fast items can be put into the queue (for example, scenes with lots of commands)
- How fast the queue is emptied (the Tick Rate)
- How busy C-Bus is (this controls how many retries are required).

The receive queue length depends on:

- How fast items can be put into the queue (one every 16ms worst case – this is unlikely to be sustained for very long).
- How fast the queue is emptied (the Tick Rate).
- Whether the queue is allowed to be processed completely every tick (see section 10.6.3).

Calculation of the queue lengths is difficult, and trial and error may be required.

21.1.5 Memory Usage

The amount of RAM used by the C-Bus Module depends on:

- The C-Bus Enabled Level
- The C-Bus Applications Used
- The configuration of the C-Bus Database (if used)

Approximate RAM usage with various options is shown in the table below. For these examples, the Lighting, Trigger and Enable Applications were active.

When using the Lighting Database, the incremental memory usage is 12 bytes per Group Address per Application

Hence having an additional Lighting Application of 256 Group Addresses will require an additional 3K of RAM.

C-Bus Enabled Level	Options	RAM Usage
1	3 Applications x 10 Group Addresses, Transmit Queue length = 5	0.6K
1	3 Applications x 256 Group Addresses, Transmit Queue length = 50	2.5K
2	3 Applications x 10 Group Addresses, Transmit Queue length = 5, Receive Queue length = 2	0.8K
2	3 Applications x 256 Group Addresses, Transmit Queue length = 50, Receive Queue length = 20	4.7K
3	3 Applications x 10 Group Addresses, Transmit Queue length = 5, Receive Queue length = 2	1.2K

C-Bus Module Interface Specification

C-Bus Enabled Level	Options	RAM Usage
3	3 Applications x 256 Group Addresses, Transmit Queue length = 50, Receive Queue length = 20	14.2K
4	3 Applications x 10 Group Addresses, Transmit Queue length = 5, Receive Queue length = 2	1.2K
4	3 Applications x 256 Group Addresses, Transmit Queue length = 50, Receive Queue length = 20	14.2K

The amount of program memory required depends on:

- The C-Bus Enabled Level
- The C-Bus Applications Used
- The target processor

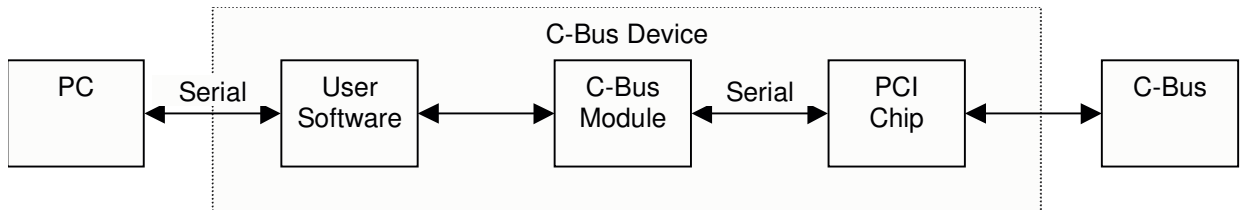
Typical program memory usage for a Hitachi H8S processor (with no optimisation) compiled with various options are shown below.

C-Bus Enabled Level	Options	Program Memory Used
1	Lighting, Enable and Trigger Applications	6.0K
1	Lighting, Enable, Trigger, Time, Measurement, Security and Telephony Applications	9.2K
2	Lighting, Enable and Trigger Applications	8.8K
2	Lighting, Enable, Trigger, Time, Measurement, Security and Telephony Applications	17.9K
3	Lighting, Enable and Trigger Applications	12.3K
3	Lighting, Enable, Trigger, Time, Measurement, Security and Telephony Applications	21.5K
4	Lighting, Enable and Trigger Applications	12.3K
4	Lighting, Enable, Trigger, Time, Measurement, Security and Telephony Applications	21.5K

C-Bus Module Interface Specification

21.2 Transparent Mode

In Transparent Mode, the C-Bus module passes data directly between the PCI and a second serial port. This provides direct access to C-Bus if needed. This is typically used where there is a device that contains a PCI chip (such as C-Touch, CBTI etc), and a PC wants to use this device as if it were a PCI.

**21.2.1 Entering Transparent Mode**

To enter or exit Transparent Mode, use the `cbus_vf_set_transparent_mode` function.

The standard method for resetting a C-Bus PCI is to send the string `~~~<CR>` it, and this sequence is used by all current Clipsal software packages. If a C-Bus device sees this string, it can assume that a software package is expecting it to behave as a PCI, and set transparent mode.

21.2.2 Registering Event Handlers

The event handlers to transmit data from the C-Bus module to the PCI and the second serial port need to be registered. For example:

```

void transmit_cbus_character(char transmit_character)
{
    /* put code in here to transmit the character to the
       C-Bus PCI Chip */
}

void transmit_serial_character(char transmit_character)
{
    /* put code in here to transmit the character to the
       second serial port */
}

void main(void)
{
    /* register the event handlers */

    cbus_vf_register_serial_transmit_handler(transmit_cbus_character);
    cbus_vf_register_serial_transmit_handler2(
        transmit_serial_character);
}
  
```

C-Bus Module Interface Specification

21.2.3 Transmit Data to C-Bus

To Transmit Data to C-Bus:

- Call `cbus_vf_serial_receive_character2` to send each character
- The character will be sent to the function registered to handle data transmission to the PCI (`transmit_cbus_character` in the example above).

21.2.4 Receive Data From C-Bus

To Receive Data from C-Bus:

- Call `cbus_vf_serial_receive_character` for each character received from the PCI
- The character will be sent to the function registered to handle data transmission to the PC (`transmit_serial_character` in the example above).

C-Bus Module Interface Specification

21.3 Use With Other C-Bus Applications

The `cbus_config.h` file needs to be edited to enable the Applications that are required, and disable the rest.

Applications that are not required are removed from the definitions, usually by commenting out. Any source code file for an application that is not used can optionally be deleted.

Use is the same as the lighting case, except that the number of lighting applications and groups is not relevant for non-lighting application support.

21.4 Programming Configuration Data**21.4.1 PCI Configuration Data**

If the PCI chip configuration data is changed via C-Bus, the Configuration Change Event Handler is called if it has been registered. It is up to the user application to read any PCI configuration data of interest to determine what has been changed.

21.4.2 C-Bus Device Specific Configuration Data

When a C-Bus device is programmed using its corresponding Graphical User Interface (GUI), a series of CAL commands will be sent across C-Bus. These will be interpreted by the C-Bus Module and passed onto the user application as a series of events.

The most common events are the Write Event and Read Event. If event handlers are registered for these, they will be called by the C-Bus Module.

The Write Event Handler is responsible for writing the received data to the appropriate address.

The Read Event Handler has to read the required number of bytes from the appropriate address, and send the data back using `cbus_vf_CAL_read_response`.

For details of the other, more specialised, functions, refer to section 12.

21.4.2.1 Virtual Memory Addresses

The document CBUS-VMA "C-Bus System Virtual Addressing for Compound Devices" describes the methods for writing to and reading from compound C-Bus devices, and describes the rationale of the addressing scheme used.

21.4.2.2 Writing the Address Pointer

When a C-Bus STORE command is sent to the C-Bus device at Virtual Address 00 (the Address Pointer), the address is stored within the C-Bus Module, and an ACKNOWLEDGE is sent back to the Programming Device. No event is generated.

C-Bus Module Interface Specification

21.4.2.3 *Writing to the C-Bus Device*

When a C-Bus STORE command is sent to the C-Bus device at Virtual Address 01 (the read / write address), the Write Event Handler (see 12.3.1.4) will be called with the write address (the Address Pointer) and the data.

When the user returns control to the C-Bus Module, an ACKNOWLEDGE will be sent to originator (the programming unit) if the result of the event handler is `cbus_true`.

21.4.2.4 *Reading from the C-Bus Device*

When a C-Bus RECALL command is sent to the C-Bus device to Virtual Address 01 (the read / write address), the Read Event Handler (see 12.3.1.5) will be called with the read address (the Address Pointer) and the number of bytes to be read.

The user needs to copy the read data to the `result` parameter. When the user returns control to the C-Bus Module, a REPLY will be sent to the originator (the programming or interrogating unit) containing the read data if the result of the event handler is `cbus_true`.

21.4.2.5 *Using Other Virtual Addresses*

When a C-Bus STORE command is sent to the C-Bus device to Virtual Address 02 - FE, the Generic Write Event Handler (see 12.3.1.2) is called with the details of the source of the command, the destination of the command (ie. Virtual Address), and the data.

The format of the data is up to the user to determine. When the user returns control to the C-Bus Module, an ACKNOWLEDGE will be sent to the Programming Device if the result of the event handler is `cbus_true`.

When a C-Bus RECALL command is sent to the C-Bus device to Virtual Address 01 (the read / write address), the Generic Read Event Handler (see 12.3.1.3) is called with the details of the source of the command, the destination of the command (ie. Virtual Address), and the number of bytes to be read.

The user needs to copy the read data to the `result` parameter. When the user returns control to the C-Bus Module, a REPLY will be sent to the Programming Device containing the read data if the result of the event handler is `cbus_true`.

Use of this method for reading and writing configuration data is strongly discouraged.

By convention, writing to virtual address FF is used for execution of some kind of command. The exact nature and purpose of commands (if any) is at the discretion of the equipment designer.

21.4.2.6 *Other CAL Commands*

Any other CAL commands will cause the Generic Programming Event Handler (see 12.3.1.1) to be called (if registered).

C-Bus Module Interface Specification

21.4.2.7 Examples

In the following examples, the C-Bus Device unit address is CC. A Programming Device is controlling the C-Bus Device.

21.4.2.7.1 Set Address Pointer

To set the Address Pointer to \$12345678, a STORE command is sent to virtual address 00:

```
\06CC0900A6000078563412
```

There will be no events raised.

21.4.2.7.2 Store Data

To store data \$AABB to the address pointed to by the Address Pointer (\$12345678 in the example above), a STORE command is sent to virtual address 01:

```
\06CC0900A40100AABB
```

This will result in the Write Event Handler being called with the following data:

source_network: depends on the source and the networks configuration (see section 7).

source_unit_address: 0

address: \$12345678

data: pointer to array {0xAA, 0xBB}

data_count: 2

This data should be stored, as appropriate, and the function return value should be set to **cbus_true** so that the C-Bus Module sends an ACKNOWLEDGE to the programming device.

21.4.2.7.3 Read Data

To read 2 bytes of data from address \$12345678, it will be necessary to reset the Address Pointer first as described above. Then a RECALL command is sent to virtual address 01:

```
\06CC09001A0102
```

This will result in the Read Event Handler being called with the following data:

source_network: depends on the source and the networks configuration (see section 7).

source_unit_address: 0

address: \$12345678

result: pointer to an array of bytes to store the read result

data_count: 2

C-Bus Module Interface Specification

The user then needs to read the data and copy it to the `result` parameter. The function return value should be set to `cbus_true` so that the C-Bus Module sends a REPLY to the programming device.

21.4.2.7.4 Execute Command

To send a command \$5245534554 (ASCII for "RESET") to the C-Bus Device, a STORE command is sent to virtual address \$FF (Control):

\06CC0900A7FF005245534554

This will result in the Generic Write Event Handler being called with the following data :

`source_network`: depends on the source and the networks configuration (see 7).

`source_unit_address`: 0

`virtual address`: \$FF

`command`: pointer to string "5245534554"

`command_length`: 5

The user needs to interpret the string "5245534554" and act on it. The function return value should be set to `cbus_true` so that the C-Bus Module sends an ACKNOWLEDGE to the programming device.

C-Bus Module Interface Specification

21.5 Creating a System with Non-Volatile Behaviour

To create a system with non-volatile behaviour (ie. it restores its state on power-up to how it was before the power-failure), it is necessary to save the values of the C-Bus databases to non-volatile memory (eg EEPROM, FLASH, FRAM, battery backed-up SRAM etc).

A typical way to implement the storing of the data is:

- When a Lighting Database Change Event is received (see section 13.7.1), save the new level.
- When a Security Application Zone Status Event is received (see section 18.3.2.5), save the state.
- When a Time event is received (see section 16.4), update the Real Time Clock (if one is available).
- Store any other data of interest on power-failure (assuming you have time).

When power is applied, the reverse process is applied:

- Read the stored Lighting Database levels and set them in the C-Bus module (see section 13.2.3).
- Read the Security Zone status and set them in the C-Bus Module (see section 0).
- Read the time and set the C-Bus Module time (see section 16.1).
- Restore any other data of interest.

C-Bus Module Interface Specification

22 EXAMPLE APPLICATION**22.1 Introduction**

In the example below, a device has an LCD Display and a connection to C-Bus to enable it to perform some simple functions:

- If the Control Group 1 is set to level 0xFF, a lighting scene is triggered (ie. several Lighting groups are set to pre-defined levels)
- If the telephone rings, the number is displayed and a light is switched on
- If the Security status changes, the details are displayed
- If the Security alarm is triggered, a light is switched on
- The outside temperature is displayed

The device also has EEPROM memory that is used to store configuration data. The configuration data can be accessed via C-Bus. The memory map is shown below:

Address	Usage
0	Number of Groups in the scene
1	Scene Group Address 1
2	Scene Level 1
3	Scene Group Address 2
4	Scene Level 2
5	Scene Group Address 3
6	Scene Level 3
7	Scene Group Address 4
8	Scene Level 4
9	Scene Group Address 5
10	Scene Level 5
11	Scene Group Address 6
12	Scene Level 6
13	Telephone Light Group Address
14	Security Light Group Address

22.2 Code

The initialisation function is called following power-up. The various event handlers will be called automatically.

The functions that write to the serial port, write to the LCD display and read/write the EEPROM are not shown.

C-Bus Module Interface Specification

```
/*
** Initialise the C-Bus Module and register call-back functions
*/
void initialisation(void)
{
    /* Register the handler to write a character to the C-Bus PCI */
    cbus_vf_register_serial_transmit_handler(
        transmit_cbus_character);

    /* Register the second network */
    cbus_bf_register_network_path(1, "A2");

    /* Register the Trigger Control event handler */
    cbus_trigger_vf_register_handler(trigger_event_handler);

    /* Register the Telephone Ringing event handler */
    cbus_telephony_vf_ringing_handler(
        telephony_ringing_event_handler);

    /* Register the Security Status event handler */
    cbus_security_vf_status_message_handler(
        security_status_handler);

    /* Register the Measurement event handler */
    cbus_measurement_vf_register_handler(measurement_handler);

    /* Register the CAL event handlers for programming EEPROM */
    cbus_vf_CAL_register_write_handler(CAL_write_event_handler);
    cbus_vf_CAL_register_read_handler(CAL_read_event_handler);

    /* Initialise the C-Bus Module */
    cbus_bf_initialise();
    while (cbus_ef_get_connect_state() !=
        cbus_ce_module_normal_operation)
    {
        delay(1);
    }
}

/*
** Send a character of data to the C-Bus PCI via serial port 3
*/
void transmit_cbus_character(char transmit_character)
{
    send_serial_character(PORT3, char);
}

/*
** If Trigger Group 1 is set to level 0xFF, then switch on
** some lights
*/
void trigger_event_handler(int8u group,
```

C-Bus Module Interface Specification

```

                                int8u level)
{
    int i;
    if ((group == 1) && (level == 0xFF))
    {
        /* read the Group Addresses and Levels from EEPROM and
           set the scene */
        for (i=0; i<get_EEPROM(0); i++)
            cbus_lighting_vf_set_level(0, 0x38, get_EEPROM(i*2+1),
                                       get_EEPROM(i*2+2), 1, 0, 0);
    }
}

/*
** If the telephone rings, display the number on the second
** line of the LCD display and switch on a light
*/
void telephony_ringing_event_handler(char * number)
{
    char s[50];
    strcpy(s, "Telephone number : ");
    strcat(s, number);
    display_message(2, s);
    cbus_lighting_vf_set_level(0,
                               0x38,
                               get_EEPROM(13),
                               255,
                               1,
                               0,
                               0);
}

/*
** If a Security Alarm status message is received, display
** a message on the second line of the LCD display.
** If the Alarm is on, switch a light on.
*/
void security_status_handler(
    cbus_security_et_status_message_type message_type,
    cbus_boolean status)
{
    char s[80];
    switch (message_type)
    {
        case cbus_security_ce_status_alarm :
            strcpy(s, "Alarm");
            break;
        case cbus_security_ce_status_tamper :
            strcpy(s, "Tamper");
            break;
        case cbus_security_ce_status_panic :
            strcpy(s, "Panic");
            break;
        case cbus_security_ce_status_low_battery :
            strcpy(s, "Low Battery");
    }
}

```

C-Bus Module Interface Specification

```

        break;
    case cbus_security_ce_status_mains_failure :
        strcpy(s, "Main Failure");
        break;
    case cbus_security_ce_status_line_cut :
        strcpy(s, "Line Cut");
        break;
    case cbus_security_ce_status_arm_failed :
        strcpy(s, "Arm Failed");
        break;
    case cbus_security_ce_status_fire_alarm :
        strcpy(s, "Fire Alarm");
        break;
    case cbus_security_ce_status_gas_alarm :
        strcpy(s, "Gas Alarm");
        break;
    case cbus_security_ce_status_other_alarm :
        strcpy(s, "Other Alarm");
        break;
    }

    /* build command string */
    if (status == cbus_true)
        strcat(s, " ON");
    else
        strcat(s, " OFF");
    display_message(2, s);

    /*
    ** If the alarm is on, switch on the light on the remote
   ** network
    */
    if (cbus_security_if_get_status(cbus_security_ce_status_alarm))
        cbus_lighting_vf_set_level(1,
                                   0x38,
                                   get_EEPROM(14),
                                   255,
                                   0,
                                   0,
                                   0);
}

/*
** Display the outside air temperature on line 1 of the LCD.
** The temperature is broadcast on device 1, channel 3
*/
void measurement_handler(int8u device_id,
                        int8u channel,
                        int32s mantissa,
                        int8s exponent,
                        cbus_measurement_et_unit_type units)
{
    char s[50];
    if ((device_id == 1) && (channel == 3))
    {
        strcpy(s, "Outside Temperature ");
    }

```

C-Bus Module Interface Specification

```
        cbus_str_add_integer_to_string(s, (long int)mantissa, 0);
        strcat(s, "C");
        display_message(1, s);
    }
}

/*
** Allow the user to write to the EEPROM via C-Bus
*/
cbus_boolean CAL_write_event_handler(int8u source_network,
                                     int8u source_unit_address,
                                     int32u address,
                                     int8u * data,
                                     int8u data_count)
{
    int8u i;
    if (address + data_count <= 16)
    {
        for (i=0; i<data_count; i++)
            set_EEPROM(address++, *data++);
        return cbus_true;
    }
    else
        return cbus_false;
}

/*
** Allow the user to read from the EEPROM via C-Bus
*/
cbus_boolean CAL_read_event_handler(int8u source_network,
                                    int8u source_unit_address,
                                    int32u address,
                                    int8u * result,
                                    int8u data_count)
{
    int8u i;
    if (address + data_count <= 16)
    {
        for (i=0; i<data_count; i++)
            *result++ = get_EEPROM(address++);
        return cbus_true;
    }
    else
        return cbus_false;
}
```