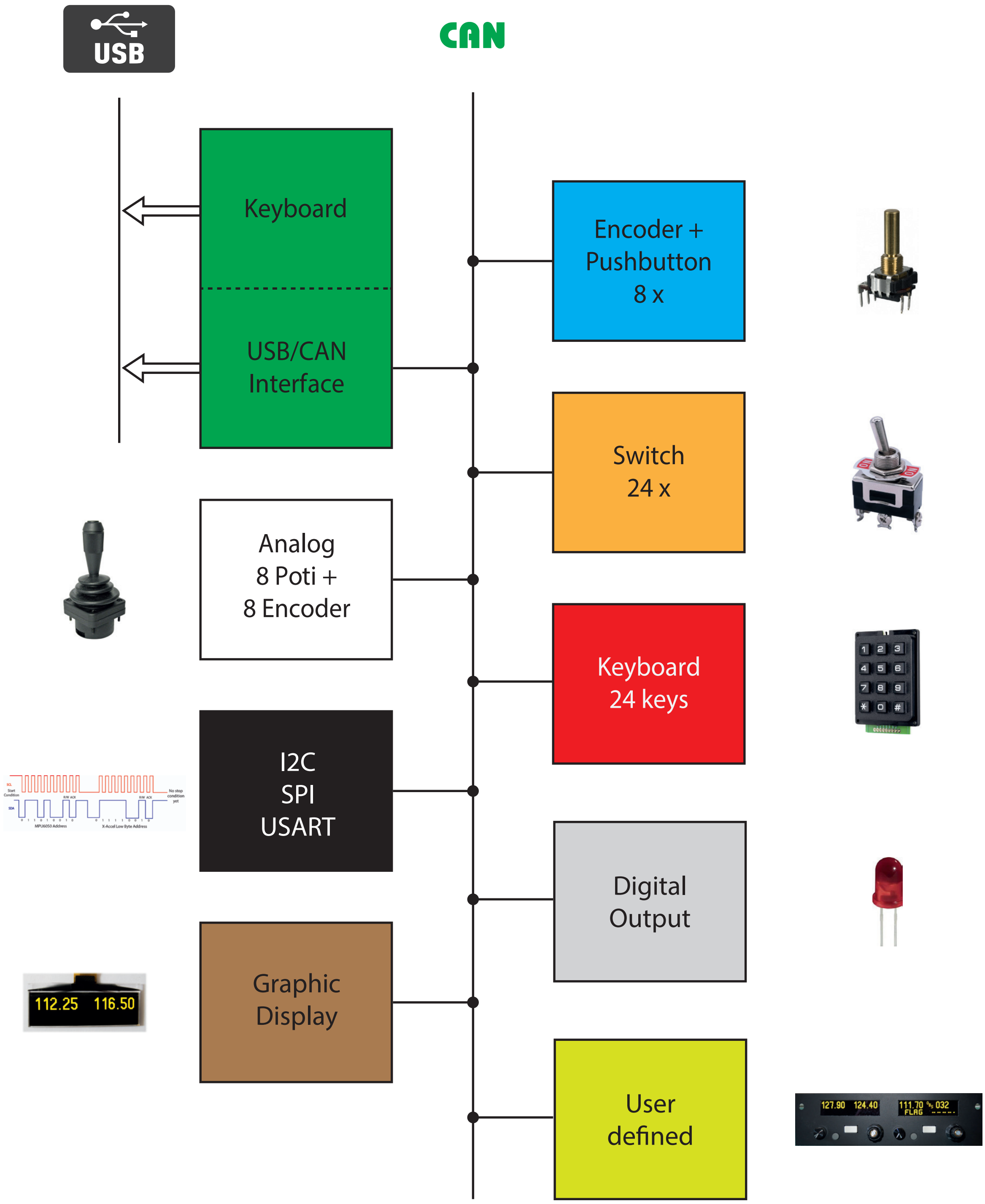
CAN in Simulation

Rev. 1.0

Overview


USB
CAN
Keyboard
USB/CAN
Interface
Encoder +
Pushbutton
8 x
Switch
24 x
Analog
8 Poti +
8 Encoder
Keyboard
24 keys
I2C
SPI
USART
Digital
Output
Graphic
Display
112.25 116.50
User
defined
127.90 124.40
111.70 032

1. CAN

Controller Area Network (CAN) is a widely used communication protocol and bus system designed for robust and reliable data exchange between electronic devices or nodes in various applications, including automotive, industrial, and aerospace sectors.

CAN was originally developed by Robert Bosch GmbH in the 1980s as a solution to address the increasing complexity of electrical systems in vehicles. It has since become an industry standard and is defined by the ISO 11898 standard.

The key features and characteristics of CAN include:

- Message-based communication: CAN uses a message-oriented communication model, where devices exchange data in the form of messages. Each message contains an identifier, data, and other control information.
- Broadcast communication: CAN uses a broadcast mechanism, where messages are sent on the bus and received by all connected devices. Each device filters and processes only the messages relevant to its own functionality, based on the message identifier.
- Deterministic and prioritized communication: CAN supports prioritization of messages through the use of message identifiers. Lower priority messages yield to higher priority messages, ensuring time-critical and important data can be transmitted with minimal delay.
- Error detection and fault tolerance: CAN incorporates a robust error detection and fault tolerance mechanism. It uses a cyclic redundancy check (CRC) to verify the integrity of transmitted data and includes error detection and signaling mechanisms to handle bus errors and recover from faults.
- Differential signaling: CAN uses differential signaling, which helps to ensure noise immunity and reliable data transmission, even in noisy environments.

CAN has gained widespread adoption due to its reliability, simplicity, and efficiency. It is particularly well-suited for applications that require real-time communication, fault tolerance, and robustness, such as automotive control systems, industrial automation, and aerospace avionics.

2. CANaerospace

CANaerospace is a cooperative initiative among avionics manufacturers, operators, and integrators aimed at defining a standardized data bus for avionics systems. It is an open architecture and protocol that allows for seamless communication between avionics devices within an aircraft.

The CANaerospace protocol is based on the Controller Area Network (CAN) bus technology, which is known for its reliability and fault-tolerant capabilities. It provides a structured framework for transmitting and receiving data between avionics systems, such as flight control, navigation, communication, and monitoring systems.

The main goal of CANaerospace is to enable interoperability and compatibility among different avionics components from various manufacturers. By defining a common protocol and message format, CANaerospace facilitates the integration and exchange of data between avionics systems, regardless of their origin.

This standardized approach offers several benefits, including reduced development time, improved system reliability, simplified maintenance, and increased flexibility in avionics upgrades and modifications. It also helps to minimize wiring complexity and weight, which are crucial considerations in the aerospace industry.

CANaerospace has been widely adopted in both commercial and military aircraft, as well as in other aerospace applications. It continues to evolve and adapt to meet the changing needs and advancements in avionics technology, ensuring efficient and reliable communication between avionics systems in modern aircraft.

3. CAN in Simulation

CAN in Simulation (CiS) is a technology that allows the integration of a Controller Area Network (CAN) bus system with a Flight Simulator running on a PC. This integration is achieved using a hardware component called the CAN-USB Interface, which connects the CAN bus to the PC via a USB port. The CAN-USB Interface is a Human Interface Device (HID) and does not require additional drivers. It converts CAN messages into a 15-byte report format and vice versa.

Flight simulators that support the SimConnect API, such as Microsoft Flight Simulator, Prepar3D, and Flight Simulator X, can benefit from AxisAndOhs software, which is highly versatile and can directly interpret CiS messages. For X-Plane users, FlyWithLua software may be preferred for this purpose.

Apart from the CAN-USB interface, several basic CAN bus devices are available for connecting input and output components. Electronic Control Units (ECUs) or CAN nodes can be used to connect potentiometers, encoders, and switches as input components. On the output side, there are devices that enable connection to alphanumeric or graphical displays, indicators, or actuators.

These CAN bus devices offer a flexible and modular approach for integrating a wide range of components with the Flight Simulator. By using these devices, custom solutions tailored to specific needs and requirements can be created. Whether operating a complete flight deck, navigation equipment, or displaying information, there are CAN bus devices available to meet your needs. Alternatively, custom CAN bus devices can be built, greatly enhancing the realism and functionality of a Flight Simulator setup.

According to the CANaerospace specification, *CAN in Simulation* uses CAN identifiers in the „Low Priority User Defined Data“ range of **0x708 - 0x76B** (1800 -1899). There are 6 distinct groups predefined, each comprising eight unique identifiers:

hexadecimal	decimal	
0x708 - 0x70F	1800 - 1807	Encoder Data
0x710 - 0x717	1808 - 1815	Switch Data
0x718 - 0x71F	1816 - 1823	Analog Data
0x720 - 0x727	1824 - 1831	Keyboard Data
0x728 - 0x72F	1832 - 1839	Digital Output Data
0x730 - 0x73F	1840 - 1847	User Defined Data

The meaning of the 8 data bytes in the corresponding CAN message is explained in the subsequent descriptions.

From the Low Priority Node Service Data (NSL) *Can In Simulation* implements the following services:

- Identification Service (IDS)
- Node Synchronisation Service (NSS)
- State Transmission service (STS)
- Baud Setting Service (BSS)
- Node-ID Setting Service (NIS)
- Module Information Service (MIS)
- Module Configuration Service (MCS)
- CAN Identifier Setting Service (CSS)

Can In Simulation follows the CANaerospace general message format with a data field like this:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

Byte 0:	node-ID
Byte 1:	data type
Byte 2:	service code
Byte 3:	message code
Byte 4 - 7:	4 data bytes

node-ID	The node-ID serves as the identifier for the module either transmitting data in Normal Operation Data (NOD) or the intended recipient module in Node Service Data (NSH/NSL). Node-IDs span from 0 to 255, where <0> carries a unique status as a special broadcast address, directing communication to „all nodes.“

data type	This indicates the coding of the message data, with a comprehensive list of available data types provided in Appendix C.

service code	In the context of Normal Operating Data (NOD), this byte offers a means to delineate various data interpretations. It should be set to zero when not in use. In the case of Node Service Data (NSH/NSL), this represents the pre-defined service code for the ongoing operation.

message code	In the context of Normal Operating Data (NOD), this byte is sequentially incremented with each message, signifying consecutive messages. When it reaches <255>, it wraps around to <0>. Within Node Service Data (NSH/NSL), this byte serves as a means to extend the service‘s specifications.

data	This represents the actual data. The number of significant bytes is determined by the code in the *data type* field.

3.1 Low Priority User Defined Data (UDL)

CAN in Simulation uses the UDL range of CANaerospace identifiers (0708h ... 076Bh). Messages are uniquely identified by their CAN-ID, and the message data are specified through the message header.

3.1.1 Encoder Data (CAN-ID 708h to 70Fh)

Encoder modules transmit a message upon detecting any encoder event, whether it be a clockwise or anti-clockwise turn, or the pressing or releasing of a pushbutton. These events are bit-oriented and efficiently packed into 1 byte.

CAN - ID	Message Header				Message Data			
708h	node	11	item	num	data	0	0	0
|								
70Fh	node	11	item	num	data	0	0	0

node-ID:	node ID (node = 1 ... 255)
data type:	BCHAR (11)
service code:	encoder number (item = 1 ... 255)
message code:	incremental message count (num++)
message data:	Byte 4: Bit 7: fast rotation Bit 3: event PUSHBUTTON OFF Bit 2: event PUSHBUTTON ON Bit 1: event COUNTERCLOCKWISE Bit 0: event CLOCKWISE

3.1.2 Switch Data (CAN-ID 710h to 717h)

Switch modules produce a message whenever there is a change in the state of any connected switches. The data is organized in a bit-oriented manner and compactly packed into a single byte.

CAN - ID	Message Header				Message Data			
710h	node	11	item	num	data	0	0	0
|								
717h	node	11	item	num	data	0	0	0

node-ID:	node ID (1 ... 255)
data type:	BCHAR (11)
service code:	switch number (item = 1 ... 255)
message code:	incremental message count (num++)
message data:	Byte 4: Bit 1: event ON Bit 0: event OFF

3.1.3 Analog Data

Analog modules generate a 16-bit output, and this data is packed into 2 bytes, with the most significant byte presented first.

CAN - ID	Message Header				Message Data			
718h	node	7	item	num	a_1	a_2	0	0
|								
71Fh	node	7	item	num	a_1	a_2	0	0

node-ID:	node ID (1 .. 255)
data type:	USHORT (7)
service code:	axis number (item = 1 ... 255)
message code:	incremental message count (num++)
message data:	Byte 4: analog value (a_1 = high byte) Byte 5: analog value (a_2 = low byte)

3.1.4 Keyboard Data

Keyboard modules transmit a message each time an associated key is either pressed or released. The key message comprises a modifier byte and a key-code byte, both defined in the USB HID usage tables published by the USB Implementers Forum (Appendix A).

CAN - ID	Message Header				Message Data			
720h	node	19	0	num	mod	key	0	0
|								
727h	node	19	0	num	mod	key	0	0

node-ID:	node ID (node = 1 ... 255)
data type:	UCHAR2 (19)
service code:	-
message code:	incremental message count (num++)
message data:	Byte 4: Modifier (mod) Byte 5: Keycode (key) (see Appendix A)

Modifier byte:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
left Ctrl	left Shift	left Alt	left GUI	right Ctrl	right hift	right Alt	right GUI

3.1.5 Digital Output Data

Digital output modules function as receiving units, accepting messages to either set or clear a designated output port.

CAN - ID	Message Header				Message Data			
728h	node	11	item	num	data	0	0	0
|								
72Fh	node	11	item	num	data	0	0	0

node-ID:	node ID (node = 1 ... 255)
data type:	BCHAR (11)
service code:	output channel (item = 1 ... 255)
message code:	incremental message count (num++)
message data:	Byte 4: Bit 1: set OFF Bit 0: set ON

3.1.6 User-defined Data

User-defined data data refer to information not explicitly specified. All 8 message bytes (header and data) can be utilized for transmitting arbitrary data, although adhering to the CANaerospace scheme is strongly recommended.

When utilizing the AxisAndOhs API, this interface processes messages falling within the specified CAN-ID range and generates (or updates) a local variable. This variable can then be utilized in scripts as needed.

CAN - ID	Message Header				Message Data			
730h	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
|								
73Fh	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8

node-ID:	user specified	AAO: unused
data type:	user specified	AAO: data type, part of L:var name
service code:	user specified	AAO: part of L:var name
message code:	user specified	AAO: part of L:var name
message data:	user specified	AAO: L:var value based on data type

3.2 Node Service Data (NSH)

3.2.1 Identification Service (IDS)

The Identification Service operates as a client/server-type service, serving the purpose of acquiring a „sign-of-life“ indication from the specified node.

If the node-ID is configured as 0 (the broadcast address), it enables the detection of all nodes connected to the network.

Identification Service

CAN - ID	Message Header				Message Data			
7D0h	node	0	0	num	0	0	0	0

node-ID:	node ID (node = 0 ... 255)
data type:	NODATA (0)
service code:	IDS (0)
message code:	incremental message count (num++)
message data:	unused

Response

CAN - ID	Message Header				Message Data			
7D1h	node	16	0	<>	xx	yy	0	0

node-ID:	node ID (node = 1 ... 255)
data type:	UCHAR4 (16)
service code:	IDS (0)
message code:	<as in request>
message data:	Byte 4: Hardware Revision (xx) Byte 5: Software Revision (yy) Byte 6: Identifier Distribution (0 = default) Byte 7: Header Type (0 = CANaerospace)

3.2.2 Node Synchronisation Service (NSS)

This service is employed to synchronize the time across all nodes connected to the network. For this purpose, the node-ID is configured as 0. The timestamp can be used to provide a 32-bit value for adjusting clock settings.

Node Synchronisation Service

CAN - ID	Message Header				Message Data			
080h	0	4	1	0	time3	time2	time1	time0

node-ID:	node ID (node = 0, broadcast)
data type:	ULONG (4)
service code:	NSS (1)
message code:	unused (0)
message data:	Byte 4 - 7: 32-bit timestamp (time3 = MSB)

There is no response.

3.2.3 State Transmission Service (STS)

This service is used by other nodes in the network which need to to obtain current data that is normally transmitted upon state change only.

State Transmission Service

CAN - ID	Message Header				Message Data			
7D0h	node	0	7	0	0	0	0	0

node-ID: node ID (node = 1 ... 255)
data type: NODATA (0)
service code: STS (7)
message code: 0
message data: unused

Response

CAN - ID	Message Header				Message Data			
7D1h	node	<>	0	0	<>	<>	<>	<>

node-ID: node ID (node = 1 ... 255)
data type: CiS: data specific
service code: STS (7)
message code: CiS:
message data: CiS: module states

Can in Simulation modules, such as the switch module or digital output modules, can be queried using the State Transmission Service to gather information about current switch settings or output states. The returned data are bit-oriented and compactly packed into 3 bytes, accommodating 24 switches and 24 output ports.

3.2.4 Baudrate Setting Service (BSS)

The Baudrate Setting Service alters the CAN baudrate of the addressed node. Since the baudrate change takes effect immediately, there is no response in the event of success. If the change fails, the node retains its original baudrate and issues an error.

Baudrate Setting Service

CAN - ID	Message Header				Message Data			
7D0h	node	6	10	0	b_1	b_2	0	0

node-ID: node ID (node = 0 ... 255)
data type: SHORT (6)
service code: BSS (10)
message code: 0
message data: Byte 4: b_1 = baudrate code (high byte)
Byte 5: b_2 = baudrate code (low byte)

(Response)

CAN - ID	Message Header				Message Data			
7D1h	node	0	10	-1	0	0	0	0

node-ID: node ID (node = 1 ... 255)
data type: NODATA (0)
service code: BSS (10)
message code: -1 = baudrate code unknown
message data: n. a.

The baudrate code may have the following values:

0	1 MBit/s
1	500 kBit/s
2	250 kBit/s
3	125 kBit/s

3.2.5 Node-ID Setting Service (NIS)

The Node-ID Setting Service is employed to configure the ID of the addressed node, and the new ID takes effect immediately. Upon successful execution, the response includes the updated node ID.

Identification Service

CAN - ID	Message Header				Message Data			
7D0h	node	0	11	id	0	0	0	0

node-ID:	node ID (node = 0 ... 255)
data type:	NODATA (0)
service code:	NIS (11)
message code:	<new node-ID> $(1 \leq id \leq 255)$
message data:	unused

Response

CAN - ID	Message Header				Message Data			
7D1h	node	0	11	0	0	0	0	0

node-ID:	node ID (node = 1 ... 255)
data type:	NODATA (0)
service code:	NIS (11)
message code:	0 = ok
message data:	unused

3.2.6 Module Information Service (MIS)

The Module Information Service is a feature designed to provide details about modules installed in the addressed node. Consequently, users have the flexibility to define the format of this service. CAN in Simulation utilizes this service to fetch parameter values from the designated node.

Module Information Service

CAN - ID	Message Header				Message Data			
7D0h	node	0	12	c	0	0	0	0

node-ID: node ID (node = 1 ... 255)
data type: CiS: NODATA (0)
service code: MIS (12)
message code: CiS: c = code of parameter to be queried
message data: unused

Response

CAN - ID	Message Header				Message Data			
7D1h	node	<>	12	<>	p_1	p_2	p_3	p_4

node-ID: node ID (node = 1 ... 255)
data type: CiS: parameter specific
service code: MIS (12)
message code: <as request> or -6 if parameter code unknown
message data: CiS: requested parameter

Currently implemented parameter codes include:

0	CAN-ID
1	offset
2	threshold
3	slow step
4	fast step
5	keystroke
6	open drain
7	switch state

3.2.7 Module Configuration Service (MCS)

The module configuration service is designed to configure the modules installed within the specified node. As a result, the format of this service is primarily user-defined. CAN in Simulation utilizes this service to establish specific parameters employed by individual nodes.

Module Configuration Service

CAN - ID	Message Header				Message Data			
7D0h	node	<>	13	c	p_1	p_2	p_3	p_4

node-ID: node ID (node = 1 ... 255)
data type: CiS: parameter specific
service code: MCS (13)
message code: CiS: c = parameter code
message data: CiS: parameter value(s)

Response

CAN - ID	Message Header				Message Data			
7D1h	node	0	13	0	<>	<>	<>	<>

node-ID: node ID (node = 1 ... 255)
data type: NODATA (0)
service code: MCS (13)
message code: 0 = ok or -6 if parameter unknown or out of range
message data: <as in request>

3.2.8 CAN Identifier Setting Service (CSS)

The CAN identifier setting service is used to configure the CAN identifier for a designated CAN message transmitted by the targeted node. This is achieved by defining the message using a distinct „message number" along with the desired CAN identifier. The permissible range for both the message number and the CAN identifier is user-defined.

CAN ID Setting Service

CAN - ID	Message Header				Message Data			
7D0h	node	12	14	0	0	0	0	0

node-ID: node ID (node = 1 ... 255)
data type: SHORT2 (12)
service code: CSS (14)
message code:
message data: Byte 4: message number high byte
Byte 5: message number low byte
Byte 6: CAN identifier high byte
Byte 7: CAN identifier low byte

Response

CAN - ID	Message Header				Message Data			
7D1h	node	0	14	<>	0	0	0	0

node-ID: node ID (node = 1 ... 255)
data type: NODATA (0)
service code: CSS (14)
message code: 0 = OK, -6 = message number or CAN-ID out of range
message data: n. a.

Appendix A

Keycodes used by the keyboard module

hex	dec	Usage Name
00	0	*Reserved*
01	1	*Keyboard Error Roll Over*
02	2	*Keyboard POST Fail*
03	3	*Keyboard Error Undefined*
04	4	*Keyboard a and A*
05	5	*Keyboard b and B*
06	6	*Keyboard c and C*
07	7	*Keyboard d and D*
08	8	*Keyboard e and E*
09	9	*Keyboard f and F*
0A	10	*Keyboard g and G*
0B	11	*Keyboard h and H*
0C	12	*Keyboard i and I*
0D	13	*Keyboard j and J*
0E	14	*Keyboard k and K*
0F	15	*Keyboard l and L*
10	16	*Keyboard m and M*
11	17	*Keyboard n and N*
12	18	*Keyboard o and O*
13	19	*Keyboard p and P*
14	20	*Keyboard q and Q*
15	21	*Keyboard r and R*
16	22	*Keyboard s and S*
17	23	*Keyboard t and T*
18	24	*Keyboard u and U*
19	25	*Keyboard v and V*
1A	26	*Keyboard w and W*
1B	27	*Keyboard x and X*
1C	28	*Keyboard y and Y*
1D	29	*Keyboard z and Z*
1E	30	*Keyboard 1 and !*
1F	31	*Keyboard 2 and @*

hex	dec	Usage Name
20	32	*Keyboard 3 and #*
21	33	*Keyboard 4 and $*
22	34	*Keyboard 5 and %*
23	35	*Keyboard 6 and ^*
24	36	*Keyboard 7 and &*
25	37	*Keyboard 8 and **
26	38	*Keyboard 9 and (*
27	39	*Keyboard 0 and)*
28	40	*Keyboard Return (ENTER)*
29	41	*Keyboard ESCAPE*
2A	42	*Keyboard DELETE (Backspace)*
2B	43	*Keyboard Tab*
2C	44	*Keyboard Spacebar*
2D	45	*Keyboard - and (underscore)*
2E	46	*Keyboard = and +*
2F	47	*Keyboard [and {*
30	48	*Keyboard] and }*
31	49	*Keyboard \and |*
32	50	*Keyboard Non-US # and ˜*
33	51	*Keyboard ; and :*
34	52	*Keyboard ‘ and “*
35	53	*Keyboard ^ and ˜*
36	54	*Keyboard , and <*
37	55	*Keyboard . and >*
38	56	*Keyboard / and ?*
39	57	*Keyboard CapsLock*
3A	58	*Keyboard F1*
3B	59	*Keyboard F2*
3C	60	*Keyboard F3*
3D	61	*Keyboard F4*
3E	62	*Keyboard F5*
3F	63	*Keyboard F6*

Keycodes used by the keyboard module (continued)

hex	dec	Usage Name
40	64	*Keyboard F7*
41	65	*Keyboard F8*
42	66	*Keyboard F9*
43	67	*Keyboard F10*
44	68	*Keyboard F11*
45	69	*Keyboard F12*
46	70	*Keyboard PrintScreen*
47	71	*Keyboard ScrollLock*
48	72	*Keyboard Pause*
49	73	*Keyboard Insert*
4A	74	*Keyboard Home*
4B	75	*Keyboard PageUp*
4C	76	*Keyboard Delete Forward*
4D	77	*Keyboard End*
4E	78	*Keyboard PageDown*
4F	79	*Keyboard RightArrow*
50	80	*Keyboard LeftArrow*
51	81	*Keyboard DownArrow*
52	82	*Keyboard UpArrow*
53	83	*Keypad NumLock and Clear*
54	84	*Keypad /*
55	85	*Keypad **
56	86	*Keypad -*
57	87	*Keypad +*
58	88	*Keypad ENTER*
59	89	*Keypad 1 and End*
5A	90	*Keypad 2 and DownArrow*
5B	91	*Keypad 3 and PageDn*
5C	92	*Keypad 4 and LeftArrow*
5D	93	*Keypad 5*
5E	94	*Keypad 6 and RightArrow*
5F	95	*Keypad 7 and Home*

hex	dec	Usage Name
60	96	*Keypad 8 and UpArrow*
61	97	*Keypad 9 and PageUp*
62	98	*Keypad 0 and Insert*
63	99	*Keypad . and Delete*
64	100	*Keyboard Non-US \ and |*
65	101	*Keyboard Application*
66	102	*Keyboard Power*
67	103	*Keypad =*
68	104	*Keyboard F13*
69	105	*Keyboard F14*
6A	106	*Keyboard F15*
6B	107	*Keyboard F16*
6C	108	*Keyboard F17*
6D	109	*Keyboard F18*
6E	110	*Keyboard F19*
6F	111	*Keyboard F20*
70	112	*Keyboard F21*
71	113	*Keyboard F22*
72	114	*Keyboard F23*
73	115	*Keyboard F24*
74	116	*Keyboard Execute*
75	117	*Keyboard Help*
76	118	*Keyboard Menu*
77	119	*Keyboard Select*
78	120	*Keyboard Stop*
79	121	*Keyboard Again*
7A	122	*Keyboard Undo*
7B	123	*Keyboard Cut*
7C	124	*Keyboard Copy*
7D	125	*Keyboard Paste*
7E	126	*Keyboard Find*
7F	127	*Keyboard Mute*

Keycodes used by the keyboard module (continued)

hex	dec	Usage Name
80	128	*Keyboard VolumeUp*
81	129	*Keyboard VolumeDown*
82	130	*Keyboard Locking CapsLock*
83	131	*Keyboard Locking NumLock*
84	132	*Keyboard Locking ScrollLock*
85	133	*Keypad Comma*
86	134	*Keypad EqualSign*
87	135	*Keyboard International 1*
88	136	*Keyboard International 2*
89	137	*Keyboard International 3*
8A	138	*Keyboard International 4*
8B	139	*Keyboard International 5*
8C	140	*Keyboard International 6*
8D	141	*Keyboard International 7*
8E	142	*Keyboard International 8*
8F	143	*Keyboard International 9*
90	144	*Keyboard LANG 1*
91	145	*Keyboard LANG 2*
92	146	*Keyboard LANG 3*
93	147	*Keyboard LANG 4*
94	148	*Keyboard LANG 5*
95	149	*Keyboard LANG 6*
96	150	*Keyboard LANG 7*
97	151	*Keyboard LANG 8*
98	152	*Keyboard LANG 9*
99	153	*Keyboard Alternate Erase*
9A	154	*Keyboard SysReq/Attention*
9B	155	*Keyboard Cancel*
9C	156	*Keyboard Clear*
9D	157	*Keyboard Prior*
9E	158	*Keyboard Return*
9F	159	*Keyboard Separator*

hex	dec	Usage Name
A0	160	*Keyboard Out*
A1	161	*Keyboard Oper*
A2	162	*Keyboard Clear/Again*
A3	163	*Keyboard CrSel/Props*
A4	164	*Keyboard ExSel*
A5	165	*Reserved*
A6	166	*Reserved*
A7	167	*Reserved*
A8	168	*Reserved*
A9	169	*Reserved*
AA	170	*Reserved*
AB	171	*Reserved*
AC	172	*Reserved*
AD	173	*Reserved*
AE	174	*Reserved*
AF	175	*Reserved*
B0	176	*Keypad 00*
B1	177	*Keypad 000*
B2	178	*Thousands Separator*
B3	179	*Decimal Separator*
B4	180	*Currency Unit*
B5	181	*Currency Sub-unit*
B6	182	*Keypad (*
B7	183	*Keypad)*
B8	184	*Keypad {*
B9	185	*Keypad }*
BA	186	*Keypad Tab*
BB	187	*Keypad Backspace*
BC	188	*Keypad A*
BD	189	*Keypad B*
BE	190	*Keypad C*
BF	191	*Keypad D*

Keycodes used by the keyboard module (continued)

hex	*dec*	*Usage Name*
C0	192	*Keypad E*
C1	193	*Keypad F*
C2	194	*Keypad XOR*
C3	195	*Keypad ^*
C4	196	*Keypad %*
C5	197	*Keypad <*
C6	198	*Keypad >*
C7	199	*Keypad &*
C8	200	*Keypad &&*
C9	201	*Keypad |*
CA	202	*Keypad ||*
CB	203	*Keypad :*
CC	204	*Keypad #*
CD	205	*Keypad Space*
CE	206	*Keypad @*
CF	207	*Keypad !*
D0	208	*Keypad Memory Store*
D1	209	*Keypad Memory Recall*
D2	210	*Keypad Memory Clear*
D3	211	*Keypad Memory Add*
D4	212	*Keypad Memory Subtract*
D5	213	*Keypad Memory Multiply*
D6	214	*Keypad Memory Divide*
D7	215	*Keypad +/-*
D8	216	*Keypad Clear*
D9	217	*Keypad ClearEntry*
DA	218	*Keypad Binary*
DB	219	*Keypad Octal*
DC	220	*Keypad Decimal*
DD	221	*Keypad Hexadecimal*
DE	222	*Reserved*
DF	223	*Reserved*

hex	*dec*	*Usage Name*
E0	224	*Keyboard Left Control*
E1	225	*Keyboard Left Shift*
E2	226	*Keyboard Left Alt*
E3	227	*Keyboard Left GUI*
E4	228	*Keyboard Right Control*
E5	229	*Keyboard Right Shift*
E6	230	*Keyboard Right Alt*
E7	231	*Keyboard Right GUI*
E8	232	*Reserved*
...	...	*...*
FF	255	*Reserved*

Appendix B

1. USB Interface Input Report

Byte 0	*Report ID <1>*
Byte 1	*CAN ID (high byte)*
Byte 2	*CAN ID (low byte)*
Byte 3	*Node ID (CAN module identifier)*
Byte 4	*Data Type (as per CANaerospace specification)*
Byte 5	*Service Code (i. e. item address within module)*
Byte 6	*Message Count (incremented by one for each message)*
Byte 7	*Message Data Byte 0 (module dependent)*
Byte 8	*Message Data Byte 1 (module dependent)*
Byte 9	*Message Data Byte 2 (module dependent)*
Byte 10	*Message Data Byte 4 (module dependent)*
Byte 11	*Time Stamp [µs] (most significant byte)*
Byte 12	*Time Stamp [µs]*
Byte 13	*Time Stamp [µs]*
Byte 14	*Time Stamp [µs] (least significant byte)*

2. USB Interface Output Report

Byte 0	*Report ID <1>*
Byte 1	*CAN ID (high byte)*
Byte 2	*CAN ID (low byte)*
Byte 3	*Node ID (CAN module identifier)*
Byte 4	*Data Type (as per CANaerospace specification)*
Byte 5	*Service Code (i. e. item address within module)*
Byte 6	*Message Count (incremented by one for each message)*
Byte 7	*Message Data Byte 0 (module dependent)*
Byte 8	*Message Data Byte 1 (module dependent)*
Byte 9	*Message Data Byte 2 (module dependent)*
Byte 10	*Message Data Byte 4 (module dependent)*
Byte 11	*<0>*
Byte 12	*<0>*
Byte 13	*<0>*
Byte 14	*<0>*

Appendix C

Data Types (according to CANaerospace)

Data Type	*Range*	*Bits*	*Explanation*	*Type #*
NODATA	*n.a.*	*0*	*"No data" type*	0 (00h)
ERROR	*n.a.*	*32*	*Emergency event data type*	1 (01h)
FLOAT	*1-bit sign 23-bit fraction 8-bit exponent*	*32*	*Single precision floating-point value according to IEEE-754-1985*	2 (02h)
LONG	*-2147483647 to +2147483648*	*32*	*2's complement integer*	3 (03h)
ULONG	*0 to 4294967295*	*32*	*unsigned integer*	4 (04h)
BLONG	*n.a.*	*32*	*Each bit defines a discrete state. 32 bits are coded into four CAN data bytes*	5 (05h)
SHORT	*-32768 to +32767*	*16*	*2's complement short integer*	6 (06h)
USHORT	*0 to 65535*	*16*	*unsigned short integer*	7 (07h)
BSHORT	*n.a.*	*16*	*Each bit defines a discrete state. 16 bits are coded into two CAN data bytes*	8 (08h)
CHAR	*-128 to +127*	*8*	*2's complement char integer*	9 (09h)
UCHAR	*0 to 255*	*8*	*unsigned char integer*	10 (0Ah)
BCHAR	*n.a.*	*8*	*Each bit defines a discrete state. 8 bits are coded into a single CAN data byte*	11 (0Bh)
SHORT2	*-32768 to +32767*	*2x16*	*2 x 2's complement short integer*	12 (0Ch)
USHORT2	*0 to 65535*	*2x16*	*2 x unsigned short integer*	13 (0Dh)
BSHORT2	*n.a.*	*2x16*	*2 x discrete short*	14 (0Eh)
CHAR4	*-128 to +127*	*4x8*	*4 x 2's complement char integer*	15 (0Fh)
UCHAR4	*0 to 255*	*4x8*	*4 x unsigned char integer*	16 (10h)
BCHAR4	*n.a*	*4x8*	*4 x discrete char*	17 (11h)
CHAR2	*-128 to +127*	*2x8*	*2 x 2's complement char integer*	18 (12h)
UCHAR2	*0 to 255*	*2x8*	*2 x unsigned char integer*	19 (13h)
BCHAR2	*n.a.*	*2x8*	*2 x discrete char*	20 (14h)
MEMID	*0 to 4294967295*	*32*	*Memory ID for upload/download*	21 (15h)
CHKSUM	*0 to 4294967295*	*32*	*Checksum for upload/download*	22 (16h)
ACHAR	*0 to 255*	*8*	*ASCII character*	23 (17h)
ACHAR2	*0 to 255*	*2x8*	*2 x ASCII character*	24 (18h)
ACHAR4	*0 to 255*	*4x8*	*4 x ASCII character*	25 (19h)
CHAR3	*-128 to +127*	*3x8*	*3 x 2's complement char integer*	26 (1Ah)
UCHAR3	*0 to 255*	*3x8*	*3 x unsigned char integer*	27(1Bh)
BCHAR3	*n.a.*	*3x8*	*3 x discrete char*	28 (1Ch)
ACHAR3	*0 to 255*	*3x8*	*3 x ASCII character*	29 (1Dh)

Data Type	*Range*	*Bits*	*Explanation*	*Type #*
DOUBLEH	*1-bit sign 52-bit fraction 11-bit exponent*	*32*	*32 msb of double precision floating-point value according to IEEE-754-1985*	30 (1Eh)
DOUBLEL	*1-bit sign 52-bit fraction 11-bit exponent*	*32*	*32 lsb of double precision floating-point value according to IEEE-754-1985*	31 (1Fh)
RESVD	*n.a.*	-	*Reserved for future use*	32 - 99
UDEF	*n.a.*	-	*User defined data types*	100 - 255