



EBus is the Electronic Industries Association's (EIA) open standard IS-60 describing a method of communication between electronic products in the

home using five different media: power line, twisted pair, coax, broadcast RF, and infrared. A sixth medium, fiber optic, has a section left open and is undefined at this time.

CEBus is a complete, packet-oriented, peer-to-peer network using a Carrier Sense Multiple Access with Collision Detection and Collision Resolution (CSMA/CD) protocol. The CEBus standard defines everything needed up to and including the language used for application-to-application communication called the CEBus Common Application Language (CAL).

In this article, I'll introduce you to packet construction and show you how to create CAL messages that control a CEBus light switch. Hang on-or the details may swamp you!

### CEBUS AND CAL

The CEBus protocol is described using the OSI/ISO seven-layer model. CEBus uses four of those layers: application, network, data link, and physical. Note the actual application function (e.g., a light switch) is distinct from application layer protocol.

### WHY CAL?

The CEBus application language is a set of common language and data constructs created to enable

# CEBus for the Masses

interoperability between products used in residential automation. This interoperability is available between different manufacturers' products even without prior knowledge of the products.

For example, information to control Company X's light module or stereo is published by the EIA or the CEBus Industry Council (CIC). This information is known to the world without having to know anything specific regarding Company X's design implementation of how they use a class A amplifier to control a vacuum-encased, electrothermal photon emitter-otherwise known as a light bulb.

### PACKET STRUCTURE

A CEBus packet frame can be broken down into several parts: the Link Protocol Data Unit (LPDU), the Network Protocol Data Unit (NPDU), the Application Protocol Data Unit (APDU), and the CAL message. I describe these different parts using a mailed letter (see Figure 1) as an example.

Figure 2 shows a breakdown of a packet structure illustrating the different parts.

### LPDUHEADER

The LPDU header contains the control field and the source and destination addresses. In the letter mailing scenario, the control field represents the postal service used to send the letter. The control field specifies the packet type,

### PETER HOUSE

Picking up where other CEBus articles in **///K left** off, Peter introduces us to packet construction and CAL messages. By the end, you'll be able to control a real-live CEBus light switch!

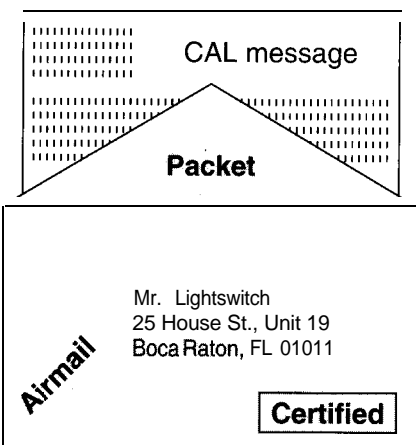


Figure 1: A letter illustrates the various services available to CEBus packets.



packet priority, and service class to the Data Link Layer (DLL). Figure 3 shows a bit-oriented breakdown of the control field.

The packet type is used to select the form of DLL service. This method roughly corresponds to sending a letter normal mail or with a return receipt requested. The DLL handles all channel acquisition, timing, and packet-receipt verification.

There are two classes of DLL service: acknowledged and unacknowledged. Acknowledged services expect a response from the receiving nodes DLL and unacknowledged services do not. DLL packet types include immediate acknowledge (IACK), acknowledged data (ACK\_DATA), unacknowledged data (UNACK\_DATA), failure (FAILURE), addressed acknowledge data (ADDR\_ACK\_DATA), addressed immediate acknowledge (ADDR\_IACK), and addressed unacknowledged data (ADDR\_UNACK\_DATA).

Once a node acquires the channel, the response from the receiving node is considered part of the acquisition. The acknowledge packet must start within 200 μs after the end of receiving a packet from the requesting node. After the DLL receives the transmit request from the application, the DLL automatically handles all of the retries and channel acquisition.

The ACK\_DATA services' acknowledge is an ultrashort packet with only an NPDU header and a null information field. The acknowledge packet's control field contains either FAILURE or IACK packet type. The destination and source address fields must be null. IACK signifies to the

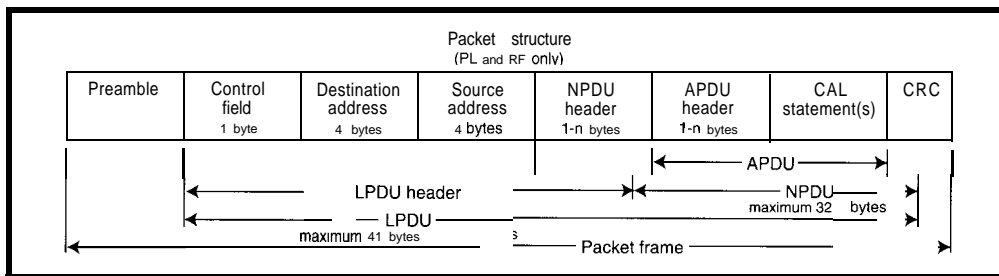


Figure 2: The elements of a CEBus packet are broken down into logical groups with size information

transmitting DLL the proper receipt of the packet. FAILURE signifies that the receiving node's DLL is operational but could not pass the packet to its network layer.

The source address is optional in the ACK\_DATA packet and can be omitted to reduce channel-access duration. If the transmitting node's DLL does not receive an IACK, a retry must begin within 600 μs. If the retry does not receive an IACK, the DLL passes an error back up the protocol stack.

ADDR\_ACK\_DATA service supports additional capabilities and enhances reliability. A one-bit sequence number is used by the receiving node to ignore duplicate packets from the transmitting node during a predetermined time interval defined in the CEBus specification. Because of this added feature, the DLL accesses the channel multiple times to make sure a packet using ADDR\_ACK\_DATA service is transmitted.

With the ADDR\_ACK\_DATA, the receipt packet includes the ADDR\_IACK type in the control field. As well, the destination addresses must be present—which means that the source and destination address must also be present in the requesting ADDR\_ACK\_DATA packet.

If the transmitting node's DLL does not receive an IACK, a retry must begin within

600 μs. If the retry does not receive an IACK, the DLL relinquishes the channel. It may reaccess the channel and attempt to repeat the transmit process without passing an error up the stack. Only if the DLL exhausts all of the predetermined channel-access attempts is an error reported.

ADDR\_UNACK\_DATA has similar capabilities to ADDR\_ACK\_DATA service without acknowledgment packets or immediate retries. For ADDR\_UNACK\_DATA, the DLL transmits multiple copies of the packet using multiple channel accesses.

Packets using a broadcast address must use unacknowledged services (either UNACK\_DATA or ADDR\_UNACK\_DATA) since the acknowledgments from the many receiving nodes would collide and result in unreceivable noise. ADDR\_UNACK\_DATA is the preferred service for broadcast packets since multiple packets using multiple channel accesses are possible and result in higher reliability.

Packet priority is used by the DLL to determine the channel-access priority timing. To gain access to the channel, a node first listens for channel activity (carrier sense). If there is activity, the node waits until it is finished. After a fixed amount of time (based on priority) plus a random amount of time, the node can attempt to gain channel access by sending a random-number packet preamble used for contention resolution. If no contention is detected, the packet is sent. If contention is detected, the node must wait for a new channel access and transmission attempt.

The earliest a packet with the highest priority can start is 1 ms after the previous packet ends. The only

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sequence number	Service class	Reserved	Packet priority		Packet type		
Packet type (bit 2, 1 and 0)			Packet priority (bit 4 and 3)				
000	IACK		00	High			
001	ACK_DATA		01	Standard			
010	UNACK_DATA		10	Deferred			
011	*		11	*			
100	FAILURE		Service class (bit 6)				
101	ADDR_ACK_DATA		0	Basic			
110	ADDR_IACK		1	Extended (undefined at this time)			
111	ADDR_UNACK_DATA		Sequence number (bit 7)				
Alternates each time a new packet is sent to a destination address							

Figure 3: The LPDU header sets the Data Link Layer services and chooses the media access priority.



bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Privilege	Routing		Packet flag	Extended services	Allowed media	Brouter subfield	
Privilege		Extended services					
0 Unprivileged		0 No extended services					
1 Privileged		1 Extended services octet to follow					
Routing		Allowed media					
00 Request_ID		0 This media only					
0 1 ID_packet		1 Allowed media octet to follow					
10 Directory route		Brouter					
11 Flood_route		00 No brouter address					
Packet flag		01 First brouter address presence					
0 First packet		10 Second brouter address presence					
1 Only packet		11 First and Second brouter address presence					

**Figure 4:** The NPDU header describes network information including allowed media and how the packet is routed.

two exceptions to this are for a packet retry and acknowledgment. An acknowledgment must start within 200  $\mu$ s after the end of the packet to be acknowledged. A packet retry is sent approximately 600  $\mu$ s after the previous packet ends.

The sequence number is a single bit field and is alternated for each packet sent to a destination address. This enables the DLL to distinguish a received packet which is a copy and not pass it up the stack to the application layer. A packet could be a copy due to a transmitting node using ADDR\_UNACK\_DATA with duplicate packets or using ADDR\_ACK\_DATA, in which a sending node does not hear the acknowledgment and sends a retry.

## DESTINATION AND SOURCE ADDRESS

The destination address is four bytes long giving CEBus a potential of four giganodes. The address is divided equally into two logical portions: system address and Media Access Control (MAC) address—usually called the house and unit codes since most people are already familiar with these terms.

If the unit code is zero, it is considered a house broadcast address—all nodes respond regardless of their unit code. If the house and unit codes are both zero, then all nodes respond because this is considered a global address. The destination address has the same formatting as the source address and is transmitted in the same order.

The address is placed in the packet from the unit code's least-significant bit of the least-significant byte to the house code's most-significant bit of the most-significant byte. This seemingly reverse ordering offers protection.

For instance, when the bits are actually transmitted over the channel, the DLL suppresses leading zeros to reduce the transmitted time of the packet and improve network throughput. Suppression of leading zeros is possible because end-of-field separator tokens are inserted by the DLL before the packet is transmitted.

## NPDUHEADER

The NPDU header specifies how the packet is sent. Using the mail analogy, it corresponds to using air mail, normal delivery, or "in care of" when a router transfers a packet from one medium to another (e.g., twisted pair to power line). The NPDU header consists of six fields: privilege, routing, packet flag, extended

services, allowed media, and brouter. Figure 4 shows a bit-oriented breakdown of the NPDU header.

The privilege field is restricted to packets related to system management.

The routing field sends an ID packet, request for the recipient to send an ID packet, and selects directory or flood routing from a router. An ID packet is sent out by a configured device whenever it is powered on as a sign-on message or when requested by a router. A router uses the ID packet to keep a list of nodes for each supported medium.

The packet flag field specifies if this is the only packet or the first packet of a multipacket message. Long messages can be segmented into several packets since the maximum packet length is 41 bytes, with nine used for control and addressing. This leaves 32 bytes for the complete NPDU including any CAL statements.

The extended services field specifies that additional NPDU bytes follow with additional NPDU services.

The allowed media field tells routers and brouters if they should route the message to another medium. If you had a PL-to-IR brouter, you probably would not want to route the messages to IR because IR is typically used for hand-held remotes or portable devices. If the allowed media field specifies other media, an additional NPDU byte specifies the allowed media.

The brouter field is used to control routing of packets originating or terminating on wireless media. A brouter is a device used to cross between wireless and wired media. For instance, you may want a hand-held IR

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Reserved	Mode	Type			Invoke ID		
Reserved		Invoke ID					
1 Must be 1		000 A three bit increment identifier used for packet tracking					
Mode		001					
0 BV-Basic variable length		010					
1 BF-Basic one byte fixed		011					
Type		100					
000 Not used		101					
001 Reject		110					
010 Result		111					
011 Receipt acknowledge							
100 Implicit invoke							
101 Explicit invoke							
110 Conditional invoke							
111 Explicit retry							

**Figure 5:** The APDU header specifies how the receiving application layer should respond to the packet.



remote control to send commands. Your TV could act as a router to the VCR and stereo via the power line.

	<b>Explicit Invoke</b>	<b>Implicit Invoke</b>	<b>Conditional Invoke</b>
<b>SetValue</b>	response	no response	no response
<b>GetValue</b>	response	no response (invalid)	response

Table 1: Whether a response is generated for the SetValue and GetValue methods depends on how they were invoked.

### APDU HEADER

The APDU header specifies how and if the receiving application layer should respond to the packet. There are three fields in the APDU: mode, type, and invoke ID. In our mail analogy, the APDU is like an RSVP at the bottom of the CAL message letter. It tells the CAL interpreter on the receiving end if it should respond (also called *end-to-end confirmation*) or if there are enclosures or a follow-up letter. Figure 5 shows a bit-oriented breakdown of the APDU header.

The mode field tells you whether the APDU uses multiple bytes. Most messages use the basic fixed-length APDU mode. Additional bytes are used for services such as authentication or encryption.

Authentication is used by the receiving node to verify the sending node's authority before the application layer passes the rest of the APDU to the CAL interpreter. Encryption sends packets with the message secured.

The type field has seven values: reject, result, receipt acknowledge, implicit invoke, explicit invoke, conditional invoke, and

explicit retry. Reject is sent from the receiving node's application layer which rejects the packet for some reason. Result and receipt acknowledge are sent from the receiving node's CAL interpreter to tell the sending node's CAL interpreter that the CAL command has been completed or initiated with a complete response to follow.

Implicit invoke tells the receiving node an application level response is not necessary. Explicit invoke tells the receiving node to respond with a CAL result response. Explicit retry expects acknowledgment from the receiving node's application layer within a predetermined amount of time-acknowledgment could be either result or receipt acknowledge. If none is received, the application layer (not the application) automatically retries the message.

Conditional invoke enables a device to send a response only if it has a nonempty result to return. If there is a result to return, the response packet contains a result type in the APDU header type field.

Conditional invoke could be used with a broadcast address. A response result would only be initiated by a node matching the conditional criteria since there would be a result only if the condition was true.

A SetValue CAL method does not normally have a response. If the transmitting node wants to make sure the SetValue method was handled properly by the receiving application's CAL interpreter, an explicit invoke can be used. Table 1 demonstrates the differences between the invokes and their set and get values.

Invoke ID is a 3-bit field incremented (and rolled over) for each new transmitted message to a destination address. The application

responds with a reply packet using the same value in the invoke ID field. Invoke ID, along with the destination address, is used by the sending node so when the results come back, the sending node

matches the result application packet to the proper command.

A transmitting node cannot stack or send more than one command to a receiving node until the receiving node responds to the first packet. A transmitting node sends packets to multiple destinations and uses invoke ID and the destination address to sort out the result responses.

Let me take a moment to clarify the distinction between the LPDU packet type and the APDU type. The DLL acknowledgment, if requested in the LPDU packet type field, takes place regardless of the APDU type and without the application's knowledge if the application requests acknowledged service.

As far as the application layer is concerned, it is communicating with the application layer of the receiving node. The application layer is unaware of any retries at the DLL layer and the application is unaware of any retries by the application layer.

Figure 6 shows two nodes-one node is sending an implicit invoke in the top example and the other is sending an explicit invoke to the bottom example. In the top example, the application requests ACK\_DATA DLL service, and in the bottom example, UNACK\_DATA DLL service is requested.

The application passes the packet to the DLL. If the DLL cannot get an IACK, only then does the DLL notify the application of an error. When the application layer calls for a response from the receiving-node application layer using the explicit invoke APDU type, the receiving node returns a result response packet to the original sending node. This activity is separate

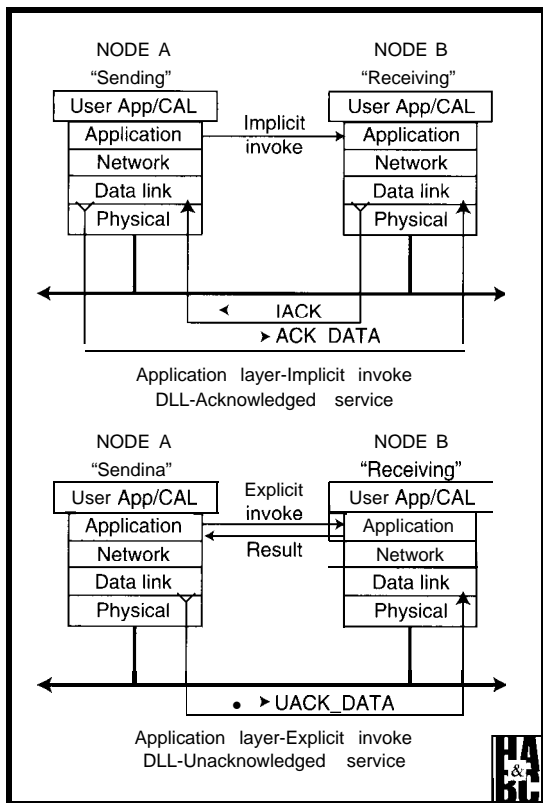


Figure 6: Here's a breakdown of the OSI layers showing the difference between CEBus Application Layer and CEBus Data Link Layer services.

from the lower level DLL acknowledgment services and can be used regardless of the DLL service.

### CAL DEVICE MODEL

CEBus uses a hierarchical model to describe each node. Each node includes two or more contexts, each made up of two or more objects. Each object contains one or more Instance Variables (IVs).

### UNIVERSAL CONTEXT

The first context in every node is called the *universal context* and has nothing to do with normal operation of the actual device. The universal context is numbered 00 and controls the device's presence on the CEBus network.

The universal context consists of two objects: the node-control object (object 0) and the context-control object (object I). The **node control object** has IVs to hold universal device information such as the device addresses, manufacturer name, and other device management information, while the context control object has a single IV called *object\_list*, which holds a list of object IDs for this context. Every context contains

Value	Name
01	Node Control
02	Context Control
03	Data Channel Receiver
04	Data Channel Transmitter
05	Binary Switch
06	Binary Sensor
07	Analog Control
08	Analog Sensor
09	Multiposition Switch
0A	Multistate Sensor
0B	Matrix Switch
0c	Multiplane Switch
0D	Ganged Analog Control
0F	Meter
10	Display
11	Medium Transport
13	Dialer
14	Keypad
15	List Memory
16	Data Memory
17	Motor
19	Synthesizer/Tuner
1A	Tone Generator
1C	Counter
1D	Clock

**Table 2:** The CAL objects published by the CEBus Industry Council are combined together to model any real-world device.

one or more IVs, which control or publish some aspect of the device. The universal context is required in every CEBus-compatible product.

Value	Mnemonic	Basic Svntax	Data Types
40	nop		
41	<b>setOFF</b>	IV	B
42	<b>setON</b>	IV	B
43	<b>getValue</b>	I V	BNC
44	<b>getArray</b>	IV [, [ <i>offset</i> ], <count>]	D
45	<b>setValue</b>	I V	BNC
46	<b>setArray</b>	IV [, [ <i>offset</i> ], <data>]	D
47	add	IV1, IV2, [IV3]	N
48	increment	IV [, <number>]	N
49	subtract	IV1, IV2, [IV3]	N
4A	decrement	IV [, <number>]	N
40	compare	IV1, IV2	BNCD
4 C	comparei	IV1 , <data>	BNCD
4D	copyValue	<b>IV1</b> , IV2 [, <context>, <object>]	BNCD
4E	swap	IV1, IV2	BNCD
52	exit	[<error number>]	
53	alias	<alias ID>[<string>]	
54	<b>inherit</b>	IV, <value>	D
55	<b>disinherit</b>	IV, <value>	D
56	if	<boolean> BEGIN <msg list> [else clause] END	
57	do	<boolean> BEGIN <msg list> END	
58	while	<boolean> BEGIN <msg list> END	
59	repeat	<boolean> BEGIN <msg list> END	
5A	build	<macro ID> BEGIN <message list> END	

Methods in bold are required for minimum CEBus implementation; “ ” is F5 delimiter

**Table 3:** CAL methods perform operations on CAL instance variables.



#210

**Tech•Arts** Home Automation

- Text to Speech Board serial I/O \$89
- Temperature boards: w/16 sensors \$239  
w/8 sensors plus 8 analog inputs \$179  
both boards: - 40° to 146°F, serial I/O
- Digital I/O ISA cards: 46 I/O ports \$125  
96 I/O ports \$169  
192 I/O ports \$249
- 4-Port ISA Serial Board w/16550s \$129  
com1-8 & irq's 2,3,4,5,10,11,12,15
- I-Servo controller board serial I/O \$89
- Windows NT TelcomFAX Personal \$129
- Automatic Drapery Controller \$139

**Call for our complete catalog!**  
support 315•455•1003  
308 E. Molloy Rd Fax 315•455•5838  
Syracuse, NY 13211 BBS 315•455•8728  
Promotions and prices are subject to change without notice. Call for guarantee and warranty information.

**800•455•9853**  
Visa • MC • Amex • COD

#209

**HOME AUTOMATION & BUILDING CONTROL**

**Home Automation**  
Two way IR & Two way X-10  
Serial Host Communications  
Standalone Operation  
Hardwire connect options  
Control hundreds of items with  
CompCo's RID/REB/RIB system!

**CompCo Engineering, Inc.**  
For on-line information call our BBS  
(615)-436-6333 evenings/nights/weekends  
Information line (615)-436-5189 voice / FAX  
Don't pay 'big \$\$\$ any longer!  
Get professional automation on a hobby budget.  
**CompCo means Computer Control!**

## OTHER CONTEXTS

CEBus defines contexts which can be grouped together to define just about every device imaginable. For instance, the lighting-control context includes parameters for defining a light switch. For a more complex device like a stereo receiver or a TV, several contexts containing various objects can be combined. For this article, I will focus on a light switch available in a 500-W dimmer version or a 15-A relay version. Refer to the CAL model for the light switch shown in Figure 6 and the CAL object list in Table 2.

The lighting context has two objects. The context control object is required to be the first object in every context with the exception of the universal context. The context control object has one IV called `object_list`, which holds a list of the objects in this context. Here, it would be 02 01 07 02, showing this context has a CEBus object type of 02 for the first (01) object and an object type of 07 for the second (02) object. The `current_value` IV is required in the light-control object to be CEBus compatible.

The CAL object 07 is an analog-control object and has 14 IVs as published by the EIA. A manufacturer can choose to implement only those IVs which make sense for a particular product or add nonstandard IVs. Unfortunately, there is no way for anyone to know what nonstandard IVs are if

Value	Description
0	Unknown Context ID
1	Unknown Object ID
2	Unknown Method ID
3	Unknown IV Label
4	Malformed Expression
5	Macro not defined
6	Alias not defined
7	Syntax error
8	Resource in use
9	Command too Complex
10	Inherit Disabled
11	Value out of Range
12	Bad Argument Type
13	Power Off
14	Invalid Argument
15	IV Read Only
16	No Default
17	Cannot Inherit Resource
18	Precondition Complete
19	Application Busy

Table 5: CAL error codes are used to indicate various application-layer error conditions.

Name	Value	Name	Value
DO	57	DELTA	ED
WHILE	58	PARAMETER	EE
REPEAT	59	NULL	FO (reserved)
BUILD	5A	MINIMUM	F1 (reserved)
AND	E0	MAXIMUM	F2 (reserved)
OR	E1	DEFAULT	F3 (reserved)
NOT	E2	DATA	F4
XOR	E3	DELIMITER	F5
GT	E4	ESCAPE	F6
GTE	E5	BEGIN	F7
LT	E6	END	F8
LTE	E7	END_OF_CMD	F9
EQ	E8	END_OF_LIST	F A
NEQ	E9	END_OF_MSG	F B
ELSIF	EA	END-OF-FILE	FC (reserved)
ELSE	EB	Error	FD
LITERAL	EC	Completed	FE

Table 4: The CAL tokens are unique symbols in the CAL message, which are used as delimiters and to create programming constructs.

they wanted to use them since the IVs are not readily available until after the manufacturer chooses to publish them.

Only five IVs are implemented in this light switch. The `current_value` stores the current dim value in percent (0-100). The `saved_value` temporarily saves the `current_value`. The `step-rate` and `step_size` set the ramp rate of the `current_value` IV used for dimming and `feature_select` manipulates the `current_value` IV and controls the light.

## METHODS, TOKENS, AND ERROR CODES

CAL methods are used to perform operations on CAL instance variables. `SetValue` and `GetValue` are probably the most used and are shown in the example packets later in the article. Table 3 shows a list of the CAL methods.

CAL tokens are used to create programming constructs and for delimiting data. The `Data` (F4), `Delimiter` (F5), and `Literal` (EC) tokens are the most common tokens found in CAL messages. The `Data` token is used as a starting delimiter to separate array data from the preceding information. The `Delimiter` token separates portions of a CAL message. The `Literal` token precedes string data. Table 4 shows a list of the CAL tokens and their hexadecimal values.

Table 5 lists the error codes returned from a CAL interpreter. These error codes appear following an APDU with a type equal to reject. In a multiple-part command, each part has a corresponding APDU header and error code.



## DATA TYPES

There are four data types used in CAL: string, data, numeric, and Boolean. Strings are delimited by CAL tokens or are at the end of a packet. Data can be thought of as array oriented. Numeric is usually represented by a string of ASCII numbers (e.g., 31h 30h 30h for the number 100). Boolean is always true or false. The byte 01h is true and 00h is false.

## PACKET BUILDING

There are many things you can do to a CEBus light switch by sending it packets.

In this example, we turn it on and off, set a dim level, ramp to a level, read the serial number, and change the device address based on the serial number.

In the LPDU, we set the packet priority to STANDARD and the packet type to ADDR\_UNACK\_DATA. All other fields are zero for a control byte of 0Fh. Remember the sequence number is set by the DLL—we don't actually have control of it.

For this demonstration, we assume the light switch has a house code of 5 and a unit code of 1. The controller (us) has a house code of 2 and a unit code of 1.

The NPDU byte has a value of 50h. This value calls for unprivileged, directory-routed service on this medium only.

The APDU is a single byte with a value between E8h and EFh. This specifies a mode of basic one-byte fixed and a type of explicit invoke. The invoke ID increments for each packet sent.

## CAL MESSAGES

In the following examples, the first 11 bytes of each packet are the same with the exception of byte 11, where the invoke ID field is incremented. The first 11 bytes (hex) are:

0F 01 00 05 00 01 00 02 00 50 E0.

As you recall, the first byte is the control byte and the next four bytes

are the destination-address information in right-to-left order, followed by the source address, also in right-to-left order. The next two bytes are the NPDU and APDU headers.

### ON, OFF, OR DIM

The on command uses the `Set Value` method to send a value of 37 to the `feature-select` IV. This sets the current value to 100 by the definition of `feature-select`. The bytes (hex) then are:

21 02 45 66 F5 37.

The 21 is the ID of the lighting context, 02 is the object number of the analog control, 45 is the `SetValue` method, 66 is the `feature-select` IV (i.e., ASCII "F"), F5 is a delimiter, and 37 is an ASCII "7". The complete packet is:

OF01 00050001 00 02 00 50 E8 21 0245 66 F5 37.

The response to this packet is:

OF 01 00 02 00 01 00 05 00 50 D0 FE.

The LPDU (50) shows a standard



packet priority and a packet type of `ADDR_UNACK_DATA`, which does not require a response from the DLL to acknowledge packet receipt. The source and destination addresses are reversed since the device is now sending to the controller instead of receiving from it. The NPDU (EO) is the same as transmitted. The APDU has the same mode, but the type field shows that it is a result packet with the result of FE, which is the completed CAL token.

Whew! I think I'll have someone get up and turn the switch on next time.

The packet to turn the light off is identical, except the value for the `feature-select` IV changes to 33h and the invoke ID field for the APDU is incremented by one. Setting the `feature-select` to 33h also saves the `current_value` in the `saved_value` IV before setting the `current_value` to 0. This offers the feature of having the light later restore to the previous dim setting.

The result packet is the same, except the invoke ID matches the invoke ID we sent, which is what the invoke ID is intended for. We could issue several commands to this light switch. Since the result packets are all identical, except for the invoke ID, we can use this field to track the responses to the packets sent. The complete sent packet is:

OF 01 00 05 00 01 00 02 00 50 E9 21 02 45 66 F5 33

and the response is:

OF01 00020001 00 05 00 50 D1 FE.

To dim the light, we set the `current_value` IV to the dim level desired. In this example, we'll use 50%. With a packet of 21 02 45 43 F5 35 30, the 21 is the ID of the lighting context, 02 is the object number of the analog control, 45 is the `SetValue` method, 43 is the `current-value` IV (ASCII "C"), F5 is a delimiter, and 35 30 is ASCII for "5" and "0" or 50%. The complete packet is:

OF 01 00 01 00 14 00 15 00 50 EA 21 02 45 43 F5 35 30.

The result packet is once again identical, except for the invoke ID:

OF01 00020001 00 05 00 50 D2 FE.

## Let Your Development Fly!

Choose Hitex professional tools for your embedded microprocessor design and get your project off the ground ahead of schedule. Hitex builds all types of microprocessor development tools, from sophisticated in-circuit emulators to remote debuggers, monitors and simulators. Complete solutions are available for:

- ✓ the complete 6051 family
- ✓ 80C86/88, 80C186/188EA/EB/EC/XL, 60266, V20/V30/V40/V50
- ✓ 80386DX/SX/CX/EX
- ✓ 80C165/166/167
- ✓ 68HC11 family
- ✓ 6800x, 6830x, 6633x, 68340

Call Hitex for your free demo package and learn how smooth and efficient development with professional tools really can be!

HiTOOLS Inc.  
2055 Gateway Place  
Suite 400  
San Jose, CA 95110  
☎ (406) 451-3986  
☎ 1-800-45HITEX  
☎ Fax: (406) 441-9486

**hitex**

© Copyright 1994 Hitex. Hitex is a registered trademark of Hitex. All other trademarks are acknowledged to their respective owners.



## GETTING THE SERIAL NUMBER

The serial-ii **instance variable** "s" can be found in the node control object of the universal context in object 1. The packet reading the `se r i a l _#` has the same first 11 bytes as above and the additional CAL command 00 01 43 73. The CAL command is in the universal context (00), object one (01), get the value (43) of instance variable "s" (73). The complete send packet is:

```
0F 01 00 02 00 14 00 15 00 70
EB 00 01 43 73.
```

The result packet is:

```
8F 01 00 02 00 01 00 05 00 50
D3 FE EC 54 30 30 30 30 30
303030303539.
```

Note the incremented invoke ID in the sent packet and the matching invoke ID in the result packet. After the complete token, there is a delimiter token (EC) and the serial number follows "T00000000059," which actually matches the serial number printed on the side of the switch!

## USING THE SERIAL NUMBER

Since the serial number is known from the manufacturer's label on the device, it provides a good way to exclusively communicate with this unit for set-up purposes. Normal communication uses the device address after it is set and the device is configured.

However, we will send a broadcast message using the conditional invoke APDU type and ask for the house code in return if the serial number condition is met. The packet this time is a little longer due to the 1%-character serial number and the extra bytes required to form the conditional expression. Note the house code is an array value and must be dealt with using the methods and delimiters for handling arrays.

The packet to get the house code from the module with the serial number equal to "T00000000059" is:

ID	CONTEXT			
00	UNIVERSAL			
NO	OBJECT			CLASS
01	NODE CONTROL (DEVICE CONTROL)			01
IV	NAME	PS	TYPE	
s	serial_#	R	string	
h	system_addr	R/W	data	
a	mac_addr	R/W	data	
g	group_addr	R/W	data	
n	manuf_name	R	string	
w	power	R	Boolean	
l	on_offline	R/W	Boolean	
o	context_list	R	data	
i	setup	R	numeric	
u	user_feedback	R/W	numeric	
p	product_name	R/W	string	
f	configured	R/W	Boolean	
r	controller_present	R/W	Boolean	
G	default_group_addr	R/W	data	
N	repeat_enable	R/W	Boolean	
t	config_master	R/W	Boolean	
H	report_header	R/W	data	
A	dest_address	R/W	data	
02	CONTEXT CONTROL			02
IV	NAME	PS	TYPE	
o	object list	R	data	
21	LIGHTING			
NO	OBJECT			CLASS
01	CONTEXT CONTROL			02
IV	NAME	PS	TYPE	
o	object list	R	data	
132	ANALOG CONTROL (LIGHT LEVEL CONTROL)			07
IV	NAME	PS	TYPE	
c	current_value	R/W	numeric	
s	saved_value	R/W	numeric	
r	step_rate	R/W	numeric	
S	step_size	R/W	numeric	
f	feature_select	R/W	numeric	

Figure 6: A typical light-control module would have two contexts, four objects, and twenty-five Instance Variables.

```
0F 00 00 00 00 01 00 05 00 50 F0 00 01 56
73 EC EC 54 30 30 30 30 30 30 30 30 35
39 F7 44 68 F8.
```

The control byte is the same as before (0F), the destination address is the system broadcast address (00 00 00 00) and the source address is our address (01 00 05 00). The NPDU header (50) is the same, the APDU header is now the conditional invoke type (F0), and we are dealing with the universal context (00) and node control object (01). The CAL message begins with the IF token (56), the `se r i a l _#` IV (73 "s"), the Equal token (ES), a literal token (EC), and the serial number. The begin token (F7), the Get `A r ray` method (44), the house code IV (68 "h"), and finally the end token (F8) wraps it up. Simplified-if the universal context object 1, `se r i a l _#` is equal to "T00000000059," then get the array value of the house code. The response packet is:



```
0F01 00 02 00 01 00 05 00 50 D4
FE F4 32 F6 00 05.
```

We actually played a sneaky trick on the module. We asked for the house code, which it dutifully sent, but we then ignore the CAL portion of the packet and get the source address, which additionally gives us the unit code without sending another packet!

## SUMMARY

I have known about CEBus for the past five years and about six months ago began developing a CEBus product. It was difficult at first because of the broad base of information necessary before you can actually do anything. This article includes a healthy mix of the things that gave me trouble or were hard to find and decipher from reading IS-60.

Good luck on your CEBus project.

*Peter House is an applications engineer with Intellon Corporation, a manufacturer of the spread-spectrum carrier components used to implement CEBus on RF and PL media. He may be reached at 71773.2775@compuserve.com.*

## SOURCES

The EIA CEBus Standard IS-60 is available from:  
Global Engineering Documents  
1990 M St. N.W., Ste. 400  
Washington, DC 20036  
(202) 429-2860  
Fax: (202) 33 1-0960

The CEBus dimmer module, relay module, serial computer interface, and the module's technical reference manual are available from:

Home Automation Labs  
105 Hembree Park Dr., Ste. H  
Roswell, GA 30076  
(404) 442-0240  
Fax: (404) 410- 1122

## I R S

416 Very Useful  
417 Moderately Useful  
416 Not Useful