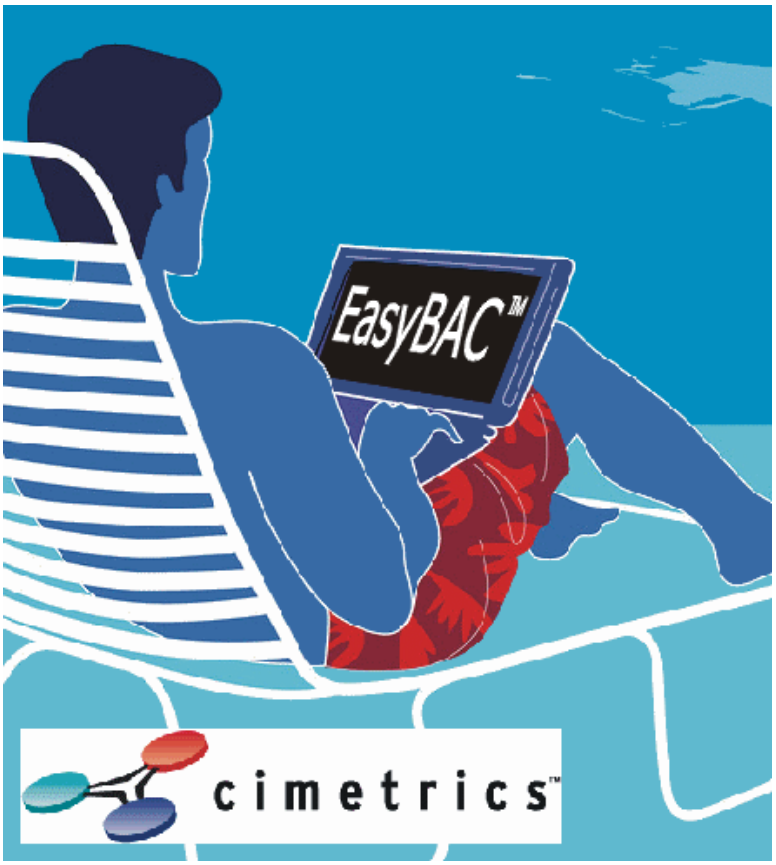


EasyBAC™ API v1.0

(EasyBAC Development Kit)

Manual v1.0

P/N: 010-002-00037

Introduction

Thank you for your interest !

by Bob Ofenstein, V.P. Products Group

Thank you for either purchasing or evaluating our products.

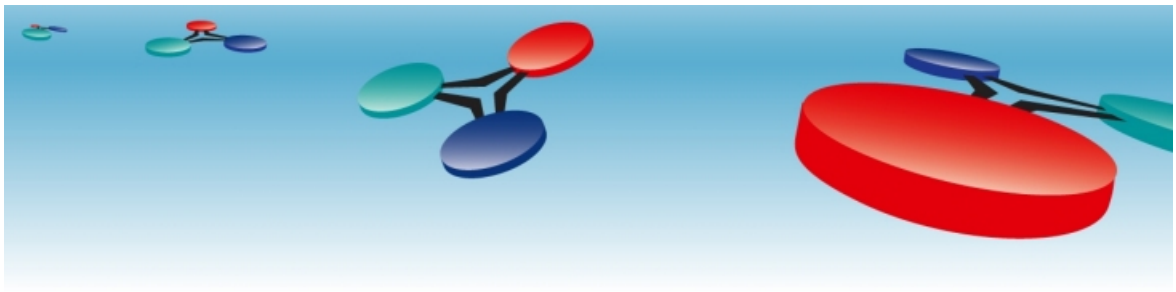
We take great pride in providing solutions that offer state-of-the-art features that help you make money.

Please stay in touch because we are CONSTANTLY adding new products and features that make your life easier.

Table of Contents

1 Introduction

1.1 The EasyBAC concept



The **Cimetrics EasyBAC API** (**A**pplication **P**rogram **I**nterface) is a **protocol specification** that let's you exchange data with a BACnet network using relatively simple commands.

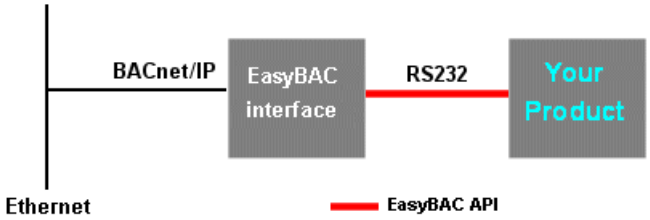
The EasyBAC interface (and module) contains a microprocessor system with an Ethernet port communicating with the standard BACnet/IP protocol, and a serial port communicating using the EasyBAC protocol. Inside the EasyBAC interface (and module) are "Virtual Objects" which represent the properties of your product to the BACnet network.

You configure a "Virtual Objects" file that represents your products features using our Virtual Object Creator software and you download this file to the EasyBAC interface (or module). You do not add any your code to this module ! This product is a closed gateway component.

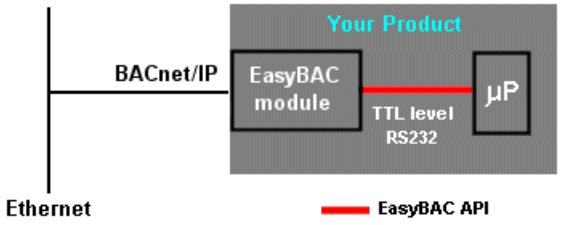
The idea here is that **you need to write a program inside your product** (NOT inside the EasyBAC product) which communicates using the EasyBAC serial protocol.

Look at the block diagrams below to understand this:

Application #1: You have an existing product and you just want to add the EasyBAC interface by connecting an RS232 cable to the Cimetrics box. You need to program your existing product to use the EasyBAC communications API and the interface presents "virtual objects" representing your product on the BACnet/IP network.



Application #2: You want to create a new product which speaks BACnet/IP to the outside world. The small EasyBAC module can be mounted inside your product and performs the same function as an integral system. We supply you with all design information and the parts as a standard manufacturing component.



1.2 EasyBAC API overview

An EasyBAC interface (or module) provides a set of **BACnet** "**Virtual Objects**" which represent the functionality of your product.

The main EasyBAC functions are:

- Maintain the BACnet "**Virtual Objects**" exposed to the BACnet network.
- Execute and initiate BACnet service requests.
- Provide a link between your product and properties in the BACnet Object Database that represent physical values (typically inputs or outputs).

BACnet Object Database is created by using the **EasyBAC Virtual Object Creator™** software (a Windows XP program). This configuration file is downloaded in advance and saved to the EasyBAC interface's (or modules) flash memory. This Virtual Object file is downloaded over Ethernet.

The EasyBAC interface (and module) **has a built-in Web server** so installation parameters (such as IP address setup) can be done using a standard browser. This browser screen can also be customized to include your company logo.

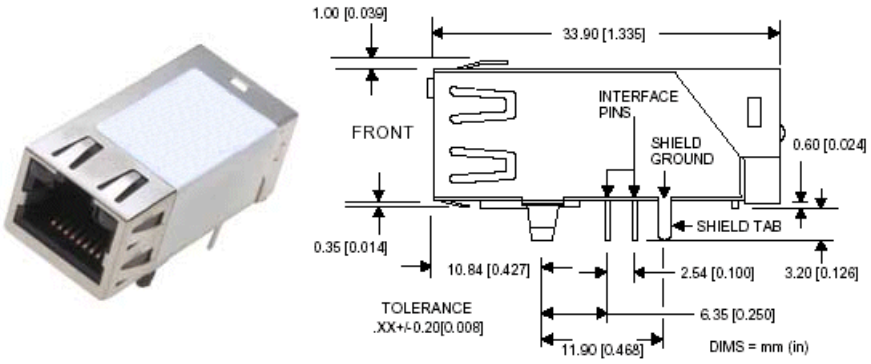
Communication between the EasyBAC and your product is performed using custom protocol over an RS-232 link.

1.3 EasyBAC development

There are three main issues that need to be understood:

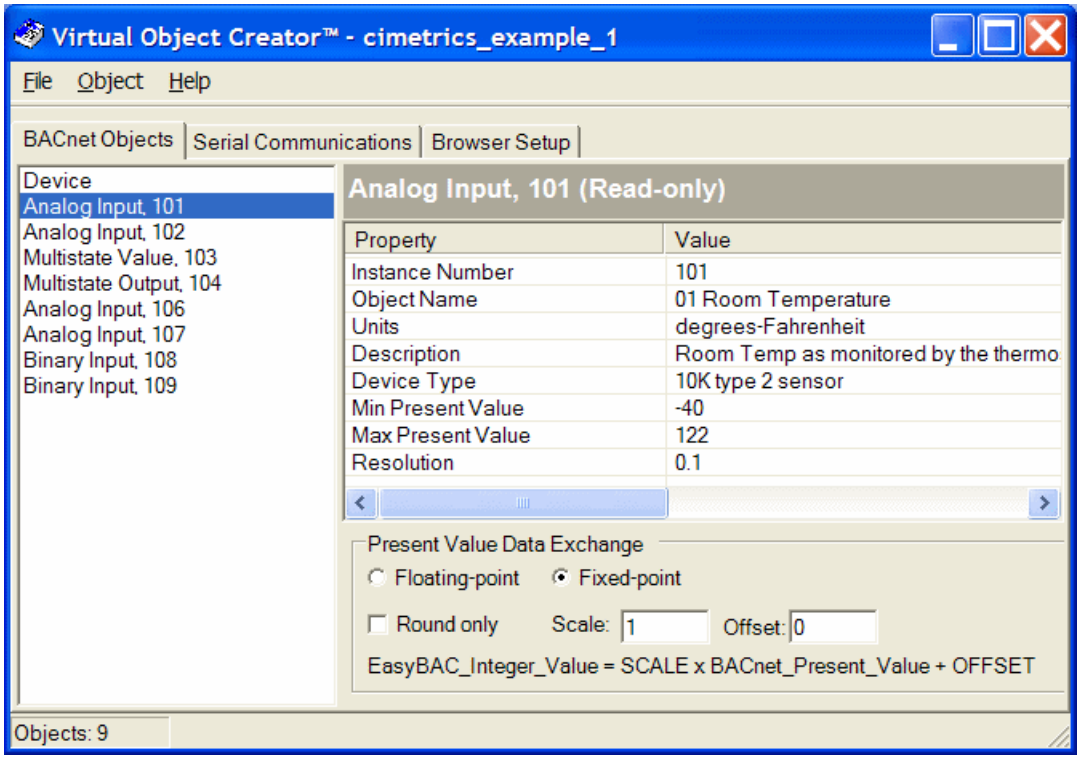
1) Add the module to your hardware (not required if you use the EasyBAC interface)
2) Create your Virtual Objects file and download it to the interface (or module).
3) Program your microprocessor to communicate via serial using our API

1) EasyBAC hardware: If you use the EasyBAC interface, then there is no hardware development. If you would like to use the EasyBAC module then you will need to do traditional design efforts with adding this component to a schematic and then creating a PCB layout for assembly. **Five signals need to be supplied by your circuit**: 3.3VDC@270mA, ground, reset, TTL level serial in, and TTL level serial out.



2) Virtual Object Creator™: The EasyBAC interface (as well as the module) contains a full microprocessor system with a Real Time Operating System and the Cimetrics BACstac® protocol stack. The only software piece that is missing is a definition of the BACnet Objects needed for your product. Therefore, we supply you with a Windows based configuration software that lets you define what BACnet Objects are needed. When you complete this definition, downloaded this configuration file into the EasyBAC hardware via Ethernet. ***This configuration only needs to be done once.***

NOTE: There is a separate HELP file for the Virtual Object Creator software.



3) Programming your product: The last part of the development process is to program your microprocessor to communicate using the EasyBAC API via a serial connection**.** In general, the format of the EasyBAC protocol is nothing more than sending an "Object number, data value" every time something changes inside your product. This communications is two way, so if something on the BACnet network changes a data value, the EasyBAC interface (or module) will

write "Object number, data value" to your product. This communications API is easily implemented by even the smallest of microprocessors.

serial API details are a separate HELP file the Virtual Object Creator software.

1.4 EasyBAC example products

EasyBAC supports functions that fit the BACnet profile called an "**Application Specific Controller**" (B-ASC). Examples of B-ASC products are:

- Low Voltage Variable Frequency AC motor drive controller
- Programmable Logic Controller - VAV (Variable Air Volume) box
- Central Plant Controller
- Air Handling Unit Controller
- Fume Hood Controller
- Large Terminal Unit Controller
- Input Monitoring Devices
- Lighting Panel Controller
- Heat Pump Controller
- Multi-Speed Fan / Motor Controller
- Network Thermostats
- Fire Panel Controllers
- Gateways to other protocols
- Universal I/O Products (Digital I/O, Analog I/O, Multi-state I/O)
- Energy Management Systems
- Wireless Access Point Gateways
- HVAC or Lighting User Interface Devices
- Energy Management Controllers

1.5 License Terms and Conditions

EasyBAC™ and Virtual Object Creator Software Agreement

This agreement is between Cimetrics, Inc., a Delaware Corporation having offices at 125 Summer Street, 21st Floor, Boston, Massachusetts 02110, hereinafter "CIMETRICS"; and the purchaser (or user/company installing this software) of the Cimetrics technology hereinafter "Licensee".

Cimetrics is a) the owner of certain software known as EasyBAC and Virtual Object Creator, and b) a provider of services in connection with the use of said products; and Licensee wishes to obtain rights to the use of these Cimetrics products and associated tools and to receive the benefit of Cimetrics services in connection therewith, all under the terms set forth herein.

THEREFORE, for good and valuable consideration, the receipt and sufficiency of which is acknowledged by the parties, it is agreed:

Definitions: The following terms used throughout this Agreement shall have the meanings set forth below:

"EasyBAC and Virtual Object Creator" means ALL programs, technologies, and associated code embodied in computer files supplied by CIMETRICS as part of, for use with, or for development of, BACnet enabled products or the like. This includes any software products included in this package which contain the Cimetrics BACstac© Protocol Stack. Licensee may not copy or distribute this code, except for a limited number of copies for backup or archival purposes.

"Documentation" means user manuals, or help files, and principles of operation, which relate to the use or understanding of the Licensed Material. Licensee may only copy and distribute such portions of the Documentation as may be necessary to enable a End User to use the Product.

"End User" means a purchaser of Licensee's Products.

"Licensed Materials" means the software supplied by CIMETRICS, in human- or machine-readable form, and related documentation, as well as hardware and firmware, owned by CIMETRICS and licensed to Licensee.

"Library component" means a collection of Source Members and Object Members supplied to Licensee by CIMETRICS for use in implementing the BACnet Protocol, whether in machine readable or human readable form. Licensee may not copy or distribute any Library, except for a limited number of copies for backup or archival purposes.

"Merged Code" means a program developed by Licensee into which one or more Library components have been incorporated through compilation, assembly or linking. Licensee is prohibited from making this type of Code and is not licensed to distribute the EasyBAC and Virtual Object Creator and Associated Code.

"Object Member" means machine code which can be directly executed on a computer (i.e. is not in a human-readable higher-level language), whether the member is a complete program or a piece of code intended to be linked with or incorporated into a larger program. Licensee may not distribute, reverse engineer or disassemble any Object Members.

"Primary Supplier" means the company who sold the EasyBAC and Virtual Object Creator and Associated code to Licensee.

"Product" means software and/or hardware, and systems manufactured, designed, or marketed by Licensee which incorporates the Licensed Materials in any form.

"Protocol" means a method of interconnecting a network of embedded controllers and hosts.

"Protocol Stack" means the BACstac libraries, service applications, and device drivers necessary to run the Protocol (contained within the EasyBAC module).

"Seat" means a single computer used by a single software developer.

"Software" means instructions intended for use with computers, including Protocol Specifications, Libraries, Test programs, and Associated Code developed by, and/or trademarked, and/or copyrighted by CIMETRICS, and Merged Code incorporating any such software.

"Software Development Kit (SDK)" means the EasyBAC and Virtual Object Creator software package, including all components of the EasyBAC and Virtual Object Creator code, Protocol Specifications, test software, simulators and other code supplied with the package.

"Source Member" means computer code written in a higher-level language, whether the member is a complete program or a piece of code intended for inclusion into a larger program. Licensee may not distribute any Source Members.

Article I – Licensing - General Issues

1.1 Copyright Law: Each copy of the Licensed Materials is protected by United States and international copyright laws, international treaty, and other applicable laws, and all copyrights therein are owned by Cimetrics. Licensee shall handle and treat the Licensed Materials like any other copyrighted material in accordance with all applicable law. Licensee shall not permit of suffer any copy to be made of any of the Licensed Materials, except as provided for herein.
1.2 No Copying: Licensee may not make or permit any copy or distribute any portions of the EasyBAC and Virtual Object Creator) and Associated Code under any circumstances.
1.3 Backups: Licensee may make a copy of the EasyBAC and Virtual Object Creator) SDK for backup purposes in connection with Licensee's Internal use.
1.4 Prohibited Uses: Licensee shall not use the Licensed Materials or any other form of Cimetrics Intellectual Property in whole or in part for any purpose except as expressly provided for in this Agreement. Prohibited uses include, without limitation: a) use, transfer, license, grant, lease, copy, disclosure, or other execution, to or for the benefit of any persons, entities, or organizations other than Licensee; and b) Permitting or suffering access to any of the Licensed Materials or Cimetrics Intellectual Property by any person other than Licensee. Cimetrics shall not be subject to any limitation on any of its rights to conduct any form of business with any form of its Intellectual Property as a result of this Agreement.
1.5 Ownership of Licensed Materials: No title or ownership of any of the Licensed Materials, any

Software, nor any proprietary technology, is transferred under this Agreement to Licensee. Notwithstanding any provision of this Agreement to the contrary Cimetrics owns and retains all title and ownership of all Intellectual Property rights in the Licensed Materials (including all software, copies and documentation). Cimetrics does not transfer any title or ownership, or any of the associated goodwill, to Licensee, and this Agreement shall not be construed to grant Licensee any right or license, whether by implication, estoppel or otherwise, except as expressly provided herein. Licensee agrees to observe the proprietary nature of the Licensed Materials licensed under this Agreement. Licensee agrees to take appropriate action by instruction or agreement with its employees, agents, and contractors who are permitted access to the Licensed Materials to fulfill Licensee's obligations under this Agreement, including but not limited to measures to secure and protect each Library contained therein, Associated Code, and Documentation and copies thereof. Violation of any provision of this Article shall be the basis for immediate termination of this License Agreement.

Article II – License Fees

2.1 SDK - Per Seat Licensing: The EasyBAC and Virtual Object Creator Software Development Kit (SDK) are sold on a per-computer basis. Every engineer who is using the SDK materials to create an application must purchase an EasyBAC SDK. The License Fee is contained in the purchase price of this product.
2.2 Run-Time Licensing: Licensee may only use the EasyBAC and Virtual Object Creator technologies when used together with an EasyBAC interface (or module) purchased from Cimetrics or a legally authorized representative of Cimetrics. Run-time licensing fees are included in the purchase price of these products.

Article III – Support

3.1 Scope – All technical support is for understanding specific EasyBAC and Virtual Object Creator software functions and technical integration with Licensee software's core functionality. This support does NOT cover basic education concerning the BACnet standard and/or Licensee application issues.
3.2 Initial Support – The initial support period is 90 days from the time of purchase. During this time, the Primary Supplier must be the main contact when requesting support. Cimetrics will work together with the Primary Supplier to provide unlimited e-mail support and occasional phone support for one designated individual per purchase.
3.3 Ongoing Support - After the initial support period, Licensee may purchase a yearly support agreement for $1,000 per year per seat.. This includes unlimited email and reasonable phone support as well as software maintenance updates. Cimetrics will offer this option for a minimum of two (2) years after Licensee's original purchase.

Article IV – Confidentiality

4.1 Confidentiality: The Licensed Materials are proprietary to CIMETRICS and title thereto remains in CIMETRICS. All applicable rights to patents, copyrights, trademarks and trade secrets in the Licensed Materials or any modifications to the Licensed Materials (whether or not made at Licensee's request) are and shall remain in CIMETRICS. Licensee agrees to secure and protect each Library and all Source Members contained therein, Associated Code, and Documentation and copies thereof, in a manner consistent with the maintenance of CIMETRICS' rights therein and to take appropriate action by instruction or agreement with its employees or consultants who are permitted access to such material to satisfy its obligations hereunder.

4.2 Warranty and Disclaimer: CIMETRICS MAKES AND LICENSEE RECEIVES NO WARRANTY EXPRESS OR IMPLIED AND THERE ARE EXPRESSLY EXCLUDED ALL WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. CIMETRICS SHALL HAVE NO LIABILITY WITH RESPECT TO ITS OBLIGATIONS UNDER THIS AGREEMENT FOR CONSEQUENTIAL, EXEMPLARY, OR INCIDENTAL DAMAGES EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Cimetrics products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the product could create a situation where personal injury or death could occur. Should Licensee use any Cimetrics product for such a use, or incorporate any Cimetrics product into any product or system intended or used for such a use, Licensee will indemnify and hold Cimetrics and its officers, employees, subsidiaries, affiliates and distributors harmless against all claims, costs, damages and expenses (including reasonable attorneys' fees) arising, directly or indirectly, out of any claim of personal injury or death associated with such unauthorized use, even if such claim alleges that Cimetrics was negligent or that Cimetrics knew, or should have known, of such use of the product.

Article V – Cimetrics' Termination Rights

5.1. Termination: Cimetrics shall have the right to terminate the license granted herein if the Licensee is in breach of their obligations under this Agreement. In particular, confidentiality and prohibited uses are obligations which Cimetrics considers particularly important. If the Licensee is in breach of their obligations,

Cimetrics will notify the Licensee that a violation has occurred and that corrective action is required.

Cimetrics shall give written notice of the nature of the breach of this Agreement, and allow the Licensee not less than thirty (30) days to cure the breach or breaches.

If the breach has not been corrected within this time period, Cimetrics shall notify the Licensee of its intent to terminate their license and shall specify the proposed termination date. This notice shall be sufficient if sent by registered or certified mail return-receipt requested addressed to the Licensee at his last known address.

Termination under this paragraph shall not relieve Licensee of any of its obligations under this Agreement. Termination of this Agreement shall be in addition to and not in lieu of any other or further equitable remedies available to Cimetrics for any breach by Licensee.

Article VII – General Provisions

7.1. General: Each party acknowledges that it has read this Agreement, understands it, and agrees to be bound by its terms, and each further agrees that this Agreement is the complete and exclusive statement of the agreement between the parties on the subject matter stated. This Agreement may not be modified or altered except by written instrument duly executed by both parties. This Agreement may not be assigned by either party without the prior written consent of the non-assigning party.

7.2. Governing Laws; Courts: This Agreement and performance hereunder shall be governed by the laws of the Commonwealth of Massachusetts, and any action arising out of this License shall be brought in the courts of the United States and/or the Commonwealth of Massachusetts in the County in which Cimetrics has its principal office.

7.3. Severable Provisions: If any provision of this Agreement is determined to be invalid under any applicable statute or rule of law, it is to that extent to be deemed omitted, and in such case the remainder of this Agreement shall remain in full force and effect.

7.4. Costs of Enforcement: In the event that either party shall be required to seek enforcement of this Agreement by any means, the prevailing party shall have the right to collect its reasonable costs and expenses incurred in doing so, including a reasonable attorney's fee.

By installing this software, Licensee causes this Agreement to be executed and enforcable in a court of law.

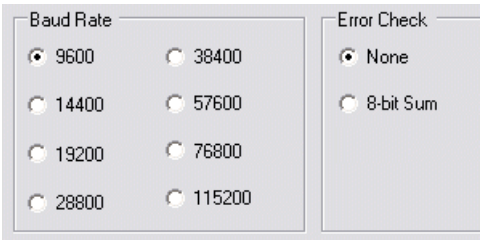
2 EasyBAC serial - overview

2.1 Baud rates / error checking

Your microprocessor's UART must have the following capabilities:

- Asynchronous, CMOS
- Development-time configurable baud rate
- No flow control
- 8 data bits (least significant bit first), no parity bits, 1 stop bit (one)

Serial transmission baud rates and error checking choices are:



NOTE: This screenshot above is from the Virtual Object Creator software that configures the EasyBAC interface (or module). Your UART needs to communicate with the EasyBAC module using the same settings.

When **Error Check = None**, the header checksum must be present (data is ignored) and the payload checksum is not required.

When **Error Check = 8-bit Sum**, the header checksum must be present (calculated), and the payload checksum is present (calculated) if the Payload Length is non-zero.

Header checksum: When calculating header checksum, three bytes are involved: 1-byte message type and 2-byte payload length.

Payload checksum: When calculating payload checksum, all payload bytes (starting from the byte immediately following the header checksum byte) are involved.

Calculations: To calculate a checksum use the following algorithm:

```
byte Calc8bitChecksum (byte data[], int count) {
        int     i;
        byte    s ;

        s = 0;
        for (i = 0; i < count; i++)
                s += data[i];
        s = ~s + 1;

        return s;
}
```

Verification: When verifying a checksum, the same bytes that were involved in calculating the checksum plus the checksum itself are involved in calculation. To verify a checksum use the following algorithm:

```
bool Verify8bitChecksum (byte data[], int count) {
        int     i;
        byte    s ;

        s = 0;
        for (i = 0; i < count; i++)
                s += data[i];

        return s == 0;
}
```

2.2 Message exchanges

EasyBAC and your microprocessor exchange **unconfirmed messages.**

Every message assumes certain actions taken by the receiving side. In some cases this involves responding with one or more messages. **However, all messages are considered to be independent and not "replies" to any other message.** This means that neither EasyBAC nor microprocessor needs to block waiting for a response/confirmation after having sent out a message.

The messaging format is essentially...

1. **Identify the Object** - The **Object ID** parameter is encoded as 32-bit value.

2. **Identify the Property** - The **Property ID** parameter is encoded as 16-bit value (matching the BACnetPropertyIdentifier enumeration)

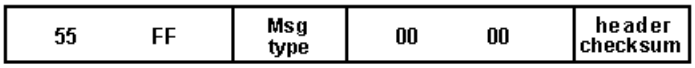
3. **Specify the Value** - This value match the data from sensors or control elements within your product or values received over the BACnet network directed to your device.

As a part of specifying the value, you need also need to specify what data type you are using and what priority this value should be (optional). Read the section on Data types and Priority.

NOTE: "microprocessor" refers to your product. See the Block diagram.

2.3 Frame format

Messages 0 thru 3 have the following frame format:



Messages 4,5, and 6 have this frame format:

example data = Object_ID, Property_ID, Priority, Data_Type, Data_Value

Field	**Length**	**Value/Comment**
Preamble	two bytes	X'55FF'
Message Type	one byte	
Payload Length	two bytes	*Most* significant byte first
Header checksum	one byte	two's complement sum of data bytes
Payload	*variable*	Is present only if Payload Length is non-zero
Payload checksum	one byte	present if Payload Length is non-zero and error checking is used

The **Message Type field** is used to distinguish between different protocol messages:

EasyBAC_Started	0
not used	1
Get_All	2
Get_All_Complete	3
Input_Property	4
Output_Property	5
Indicate_Property	6

3 EasyBAC serial - messages

3.1 (0) FB: EasyBAC Started

Message Type: 0

Initiated by the EasyBAC interface (or module). EasyBAC sends this message to the microprocessor every time it boots and is ready to communicate.

The FB: EasyBAC_Started message contains no fields.

A sample encoding of an **EasyBAC_Started** message:

```
55:FF          -- preamble
00             -- message type      (EasyBAC_Started)
00:00          -- payload length
00             -- header checksum   (example = no error checking)
```

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.

3.2 (1) not used

Message Type: 1

This was used during EasyBAC beta testing but is not used in the production version.

3.3 (2) TB: Get All

Message Type: 2

Initiated by your microprocessor. Queries EasyBAC about current values of Present_Value and Relinquish_Default properties of all Output and writeable Value objects.

Upon receipt of a TB: **Get_All** message, EasyBAC sends:

FB: Indicate_Property
FB: Indicate_Property
...etc... (for every relevant property)
FB: Indicate_Property
FB: Get_All_Complete

The FB: Get_All messages contains no fields.

A sample encoding of an **Get_All** message:

```
55:FF          -- preamble
02             -- message type      (Get_All)
00:00          -- payload length
00             -- header checksum   (example = no error checking)
```

NOTE:
1) "microprocessor" refers to your product

2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.
3) the header checksum is always present. If Error Checking = 8 bit Sum this is used, and if Error Checking = none this is ignored.

3.4 (3) FB: Get All Complete

Message Type: 3

Initiated by the EasyBAC interface (or module). Notifies the microprocessor that all Indicate_Property messages triggered by Get_All have been sent.

see an example here.

The Get_All_Complete message contains no fields.

A sample encoding of an **Get_All_Complete** message:

```
55:FF       -- preamble
03          -- message type      (Get_All_Complete)
00:00       -- payload length
00          -- header checksum   (example = no error checking)
```

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.
3) the header checksum is always present. If Error Checking = 8 bit Sum this is used, and if Error Checking = none this is ignored.

3.5 (4) TB: Input Property

Message Type: 4

Initiated by your microprocessor. This sends a new property value to EasyBAC. This will be the new value represented on the BACnet network.

See Out-of-Service behavior for exceptions to the normal message processing.

The Input_Property message consists of the following fields:

Object ID	4 bytes	identifies object whose property must be set
Property ID	2 bytes	specifies property whose value must be set
Priority	1 byte	indicates the level of importance (see the Priority chapter)
Data Type	1 byte	Real or Int16 (for Analog objects), Enum8 (for Binary objects) or Unsigned16 (for Multi-state objects)
Value	N	(depends on Data Type, see above)

A sample encoding of an **Input_Property** message:

55:FF	-- preamble	
04	-- message type	(Input_Property)
00:0C	-- payload length	(example)
F0	-- header checksum	(example = 8 bit checksum)
00:00:00:01	-- Object ID	(example = Analog-input, 1)
00:55	-- Property ID	(example = Present-Value)
10	-- Priority	(example = unused/default)
03	-- data type	(example = real)
40:48:F5:C3	-- data value	(example = 3.14)
57	-- payload checksum	(example = 8 bit checksum)

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.
3) the header checksum is always present. If Error Checking = 8 bit Sum this is used, and if Error Checking = none this is ignored.
4) The sample above does not show a payload checksum (which is set as an error checking option in the Virtual Object Creator software)

3.6 (5) FB: Output Property

Message Type: 5

Initiated by the EasyBAC interface (or module). EasyBAC tells the microprocessor of changes initiated from the BACnet network.

See Out_Of_Service handling for exceptions to the normal message processing.

The Output_Property message consists of the following fields:

Object ID	4 bytes	identifies object whose property is delivered
Property ID	2 bytes	specifies property whose value is delivered
Status Flags	1 byte	indicates current Object status flags (see Status Flag chapter)
Priority	1 byte	indicates the level of importance (see the Priority chapter)
Data Type	1 byte	Real or Int16 (for Analog objects), Enum8 (for Binary objects) or Unsigned16 (for Multi-state objects)
Value	N	(depends on Data Type, see above)

A sample encoding of an **Output_Property** message:

55:FF	-- preamble	
05	-- message type	(Output_Property)
00:0B	-- payload length	(example)
F0	-- header checksum	(example = 8 bit checksum)
03:80:00:01	-- Object ID	(example = Multistate-output, 1)
00:55	-- Property ID	(example = Present-Value)
00	-- Status Flags	(example = no fault detected - not out-of-service)
10	-- Priority	(example = unused/default)
02	-- data type	(example = unsigned16)
02:2B	-- data value	(example = 555 decimal)
E8	-- payload checksum	(example = 8 bit checksum)

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the

direction of the communications and is not a part of the command name.
3) the header checksum is always present. If Error Checking = 8 bit Sum this is used, and if Error Checking = none this is ignored.

3.7 (6) FB: Indicate Property

Message Type: 6

Sent by EasyBAC on request from the microprocessor. This Indicates the current value of a property.

The Indicate_Property message consists of the following fields:

Object ID	4 bytes	identifies object whose property is delivered
Property ID	2 bytes	specifies property whose value is delivered
Status Flags	1 byte	indicates current Object status flags (see Status Flag chapter)
Priority	1 byte	indicates the level of importance (see the Priority chapter)
Data Type	1 byte	Real or Int16 (for Analog objects), Enum8 (for Binary objects) or Unsigned16 (for Multi-state objects)
Value	N	(depends on Data Type, see above)

A sample encoding of an **Indicate_Property** message:

55:FF	-- preamble	
06	-- message type	(Indicate Property)
00:0A	-- payload length	(example)
F0	-- header checksum	(example = 8 bit checksum)
03:80:00:01	-- Object ID	(example = Multistate-output, 1)
00:55	-- Property ID	(example = Present-Value)
00	-- Status Flags	(example = no fault detected - not out-of-service)
10	-- Priority	(example = unused/default)
02	-- data type	(example = unsigned16)
02:2B	-- data value	(example = 555 decimal)
F8	-- payload checksum	(example = 8 bit checksum)

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.
3) the header checksum is always present. If Error Checking = 8 bit Sum this is used, and if Error Checking = none this is ignored.
4) The sample above does not show a payload checksum (which is set as an error checking option in the Virtual Object Creator software)

4 EasyBAC serial - options

4.1 Option - Priority

IMPORTANT: Use of this function is **optional**, but you must specify a value of "0" in the communications packet when you are not using this.

The **Priority** parameter specifies priority for writing a commandable property (range = 1 to 16). 1 is the highest and 16 is the lowest.

From the EasyBAC module, Priority has the value specified by the original WriteProperty BACnet service request, or 0 if the property in question is not commandable.

From your microprocessor, Priority can be any value within range. For non-commandable properties Priority parameter is ignored.

4.2 Option - Status Flags

The Status Flags parameter of the Output_Property and Indicate_Property messages delivers the current value of the object's Status_Flags BACnet property, which, according to BACnet, "represents four Boolean flags that indicate the general "health" of an object". Of the four flags constituting the Status_Flags property only two are currently relevant to EasyBAC:

FAULT and
OUT_OF_SERVICE

The Bit masks for these two flags that EasyBAC programmers should use are defined in the easybac.h header file:

```
//
// Flags used in the Status Flags field. Other bits are reserved.
//
#define EASYBAC_FAULT  (0x40)
#define EASYBAC_OUT_OF_SERVICE (0x10)
```

FAULT flag reflects the value of the Reliability BACnet property (writeable by both the network and the micro). The meaning of this flag is "the point is OK" (flag cleared) or "the point is faulty" (flag set).

OUT_OF_SERVICE reflects value of the Out_Of_Service BACnet property (writeable by both the network and the micro). The meaning of this flag is whether the point is working normally (flag cleared) or is turned off and is subject for the special standard-defined OUT-OF-SERVICE treatment (flag set). I don't think we shall go into details of the OUT-OF-SERVICE algorithm but refer to the Standard instead.

> **NOTE:** Your microprocessor does not need to make use of these indicators as they are handled transparently by the EasyBAC interface (or module). These are here to allow microprocessor control if desired.

4.3 Option - Out-of-Service

The EasyBAC interface (or module) automatically handles this logic when a property is set to "Out_Of_Service" from the BACnet network. Out_Of_Service can be changed by the TB: Input_Property message or from the BACnet network. Either way, the following characteristics take place:

When a property's **Out_Of_Service** changes **from FALSE to TRUE**:

- non-writeable properties become writeable (such as the Present_Value of an Input object or read-only Value objects)
- Status_Flags are updated (OUT_OF_SERVICE flag is set)
- For Output and writeable Value objects: no data updates sent from EasyBAC
- For Input and all Value objects: no Present Value updates accepted by EasyBAC
- For Input and non-commandable Value objects: value sent by the microprocessor are accepted by EasyBAC but are not network-visible until Out_Of_Service becomes FALSE.

NOTE: For commandable objects, Present_Value and Relinquish_Default values delivered by TB: Input_Property serial messages while Out_Of_Service is TRUE are ignored.

When a property's **Out_Of_Service** changes **from TRUE to FALSE**:

- non-writeable properties return to non-writeable (such as the Present_Value of an Input object or read-only Value objects)
- Status_Flags property is updated (OUT_OF_SERVICE flag is cleared).
- For Output and commandable Value objects: EasyBAC sends two FB: Output_Property messages - one with the current Present_Value and one with the current Relinquish_Default value.
- For Input and read-only Value objects: EasyBAC now responds to the BACnet network with Present_Value property
- For writeable but non-commandable Value objects: EasyBAC sends a FB: Output_Property message with the current value.
- Normal processing of WriteProperty BACnet service requests and TB: Input_Property serial messages is restored.

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.

5 EasyBAC serial - references

5.1 Data types

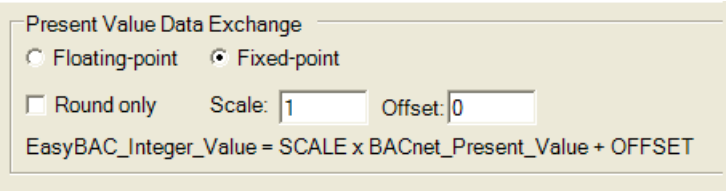
The following data types are used:

Data Type	Type Encoding	Length in Bytes	Value Encoding
Enum8	0	1	single byte
Enum16	1	1	two bytes, most significant byte first
Unsigned16	2	2	two bytes, most significant byte first
Real	3	4	IEEE 754-1985 single precision (32-bit) floating point number, most significant byte first
Bool	4	1	single byte, zero designates false, non-zero designates true
Int16	5	2	two bytes, most significant byte first

5.2 Integer behavior

One of the powerful features that we have implemented in the EasyBAC concept is allowing analog data values to be used between your microprocessor and the EasyBAC interface (or module) using integers. What is so unusual about this is that this "simple data type" is not used in the BACnet standard so the EasyBAC module must convert between **Real** numbers (Floating Point) on the BACnet side to **Integers** (Fixed Point) on the serial side.

In the **Virtual Object Creator** software, *every Analog object* lets you choose the format for exchanging the Present_Value property value between your microprocessor and the EasyBAC interface (or module).



By default, the Real datatype (Floating Point) is used - this is the same datatype used by the BACnet standard.

If Fixed point data exchange is chosen, then communication between the EasyBAC interface (or module) and your microprocessor it is calculated using the following equation:

EasyBAC_Integer_Value = scale x BACnet_Present_value + offset

where scale and offset are Real values you specify on the per-object basis. The result of evaluating of the right side of the equation is rounded off (down for fractions less than 0.5 and up for fractions greater or equal to 0.5).

Example#1:

You can simulate Integer objects (not normally supported by BACnet) with Analog objects by selecting "Round only" as this sets the scale to 1.0 and offset to 0.0. So if your microprocessor writes "**254**" then this will be represented to the BACnet network as a Real datatype of value "254.0".

scale		BACnet		offset		total
1.0	x	254.0	+	0.0	=	254

Example#2:

If you selected "Round only", and the BACnet network wrote the value "22.3" to the Present_Value of this object, EasyBAC would communicate the Integer value of "**22**".

scale		BACnet		offset		total
1.0	x	22.3	+	0.0	=	22

Example#3:

If you selected "Round only", and the BACnet network wrote the value "16.6" to the Present_Value of this object, EasyBAC would communicate the Integer value of "**17**".

scale		BACnet		offset		total
1.0	x	16.6	+	0.0	=	17

Example#4:

If the scale value is 1.2, offset value is 3.4, and the Present_Value written from the BACnet network is 2.72.

scale		BACnet		offset		total
1.2	x	2.72	+	3.4	=	6.664

Then, EasyBAC will send the Fixed point (Int16) value of "**7**" to the microprocessor.

Example#5:

If the scale value is 1.2, offset value is 3.4, and your microprocessor sends the Int16 value of "**10**" to the EasyBAC interface (or module).

scale		BACnet		offset		total
1.2	x	5.50	+	3.4	=	10

Then the value exposed to the BACnet network will be 5.50.

5.3 Object ID's

The Object ID numbers that you will need to read or write values to the EasyBAC interface (or module) are the following:

	Decimal	Hex
Device	8	8
Analog input	0	0
Analog output	1	1
Analog value	2	2
Binary input	3	3
Binary output	4	4
Binary value	5	5
Multistate input	13	D
Multistate output	14	E
Multistate value	19	13

NOTE: Hex values are used in the serial transmission.

5.4 Property ID's

The Property ID numbers that you will need to read or write values to the EasyBAC interface (or module) are the following:

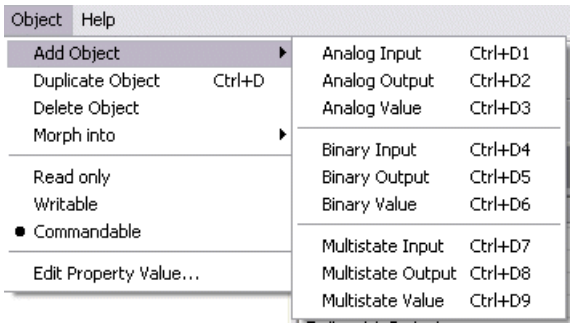
	Decimal	Hex
Present_Value	85	55
Reliability	103	67
Relinquish_Default	104	68
Out_Of_Service	81	51

NOTE: Hex values are used in the serial transmission.

6 BACnet/IP - Object types

6.1 BACnet Object overview

There are **ten BACnet Object types** that are supported by the EasyBAC protocol. Nine of these are selectable and the **Device Object** is added automatically (BACnet requires this).



Within each of these Objects, the BACnet standard defines many required and optional properties. The individual chapters for each Object gives details as to which properties are supported by the EasyBAC protocol.

> **NOTE:** All of the required properties for these Objects are supported (and some of the optional).

Supported properties are presented in a per-object table where the ***Value*** column specifies source for the property value:

- ***configured at install-time*** - these property value is specified in the field using the browser based setup screen which is shown by the EasyBAC interface (or module) These are set once by the installer and **do not change during run-time**
- ***download*** - these property values are specified in the **Cimetrics Virtual Object Creator** software and downloaded as a complete configuration "set". These **do not change during run-time**.
- **a constant** - property value is predefined and cannot be changed
- ***variable*** - property value is **changed during run-time** in response to BACnet service requests and/or microprocessor serial messages

The ***Is Writeable*** column indicates whether this property appears writeable to the BACnet network.

6.2 Device Object

Property	Value	Is Writable
Object_Identifier	configured at install-time (unique internetwork-wide)	-
Object_Name	configured at install-time (unique internetwork-wide)	-
Location	configured at install-time (optional)	-
Description	configured at install-time (optional)	-
Vendor_Name	download	-
Vendor_Identifier	download	-
Model_Name	download	-
Firmware_Revision	download	-
Application_Software_Version	download	-
Object_Type	DEVICE	-
System_Status	OPERATIONAL	-
APDU_Timeout	3 sec *(download?)*	-
Number_Of_APDU_Retries	3 *(download?)*	-
Protocol_Version	1	-
Protocol_Revision	1	-
Protocol_Services_Supported	{ READ-PROPERTY, WRITE-PROPERTY, WHO-IS, I-AM }	-
Protocol_Object_Types_Supported	{ DEVICE, AI, AO, AV, BI, BO, BV, MSI, MSO, MSV }	-
Object_List	based on downloaded configuration	-
Max_APDU_Length_Accepted	1024	-
Segmentation_Supported	FALSE	-
Device_Address_Binding	empty	-
Database_Revision	download	-

The following optional properties are not present: Max_Segments_Accepted, VT_Classes_Supported, Active_VT_Sessions, Local_Time, Local_Date, UTC_Offset, Daylight_Savings_Status, APDU_Segment_Timeout, List_Of_Session_Keys, Time_Synchronization_Recipients, Max_Master, Max_Info_Frames, Configuration_Files, Last_Restore_Time, Backup_Failure_Timeout, Active_COV_Subscriptions, Slave_Proxy_Enable, Manual_Slave_Address_Binding, Auto_Slave_Discovery, Slave_ Address_Binding, Profile_Name.

6.3 Analog Input Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Units	download	-
Description	download	-
Device_Type	download	-
Min_Pres_Value	download	-
Max_Pres_Value	download	-
Resolution	download	-
Object_Type	ANALOG-INPUT	-
Present_Value	variable (initial value is 0.0)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: Update_Interval, COV_Increment, Time_Delay, Notification_Class, High_Limit, Low_Limit, Deadband, Limit_Enable, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.4 Analog Output Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Units	download	-
Relinquish_Default	download	Yes
Description	download	-
Device_Type	download	-
Min_Pres_Value	download	-
Max_Pres_Value	download	-
Resolution	download	-
Object_Type	ANALOG-OUTPUT	-
Present_Value	variable (calculated from Present_Value and Relinquish_Default)	Yes
Priority_Array	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: COV_Increment, Time_Delay, Notification_Class, High_Limit, Low_Limit, Deadband, Limit_Enable, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.5 Analog Value Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Units	download	-
Relinquish_Default 1)	download	Yes
Description	download	-
Object_Type	ANALOG-VALUE	-
Present_Value	variable	*Optionally*2)
Priority_Array 1)	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

1) These properties are optional, they may be present if Present_Value property is writeable. Either both are present or both are absent. If these properties are present, Present_Value is commandable, otherwise it is not.

2) May be either read-only or writeable. Writeable Present_Value may or may not be commandable.

The following optional properties are not present: COV_Increment, Time_Delay, Notification_Class, High_Limit, Low_Limit, Deadband, Limit_Enable, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.6 Binary Input Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Polarity	download	-
Description	download	-
Device_Type	download	-
Inactive_Text	download	-
Active_Text	download	-
Object_Type	BINARY-INPUT	-
Present_Value	variable (initial value is INACTIVE)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: Change_Of_State_Time, Change_Of_State_Count, Time_Of_State_Count_Reset, Elapsed_Active_Time, Time_Of_Active_Time_Reset, Time_Delay, Notification_Class, Alarm_Value, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.7 Binary Output Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Polarity	download	-
Relinquish_Default	download	Yes
Description	download	-
Device_Type	download	-
Inactive_Text	download	-
Active_Text	download	-
Object_Type	BINARY-OUTPUT	-
Present_Value	variable (calculated from Present_Value and Relinquish_Default)	Yes
Priority_Array	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: Change_Of_State_Time, Change_Of_State_Count, Time_Of_State_Count_Reset, Elapsed_Active_Time, Time_Of_Active_Time_Reset, Minimum_Off_Time, Minimum_On_Time, Time_Delay, Notification_Class, Feedback_Value, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.8 Binary Value Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
*Relinquish_Default*1)	download	Yes
Description	download	-
Inactive_Text	download	-
Active_Text	download	-
Object_Type	BINARY-VALUE	-
Present_Value	variable	*Optionally*2)
*Priority_Array*1)	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

1) These properties are optional, they may be present if Present_Value property is writeable. Either both are present or both are absent. If these properties are present, Present_Value is commandable, otherwise it is not.

2) May be either read-only or writeable. Writeable Present_Value may or may not be commandable.

The following optional properties are not present: Change_Of_State_Time, Change_Of_State_Count, Time_Of_State_Count_Reset, Elapsed_Active_Time, Time_Of_Active_Time_Reset, Minimum_Off_Time, Minimum_On_Time, Time_Delay, Notification_Class, Alarm_Value, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.9 Multi-state Input Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Number_Of_States	download	-
State_Text	download (optional)	-
Description	download	-
Device_Type	download	-
Object_Type	MULTISTATE-INPUT	-
Present_Value	variable (initial value is 1)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: Time_Delay, Notification_Class, Alarm_Values, Fault_Values, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.10 Multi-state Output Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
Relinquish_Default	download	Yes
Number_Of_States	download	-
State_Text	download (optional)	-
Description	download	-
Device_Type	download	-
Object_Type	MULTISTATE-OUTPUT	-
Present_Value	variable (calculated from Present_Value and Relinquish_Default)	Yes
Priority_Array	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

The following optional properties are not present: Time_Delay, Notification_Class, Feedback_Value, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

6.11 Multi-state Value Object

Property	Value	Is Writable
Object_Identifier	download (unique device-wide)	-
Object_Name	download (unique device-wide)	-
*Relinquish_Default*1)	download	Yes
Number_Of_States	download	-
State_Text	download (optional)	-
Description	download	-
Object_Type	MULTISTATE-VALUE	-
Present_Value	variable	*Optionally*2)
*Priority_Array*1)	variable (initial value is all NULLs)	-
Status_Flags	{ 0, 0, 0, 0 }	-
Event_State	NORMAL	-
Out_Of_Service	FALSE	Yes
Reliability	No_Fault_Detected	-

1) These properties are optional, they may be present if Present_Value property is writeable. Either both are present or both are absent. If these properties are present, Present_Value is commandable, otherwise it is not.

2) May be either read-only or writeable. Writeable Present_Value may or may not be commandable.

The following optional properties are not present: Time_Delay, Notification_Class, Alarm_Values, Fault_Values, Event_Enable, Acked_Transitions, Notify_Type, Event_Time_Stamps, Profile_Name.

7 BACnet/IP - Services & Behavior

7.1 Services supported

EasyBAC supports the following BACnet protocol services:

Who-Is (Execute) Upon receipt of a Who-Is request, EasyBAC initiates an I-Am request, as

appropriate, using Device object's properties values for service request parameters.

> **NOTE: Your microprocessor is not involved** in Who-Is request execution.

I-Am (Initiate) EasyBAC initiates I-Am requests filled with Device object's properties values in the following situations:

- at startup
- upon receipt of a Who-Is request

> **NOTE: Your microprocessor is not involved** in I-Am request initiating.

ReadProperty (Execute) All properties present in the Object Database are readable.
Upon receipt of a ReadProperty request, EasyBAC performs request validation and sends back an acknowledgement, as defined by the BACnet standard. In case of a success, EasyBAC sends back to the BACnet network positive acknowledgement (ReadProperty-ACK) containing current value of the requested property from the BACnet Object Database. In case of a failure, EasyBAC sends negative acknowledgement (BACnet-Error) with appropriate BACnet error class and error code.
Current value of a property in the Database may originate from:

- EasyBAC
- Microprocessor (set by means of an Input_Property serial message)
- Another BACnet device (set by means of a WriteProperty BACnet service request)

> **NOTE: Your microprocessor is not directly involved** in processing of the ReadProperty service requests. It is involved **indirectly** by changing values of properties in the Database by means of the Input_Property serial message.

WriteProperty (Execute) Most of the properties in the BACnet Object Database are not writeable and cannot be changed by means of a WriteProperty service request. See Object Types Supported for complete list of writeable properties in each supported object type.

Upon receipt of a valid WriteProperty request, EasyBAC writes to the Virtual Object Database specified value of the specified property of the specified object and sends back to the BACnet network positive acknowledgement, as defined by the BACnet standard. In case of a failure EasyBAC sends negative acknowledgement.

If the property being written is Present_Value or Relinquish_Default, and the object in question is not out of service (see Handling Out_Of_Service Property), new property value is sent to the microprocessor using the Output_Property serial message. FB: Output_Property serial message is sent asynchronously and may be sent to the microprocessor either after or before BACnet acknowledgement is actually sent over the BACnet network.

EasyBAC WriteProperty handler performs basic request validity checks, such as existence of the object specified, existence and writeability of the property specified. Standard-mandated BACnet logic is also implemented: see handling Command Priorities and Out_Of_Service handling. However, application-level checks, such as checking Present_Value against device-specific bounds, are not performed.

The value of the priority parameter specified in a WriteProperty request for a commandable Present_Value property is included in the FB: Output_Property serial message EasyBAC sends to the microprocessor.

> **IMPORTANT: Your microprocessor is directly involved** with this command because an FB: Output_Property serial message is sent to your microprocessor.

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.

7.2 BACnet BIBBs supported

The BACnet standard defines a concept called **BIBBs** (BACnet Interoperability Building Blocks). A BIBB is a simple definition of a specific set of BACnet features that must be implemented by a device to support that BIBB.

The EasyBAC interface (and module) are capable of performing the functionality of the following BIBBs:

- **DS-RP-B** This means **DS** (data sharing), **RP** (read property), **B** (Server device)
- **DS-WP-B** This means **DS** (data sharing), **WP** (write property), **B** (Server device)
- **DM-DDB-B** This means **DM** (device management), **DDB** (Dynamic Device Binding), **B** (Server device) The "DDB" description means that this device can find another device on the network.

This set of BIBBs matches the **BACnet B-ASC profile** (without support for Who-Has/I-Have and DCC - Device Communications Control). Examples of these types of products can be found here.

7.3 Command priorities

EasyBAC handles BACnet Command Prioritization defined by the BACnet standard automatically. When a WriteProperty request is executed for a commandable Present_Value property (i.e. Present_Value in an Output object or a Value object with Priority_Array property), EasyBAC does the following:

- Write new value to the Priority_Array property with respect to the specified priority.
- Calculate effective Present_Value from the Priority_Array property and Relinquish_Default property.
- Write calculated Present_Value to the Virtual Object Database.
- Send calculated Present_Value to the microprocessor in a FB: Output_Property serial message.

Similarly, EasyBAC re-calculates Present_Value property when a WriteProperty request is executed for a Relinquish_Default property, and if Present_Value changes, EasyBAC sends new Present_Value to the microprocessor in an FB: Output_Property serial message.

NOTE:
1) "microprocessor" refers to your product
2) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.

7.4 Reliability property handling

Upon receipt of a TB: Input_Property with Property_ID set to Reliability EasyBAC automatically updates FAULT flag in the Status_Flags property in the BACnet Object Database: sets it if new Reliability value is not equal to NO_FAULT_DETECTED, and clears it otherwise.

NOTE:
1) **FB:** means "**F**rom **B**ACnet" and **TB:** means "**T**o **B**ACnet" - This is a generalization concerning the direction of the communications and is not a part of the command name.

8 General Information

8.1 Links to BACnet Resources

Here is where you can purchase a copy of the BACnet standard:

ASHRAE BACnet Standard 135-2004 - or the crazy long URL is this:

http://resourcecenter.ashrae.org/store/ashrae/newstore.cgi?itemid=22170&view=item&page=1&loginid=5193944&priority=cat311egory&words=135-2004&method=and&

This is available in as a hard-copy , CD-ROM, or download.

BACnet.org - The official ASHRAE BACnet web site.

BACnet International - A group of manufacturers who promote the use of BACnet.

BACnet Testing Lab - The organization that tests BACnet devices for conformance to the standard..

BACnet - European Interest Group - The European group which promotes the use of BACnet and holds regular training conferences in Europe.

BACnet FAQ - A good frequently asked questions page on the www.bacnet.org web site.

9 Cimetrics Information

9.1 Cimetrics Software Products

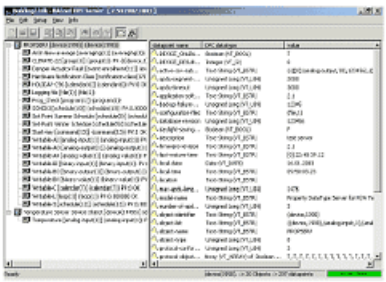
We have other **Automation Tools and Solutions** that can help you with your next job. Visit our website at www.cimetrics.com or send an email to info@cimetrics.com for further details.

NOTE: Several of these programs have demo versions (upon request).

- BACnet OPC Server - Control BACnet devices with any OPC workstation.
- BACnet Explorer - Auto discovery of devices and status on existing BACnet networks.
- BACtiveX - ActiveX software for writing custom BACnet control programs.
- BACstac series - BACnet Windows and embedded protocol stacks for manufacturers.
- BAS-o-matic - A powerful protocol analyzer for **building automation** protocols.
- Indy/A - A powerful protocol analyzer for **industrial automation** protocols.

BACnet OPC Server: Enables control and monitoring BACnet devices from any OPC workstation.

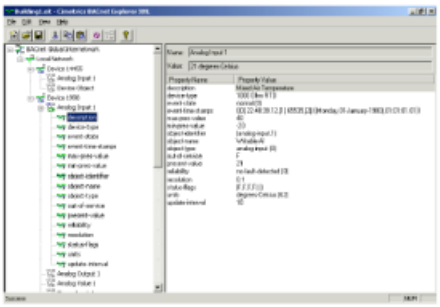
- BACnet Client and OPC Server
- Robust / mature BACnet code
- Easy "Explorer-style" interface
- BACnet auto-discovery features
- Savable configuration files
- Automatic polling features
- BACnet Event Services
- BACnet/Ethernet & BACnet/IP



go to top

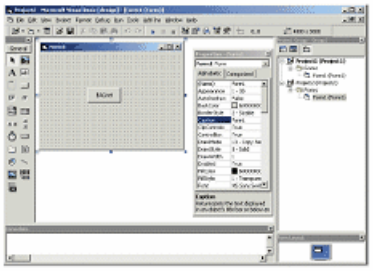
BACnet Explorer: Connect to any BACnet network and automatically discover all BACnet device settings.

- Automatic discovery of BACnet devices, objects and properties
- Expanding Tree display
- Read / Write property values
- Save point lists in machine readable format using XML
- BACnet/Ethernet & BACnet/IP



BACtiveX: Create custom Building Control programs using reusable BACnet script libraries.

- Significant software development and time to market savings
- Many BACnet script examples
- Uses robust / mature BACstac code
- Support of BACnet data sharing and BACnet Alarm and Event Services
- Password security mechanism
- BACnet/Ethernet & BACnet/IP



BACstac series: Save design time and support costs by using the industry's leading OEM BACnet protocol stack.

- Used in thousands of products from leading building automation suppliers
- Development kit with example code, test programs, well documented
- BACnet 2004 objects & services
- Porting example included (/32 only)
- Full routing network layer available
- Estimated savings of well over one man-year of development effort


BACstac Programmers' Reference © 2002 by Cimetrics Inc.

3.10 Get Alarm Summary

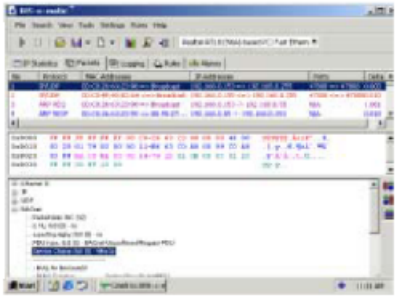
Description:

Declaration:

Access Routines:

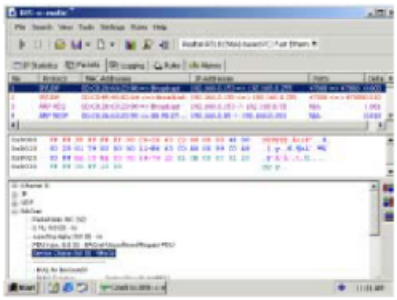
BAS-o-matic: A Protocol Analyzer with support for **building automation** protocols.

- Verify your network installation
- View communications between devices
- Easy (one button) installation and record
- Decoders for 50+ IT protocols + BACnet/IP and BACnet/E and optional BACnet MS/TP
- View statistics and bandwidth usage
- Email data to equipment support staff
- Extensive data filter options
- Save captured packets to files



Indy/A: A Protocol Analyzer with support for **industrial automation** protocols.

- Verify your network installation
- View communications between devices
- Easy (one button) installation and record
- Decoders for 50+ IT protocols + Modbus/TCP and options (Modbus RTU, EtherNet/IP)
- View statistics and bandwidth usage
- Email data to equipment support staff
- Extensive data filter options
- Save captured packets to files



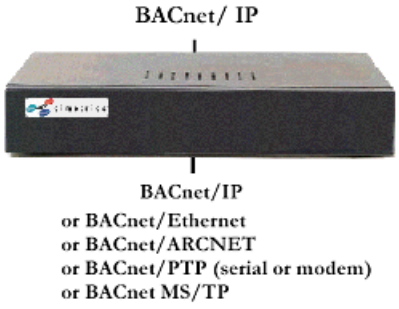
9.2 Cimetrics Hardware Products

We have many **Building Connectivity** solutions that can help you with your next job. Visit our website at www.cimetrics.com or send an email to products@cimetrics.com for further details.

- BR2 Router Series - connect BACnet systems together.
- BR4 Router Series - Encrypt your BACnet network and send data through firewalls.
- E+ Protocol Interfaces - Inexpensive BACnet/IP to serial and I/O solutions.
- Native IP Interfaces - Inexpensive TCP/IP & Web Services to serial and I/O solutions.
- U+4 Protocol Interface - USB to RS485 coprocessor interface.
- LISA BACnet/EIA 709 IC Chip - System on a chip for BACnet and EIA 709
- EasyBAC module - Create BACnet products without learning BACnet !

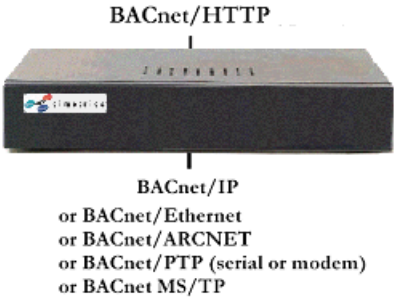
BR2 Router series: Connect BACnet systems from different manufacturers or connect BACnet networks together over the Internet (BBMD functionality).

- Connects BACnet systems from different suppliers
- Enables campus-wide control systems
- Uses industry's leading BACnet OEM protocol stack (Cimetrics BACstac)
- High bandwidth design
- Easy browser-based configuration



BR4 Router series: BACnet/HTTP gets through firewalls (port 80) and can let BACnet networks communicate from behind a NAT device.

- HTTP traffic gets through firewalls
- Connects BACnet systems from different suppliers
- Uses industry's leading BACnet OEM protocol stack (Cimetrics BACstac)
- High bandwidth design
- Many Link Layers supported
- Easy browser-based configuration



E+ series: Inexpensive serial and I/O solutions. Models include **BACnet/IP to MS/TP Router**, **BACnetIP to Web Services**, **Bacnet/IP/WS to Relays and DIN**, and more. See our web site

for the latest models (www.cimetrics.com).

- Ethernet to I/O or serial standards
- Automation protocol code by the creators of the industry's leading BACnet OEM protocol stack (BACstac)
- Professional electrical construction in a rugged metal housing
- Data and Power indication
- Screw terminal connections

Native IP series: Use IT protocols to communicate to building sensors and networks. Models include **Native IP to Building Automation (BACnet/IP)**, **Native IP to Relays & Sensors**, **Native IP to 4 Utility Meters (pulse)**, and more. See our web site for the latest models (www.cimetrics.com).

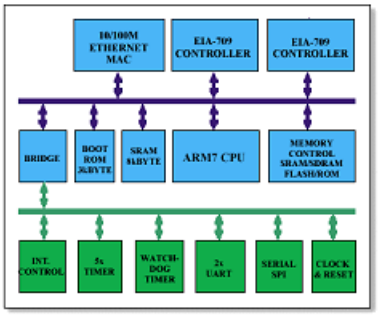
- Professional Ethernet & TCP/IP connectivity
- XML / SOAP Web Services compatibility
- Several types of BAS serial and I/O models
- Ideal for M2M & Facility to Enterprise connectivity
- Browser setup and user screens (web server)
- Professional electrical construction in a rugged metal housing

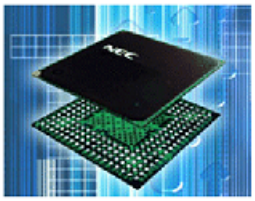
U+ series - Fieldbus Interface: USB to RS485 interface for BACnet MS/TP or Modbus RTU.

- USB to RS485 interface
- High speed co-processor handles time critical data and solves Windows real time response issues
- Both DB9 and screw terminals
- Professional electrical construction in a rugged metal housing
- Data and Power indication
- USB powered (no external power)

LISA - BACnet / EIA 709 Controller Chip: A powerful System on a Chip solution for creating BACnet or EIA 709 products.

- Highly integrated System on a Chip
- Supports BACnet/IP and MS/TP
- Supports ANSI/EIA 709 and 852
- 10/100 BaseT Ethernet MAC built in
- Two EIA 709 Network Controllers
- ARM7 RISC CPU running at 60 MHz
- 32 general purpose I/O ports
- 2 UARTS (1 full HS) to 3 Mbaud
- "unlimited" external RAM / FLASH
- Watchdog timer
- Boot ROM with 3k boot code
- Serial SPI interface
- Uses the leading 3rd party BACnet protocol stack (Cimetric's BACstac™)
- ANSI/EIA 709 and 852 LOYTEC ORION™ protocol stack
- SDK's have an Integrated Development Environment, RTEMS OS, TCP/IP stack, protocol analyzer, test programs, prototype PCB, more.





EasyBAC Module: This is a small module (an elongated RJ45 connector !) which contains a microprocessor, memory, and Ethernet to serial converter. Powerful 3rd generation BACstac™ software conforms to the BACnet/IP protocol on the Ethernet side, and implements a very simple serial protocol that your microprocesor software can easily perform ! Even the smallest microcontrollers should be capable of connecting to this module and become BACnet enabled. Layout a PCB that accepts this module, spend two weeks writing and testing your microcontroller software, and you have a BACnet product.

- Save significant development time - easily support BACnet/IP by adding minimal code
- After startup, host µP to interface communications is simply "object number, value"
- Analog, Binary, and Multi-state Object support
- BACnet code by the creators of the industry's leading BACnet OEM protocol stack (BACstac™)
- Easy browser setup (built-in web server)
- Available mounted in a stand-alone interface

EasyBAC Module
(build-in applications)

EasyBAC Interface
(stand-alone applications)

9.3 Contact Us - Support

Our full contact information is:

Cimetrics, Inc.
125 Summer Street, Suite 2100
Boston, MA 02110

tel: +1-617-350-7550
fax: +1-617-350-7552

info@cimetrics.com
www.cimetrics.com

Please look at our EasyBAC support **KNOWLEDGEBASE** and use our **HELP TICKET** system when you have questions. These are located at:

www.cimetrics.com/support

You can also send email to:

support@cimetrics.com

We would prefer that you use the HELP TICKET system because this gives us a record of what you asked and when so we can make sure that no questions get lost in an "email pile up". However, if you really need to speak to someone immediately call:

+1-617-350-7550

9.4 Ordering

Price quote: send an email to products@cimetrics.com or call 617-350-7550

Order: please **fax a purchase order to 617-350-7552** with the following information:

- EasyBAC products include the following
  - **B6090 - EasyBAC Development Kit**
  - **B6091 - EasyBAC module (built in applications)**
  - **B6095 - EasyBAC interface (stand-alone applications)**
- Your shipping and billing information

Payment options are credit card, bank deposit, or on account (USA only - credit verification required). We will contact you to complete the payment process.

More questions about purchasing are answered here.

Index

- P -

- R -

- S -

- U -

- V -

- W -