NIST Technical Note 2023

Programmers Guide to the BACnet Communications DLL

Michael A. Galler

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2023

NIST Technical Note 2023

Programmers Guide to the BACnet Communications DLL

Michael A. Galler
Energy and Environment Division
Engineering Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2023

October 2018

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Technical Note 2023
Natl. Inst. Stand. Technol. Tech. Note 2023, 20 pages (October 2018)
CODEN: NTNOEF

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2023

ABSTRACT

ASHRAE Standard 135- *BACnet®--A Data Communication Protocol for Building Automation and Control Networks* defines a communications protocol for information exchange between components of a distributed building automation and control system. The proliferation of BACnet enabled heating, ventilating, and air conditioning (HVAC) systems has enabled direct access to HVAC system data. The BACnet communications dynamic linked library (DLL) enables researchers to implement selected BACnet communications, creating an interface for data collection that is easy to use. The ability to access HVAC controllers from a computer is valuable because it allows researchers, designers, or other users to retrieve data directly from the BACnet objects on the controllers. These data could include information on the conditions in the building space served by the equipment, the conditions and status of the equipment, the actions the controller is currently taking, and other values that may be set by internal program logic. These data could then be used for a variety of research purposes, including real-time monitoring of the HVAC system, commissioning, fault detection and diagnostics (FDD), data logging, or other applications. Researchers can integrate the BACnet communications DLL (BCD) into existing tools, create a new tool with the BCD, or use the BCD with common data acquisition and analysis tools.

Table of Contents

1 Introduction

ASHRAE Standard 135- *BACnet®--A Data Communication Protocol for Building Automation and Control Networks* [1] defines a communication protocol for information exchange between components of a distributed building automation and control system. . The proliferation of BACnet enabled heating, ventilating, and air conditioning (HVAC) systems has enabled direct access to HVAC system data. The ability to access HVAC controllers from a computer is valuable because it allows researchers to retrieve data directly from the BACnet objects on the controllers. These data could include information on the conditions in the building space served by the equipment, the conditions and status of the equipment, the actions the controller is currently taking, and other values that may be set by internal program logic. These data could then be used for a variety of research purposes, including real-time monitoring of the HVAC system, commissioning, fault detection and diagnostics (FDD), data logging, or the application of other types of research tools.

The BACnet communications dynamic-linked library (DLL) is designed to simplify communications with BACnet-enabled controllers. It allows programs to read values from and write values to objects on local BACnet-enabled controllers, and to implement a range of BACnet commands. The BACnet Communications DLL (BCD) allows communications to any controller accessible from a computer by a local Ethernet or BACnet/IP (IPv4 only) connection. Since it is provided as a DLL, this capability can be integrated into research applications with minimal effort. The BCD can also be accessed by applications commonly used for data acquisition, which are also capable of communicating with a DLL. This allows the same communications code to be used by multiple applications for different purposes. The BCD is written in C++ and was developed using Microsoft Visual Studio 2015 and uses the WinPcap 4.1 library [2] for Ethernet communications. The BCD was designed to offer a high degree of functionality and stability, while being flexible and easy to use.

2 Usage

Communications with the BCD is achieved by calling one of a number of subroutines. Each subroutine is designed to perform a specific task with a simple, easy to use interface. The subroutines are divided into two categories: BACnet data transfer and configuration data transfer. BACnet data transfer involves reading data from or writing data to BACnet controllers. Configuration data transfer involves retrieving data about the network configuration, the controller configuration, or program configuration. The interface subroutines are listed in Table 2.1.

Table 2.1- BCS Subroutines

Transfer Type	Available Subroutines
BACnet Data	ReadBACnetValues WriteBACnetValues
Configuration Data	GetDevInfo BACnetObjectInfo WPCInit WPCClose GetDefaultAdapter SetDefaultAdapter GetAdapterMAC GetAdapterIP

To use the BCD, each subroutine must be declared in the code of the calling application using an `extern` statement. Example code is provided for each subroutine in the following sections.

Some of the data transfer subroutines will create messages on the BACnet network. The BACnet data transfer subroutines will initiate one or more BACnet ReadProperty, WriteProperty, ReadPropertyMultiple, or WritePropertyMultiple confirmed service messages to be sent to the appropriate controller. If a BACnet data transfer request only has one item, then a ReadProperty or WriteProperty message will be sent. If a BACnet data transfer request has multiple items, then a ReadPropertyMultiple or WritePropertyMultiple message will be sent. The GetDevInfo and WPCInit subroutines will transmit BACnet Who-Is unconfirmed service messages to the local Ethernet and BACnet/IP networks.

The BACnet data transfer subroutines use a data structure to hold device and router network information. This data structure must also be declared in the code of the calling application. This data structure and its use is described in the following sections.

2.1 Device Address Data Structure

2.1.1 Description

The BCD defines a data structure used to hold information about the network address of each device and router. These data structures are used by the BACnet data transfer subroutines to route requests. The data structure is defined in Table 2.1.1.

Table 2.1.1- The Device Address Data Structure

Variable Name	Description
mac_device_pr	Text version of the device media access control (MAC) address. This includes Ethernet and MS/TP addresses. This field must be filled.
mac_device	Optional hexadecimal version of the MAC address. It is implemented as an array of unsigned char with length 6, so it can hold either an Ethernet address or an MS/TP address.
mac_device_len	Optional length, in bytes, of the address entered in the mac_device field. This would be 6 for an Ethernet address or usually 2 for a MS/TP address.
mac_device_dec	Optional decimal version if a MS/TP MAC address is used.
ip	The IP address, if an UPD/IP address is being used. This is an array of length 4. If no IP address is being used, set to 0.0.0.0. IPv6 is not supported.
ip_port	The BACnet port, usually set to 0xBAC0.
ip_or_eth	Set to 1 if an IP address is being used, set to 0 if an Ethernet address is being used.
net	The BACnet network number.

Note that some of the fields are optional. When setting an IP address, the text version of the MAC address is required. The optional fields will be filled in by the BCD when passed into one of the BACnet data transfer subroutines.

2.1.2 Example Code

A sample implementation of the device address data structure is provided below. The names of the variables may be changed if required by local implementation. This example uses the names provided in Table 2.1.1. If changing the name of the structure (DevAddr in the example), the name must also be changed when declaring the BACnet data transfer subroutines.

Example 1: Sample implementation of the device address data structure

```
typedef struct tagdevAddr {
       unsigned char  mac_device[6];
       int                   mac_device_len;
       int                   mac_device_dec;
       char                  mac_device_pr[15];
       unsigned char         ip[4];
       int                   ip_port;
       char                  ip_or_eth;
       int                   net;
} DevAddr;

DevAddr myDev;
DevAddr myRouter;
```

Example 2: Device uses Ethernet address, Router not used

```
memset(&myDev, 0, sizeof(DevAddr));  // zero out all fields
strcpy(myDev.mac_device_pr, ”A1B2C3D4E5F6”);
myDev.ip_or_eth = 0;// set to use Ethernet
myDev.net = 42;
memset(&myRouter, 0, sizeof(DevAddr));      // no router
```

Example 3: Device uses Ethernet address, Router uses IP address

```
memset(&myDev, 0, sizeof(DevAddr));  // zero out all fields
myDev.ip_or_eth = 0;// set to use Ethernet
strcpy(myDev.mac_device_pr, ”0A”);
myDev.net = 42;

memset(&myRouter, 0, sizeof(DevAddr));      // zero out all fields
strcpy(myRouter.mac_device_pr, ”A1B2C3D4E5F6”);
myRouter.ip[0] = 192;
myRouter.ip[1] = 168;
myRouter.ip[2] = 0;
myRouter.ip[3] = 1;
myRouter.ip_port = 0xBAC0;
myRouter.ip_or_eth = 1;// set to use BACnet/IP
myRouter.net = 10;
```

2.2 The ReadBACnetValues Subroutine

2.2.1 Description

Requests for retrieval of the values of properties of BACnet objects are implemented by calling the ReadBACnetValues subroutine. Multiple values (up to 250) can be queried with one call to the subroutine. All queries must be directed to the same BACnet device. Any data that is from a supported object type as shown in Table 2.2.2, and is of type REAL, Unsigned Integer, BACnetBinaryPV, CharacterString or BOOLEAN, can be retrieved.

2.2.2 Declaration

The declaration in the calling program for the ReadBACnetValues subroutine:

```
extern "C" __declspec(dllimport) int ReadBACnetValues(int count, int *obj, int *inst,
int *prop, char **bv, int *err, DevAddr *dev, DevAddr *rtr);
```

2.2.3 Arguments and Return Value

The arguments used in the ReadBACnetValues subroutine are described in Table 2.2.1.

Table 2.2.1- Explanation of Arguments to ReadBACnetValues

Argument	Description
count	An integer in the range 1 to 250 used to indicate the number of values in the request and the size of the remaining array parameters.
obj	An array of integers representing the BACnet object type of a query (see Table 2.2.2). The values correspond to the values in the BACnetObjectType enumeration.
inst	An array of integers representing the instance number of the BACnet object being queried.
prop	An array of integers representing the BACnetPropertyIdentifier being read.
bv	An array of strings containing the response to the read requests (or an error message if there was an error for a specific request). Each string should be allocated to a length of 20 or larger.
err	An array of integers representing the error status of each request, 1 for no error or 0 for an error.
dev	A DevAddr structure configured with the network information for the BACnet device being queried.
rtr	A DevAddr structure configured with the network information for the router to the BACnet device being queried, if a router is required.

Table 2.2.2- Valid Values For obj Arguments for the ReadBACnetValue Subroutine

Object Type	Argument Value
Analog Input (AI)	0
Analog Output (AO)	1
Analog Value (AV)	2
Binary Input (BI)	3
Binary Output (BO)	4
Binary Value (BV)	5
Multi-State Input (MI)	13
Multi-State Output (MO)	14
Multi-State Value (MV)	19

The return values for ReadBACnetValues are described in Table 2.2.3. Note that if any of the values can not be read, the return value will indicate an error. Other values will be returned as available. The err array must be examined to determine which return values are valid.

Table 2.2.3- Return Values for the ReadBACnetValues Subroutine

Return Value	Description
1	No errors returned.
0	One or more errors returned. Examine the err array to determine which request returned an error.

2.2.4 Example Code

Below are samples of code showing how this subroutine might be implemented. This code is provided for demonstration purposes only.

Example 1: Request single value:

```
DevAddr myDev;
DevAddr myRouter;
// Not shown: myDev and myRouter must be configured properly
int obj[1] = { 1 }; // Analog Output
int inst[1] = { 5 }; // Instance number 5 (AO5)
int prop[1] = { 85 }; // present-value property
char **bv;  // must be allocated, code not shown
int err[1];

int i = ReadBACnetValues(1, obj, inst, prop, bv, err, &myDev, &myRouter);
// Not shown: check the return value of i
```

Example 2: Request multiple values:

```
DevAddr myDev;
DevAddr myRouter;
// Not shown: myDev and myRouter must be configured properly
int obj[4] = { 1, 2, 1, 2 }; // Analog Outputs and Analog Values
int inst[4] = { 5, 6, 5, 6 }; // Instance numbers (AO5, AV6, AO5, AV6)
int prop[4] = { 81, 81, 85, 85 }; // out-of-service and present-value
char **bv; // must be allocated, code not shown
int err[4];

int i = ReadBACnetValues(4, obj, inst, prop, bv, err, &myDev, &myRouter);
// Not shown: check the return value of i and the value stored in bv.
```

2.3 The WriteBACnetValues Subroutine

2.3.1 Description

Requests for writing the values of properties to BACnet objects are implemented by calling the WriteBACnetValues subroutine. Multiple values (up to 250) can be sent with one call to the subroutine. All writes must be directed to the same BACnet device. Any data that is of a supported object type as shown in Table 2.2.2, and is of type REAL, Unsigned Integer, BACnetBinaryPV, CharacterString or BOOLEAN, can be sent. Values can be written with a priority by adding a semicolon and the priority value after the property value.

2.3.2 Declaration

The declaration in the calling program for the WriteBACnetValues subroutine:

```
extern "C" __declspec(dllimport) int WriteBACnetValues(int count, int *obj, int *inst,
int *prop, char **bv, int *err, DevAddr *dev, DevAddr *rtr);
```

2.3.3 Arguments and Return Value

The arguments used in the WriteBACnetValues subroutine are described in Table 2.3.1.

Table 2.3.1- Explanation of Arguments to WriteBACnetValues

Argument	Description
count	An integer in the range 1 to 250 used to indicate the number of values in the request and the size of the remaining array parameters.
obj	An array of integers representing the BACnet object type of a write request (see Table 2.2.2).
inst	An array of integers representing the instance number of the BACnet object being written to.
prop	An array of integers representing the BACnetPropertyIdentifier being written.
bv	An array of strings containing the response to the read requests (or an error message if there was an error for a specific request). Each string should be allocated to a length of 20 or larger.
err	An array of integers representing the error status of each request, 1 for no error or 0 for an error.
Dev	A DevAddr structure configured with the network information for the BACnet device being written to.
Rtr	A DevAddr structure configured with the network information for the router to the BACnet device being written to, if a router is required.

The return values for WriteBACnetValues are described in Table 2.3.2. Note that a return value indicating an error does not mean that no values were written, but that one or more values could not be written.

Table 2.3.2- Return Values for the WriteBACnetValues Subroutine

Return Value	Description
1	No errors returned.
0	One or more errors returned. Examine the err array to determine which request returned an error.

2.3.4 Example Code

Below are sample implementations of the WriteBACnetValues subroutine. This code is provided for demonstration purposes only.

Example 1: Write a single value with priority level 8:

```
DevAddr myDev;
DevAddr myRouter;
// Not shown: myDev and myRouter must be configured properly
int obj[1] = { 0 }; // Analog Input
int inst[1] = { 5 }; // Instance number 5 (AI5)
int prop[1] = { 85 }; // present-value property
char **bv; // must be allocated, code not shown
int err[1];
float overrideTemp = 75;

sprintf(bv[0], ”%5.2f:8”,overrideTemp);

int i = WriteBACnetValues(1, obj, inst, prop, bv, err, &myDev, &myRouter);
// Not shown: check the return value of i
```

Example 2: Write multiple values:

```
DevAddr myDev;
DevAddr myRouter;
// Not shown: myDev and myRouter must be configured properly
int obj[4] = { 1, 2, 1, 2 }; // Analog Outputs and Analog Values
int inst[4] = { 5, 6, 5, 6 }; // Instance numbers (AO5, AV6, AO5, AV6)
int prop[4] = { 81, 81, 85, 85 }; // out-of-service and present-value
char **bv; // must be allocated, code not shown
int err[4];
float newValues[4] = { 1, 1, 0.2, 55 };

for(int iter=0; iter<4; iter++) sprintf(bv[iter],”5.2f”,newValues);

int i = WriteBACnetValues(4, obj, inst, prop, bv, err, &myDev, &rtr);
// Not shown: check the return value of i
```

2.4 The GetDevInfo Subroutine

2.4.1 Description

This subroutine is used to get information about the network addressing of BACnet controllers on the local network. This subroutine will transmit BACnet Who-Is unconfirmed service messages to the local Ethernet and BACnet/IP networks. The network information of responding controllers will be passed back from this subroutine. Note that the BCD begins collecting this information when the subroutine WPCInit is first called. This network information can then be used in subsequent calls to the BACnet data transfer subroutines, or other subroutines requiring network information.

2.4.2 Declaration

The declaration in the calling program for the GetDevInfo subroutine:

```
extern "C" __declspec(dllimport) int GetDevInfo(int maxcount, DevAddr *dev, DevAddr
*rtr);
```

2.4.3 Arguments and Return Value

The arguments to GetDevInfo are described in Table 2.4.1.

Table 2.4.1- GetDevInfo Subroutine Arguments

Argument	Description
maxcount	The maximum number of devices the dev and rtr arrays can hold. This should be sized to the requirements of the local network.
dev	An array of DevAddr of length maxcount. This will hold device configuration data. Valid returned values can be used with the BACnet data transfer subroutines, or other subroutines requiring network information.
rtr	An array of DevAddr of length maxcount. This will hold router configuration data. Valid returned values can be used with the BACnet data transfer subroutines, or other subroutines requiring network information.

Table 2.4.2- Return Values for the GetDevInfo Subroutine

Return Value	Description
≥ 1	The number of BACnet controllers found.
0	No controllers found.
< 0	There was an error. Examine the input parameters.

2.4.4 Example Code

Below is a sample implementation of the GetDevInfo subroutine. This code is provided for demonstration purposes only.

Example 1: Read local network information:

```
DevAddr dev[10];
DevAddr rtr[10];

int i = GetDevInfo(10, dev, rtr);
// Not shown: check the return value of i and use dev, rtr if valid
```

2.5 The BACnetObjectInfo Subroutine

2.5.1 Description

This subroutine is used to obtain information about BACnet object types supported by the BCD and their properties. It returns information on the set of objects and properties supported by the current version of the BCD. This is useful to ensure compatibility, as the BCD does not support every possible BACnet object type and property. Note that future versions of the BCD may have support for draft versions of new BACnet objects, to help fulfill its mission as a research tool. The information that is available includes the number of BACnet object types, the names of the BACnet objects, and the count and names of the properties of each object type. Note that this subroutine does not give information or values from any controllers on the network, or from any specific instance of an object or property on a controller. Use of this subroutine does not generate any messages on the network or rely on information previously gathered from the network. Information about BACnet objects located on a specific controller can instead be obtained using the ReadBACnetValues subroutine.

2.5.2 Declaration

The declaration in the calling program for the BACnetObjectInfo subroutine:

```
extern "C" __declspec(dllimport) int BACnetObjectInfo (int reqObject, int reqProp, char
*propName);
```

2.5.3 Arguments and Return Value

The arguments to BACnetObjectInfo are described in Table 2.5.1.

Table 2.5.1- BACnetObjectInfo Subroutine Arguments

Argument name	Value	Description
reqObject	< 0	Returns the number of BACnet objects.
	≥ 0	If reqProp is < 0, returns the number of properties for the indicated object, and copies the text name of the indicated object to propName. If reqProp is > 0, returns the value 1 and copies the text name of the indicated property to propName. Note that the numbering for BACnet objects starts at 0. If there are N objects, the last one is specified by the value N-1. This is also valid for properties. This is standard for C and C++ programming. If reqObject is a number greater than the number of BACnet objects, an error value of -1 will be returned.
reqProp	< 0	Only evaluated if reqObject refers to a valid object. Returns property count for reqObject.
	≥ 0	Only evaluated if reqObject is set to a valid object. Returns text label of reqProp in the propName parameter. If reqProp is a number greater than the number of properties for the indicated BACnet object, an error value of -2 will be returned.
propName		This variable is only used to return text labels of objects or properties. It should be initialized as an array of characters at least 40 characters long.

The meaning of the return value depends on which arguments are used. If the reqObject parameter is negative, the return value will be the number of BACnet object types supported. If reqObject is larger than the number of BACnet object types supported, an error value of -1 will be returned. If reqObject is valid and reqProp is negative, the return value will be the number of properties supported by the BACnet object indicated by reqObject. If reqProp is a valid property number, the property name requested will be copied to propName and a value of 1 will be returned. If reqProp is larger than the valid range, an error value of -2 will be returned. The return values are described in Table 2.5.2.

Table 2.5.2- Return Values for the BACnetObjectInfo Subroutine

Return Value	Description
> 0	The request was returned successfully.
-1	There was an error with the reqObject parameter.
-2	There was an error with the reqProp parameter.

2.5.4 Example Code

Below are sample implementations of the BACnetObjectInfo subroutine. This code is provided for demonstration purposes only.

Example 1: Retrieve the number of BACnet object types supported:

```
char opname[40];

int i = BACnetObjectInfo(-1,0,opname);
// note that the value of reqProp will not be evaluated
printf("There are %d BACnet object types supported.",i);
```

Example 2: Retrieve the name of, and the number of properties supported by BACnet, object type 0:

```
// Note: BACnet object type 0 is analog-input
i = BACnetObjectInfo(0,-1,opname);
//NOTE- opname == "analog-input"
printf("BACnet object type 0 is named %s",opname);
printf("BACnet object type 0 has %d properties.",i);
```

Example 3: Retrieve the name of a specific property of BACnet object type 0 (analog-input):

```
i = BACnetObjectInfo(0, 5, opname);
// Note: opname == "status-flags"
printf("Property 5 of BACnet object type 0 is labeled '%s'.",opname);
```

2.6 The WPCInit and WPCClose Subroutines

2.6.1 Description

The WPCInit subroutine is used to explicitly initialize the WinPCap connection. If this function is not called explicitly by the calling program, it will be called by the first BACnet data transfer subroutine called. The benefit of calling this subroutine explicitly is that when called it will return the number of network adapters present on the computer. If there are multiple adapters present, it will also select the first one with a valid Ethernet address as the default adapter. Note that it will automatically skip modems and some virtual network adapters. A virtual network adapter can be set as the default adapter by calling the SetDefaultAdapter subroutine with the appropriate adapter identifier. If a user experiences network connectivity problems, this subroutine and the GetDefaultAdapter and SetDefaultAdapter subroutines could be used to investigate and solve them.

The WPCClose subroutine is used to close the WinPCap connections. It will be called automatically by the BCD when it closes, but it may also be called explicitly. This might be useful if a program does not need to send or receive further BACnet packets, but has other processing to finish.

2.6.2 Declaration

The declaration in the calling program for the WPCInit subroutine:

```
extern "C" __declspec(dllimport) int WPCInit(void);
```

The declaration in the calling program for the WPCClose subroutine:

```
extern "C" __declspec(dllimport) void WPCClose(void);
```

2.6.3 Arguments and Return Value

There are no arguments passed to the WPCInit subroutine. The return values are described in Table 2.7.1.

Table 2.7.1- Return Values for the WPCInit Subroutine

Return Value	Description
< 0	An error occurred with WinPCap.
≥ 0	The count of adapters found on the system.

There are no arguments to or return values from the WPCClose subroutine.

2.6.4 Example Code

Below is a sample implementation of the WPCInit and WPCClose subroutines. This code is provided for demonstration purposes only.

Example 1: Initialize WinPCap and get the number of adapters:

```
int count = WPCInit();
if(count < 0){        printf(“There was an error initializing WinPCap.\n”);
} else {      printf(“There are %d adapters on this computer\n”, count);}
/* ...Note: your program goes here... */
WPCClose();
```

2.7 The GetDefaultAdapter Subroutine

2.7.1 Description

The GetDefaultAdapter subroutine is used to get the identifier for the current default adapter. Note that numbering starts at 0.

2.7.2 Declaration

The declaration in the calling program for the GetDefaultAdapter subroutine:

```
extern "C" __declspec(dllimport) int GetDefaultAdapter(void);
```

2.7.3 Arguments and Return Value

There are no arguments passed to GetDefaultAdapter. The return values are described in Table 2.8.1.

Table 2.8.1- Return Values for the GetDefaultAdapter Subroutine

Return Value	Description
< 0	The default adapter has not been set.
≥ 0	The identifier of the default adapter.

2.7.4 Example Code

Below is a sample implementation of the WinPCap subroutine. This code is provided for demonstration purposes only.

Example 1: Initialize WinPCap and get default adapter identifier:

```
int count = WPCInit();

if(count < 0){
printf(“There was an error initializing WinPCap.\n”);
} else {
       printf(“There are %d adapters on this computer\n”, count);
}

Int adapter_id = GetDefaultAdapter( );
printf(“The default adapter ID is %d\n”, adapter_id);
// Note that adapter_id will be 0 if there is only one adapter
```

2.8 The SetDefaultAdapter Subroutine

2.8.1 Description

The SetDefaultAdapter subroutine is used to select the identifier for the current default adapter. Note that numbering starts at 0.

2.8.2 Declaration

The declaration in the calling program for the SetDefaultAdapter subroutine:

```
extern "C" __declspec(dllimport) int SetDefaultAdapter(int i);
```

2.8.3 Arguments and Return Value

There is one argument passed to SetDefaultAdapter, which is the ID of the new default adapter. This must be a valid adapter ID, which is a nonnegative integer, and less than the adapter count as returned by WPCInit. The return values are described in Table 2.9.1.

Table 2.9.1- Return Values for the SetDefaultAdapter Subroutine

Return Value	Description
0	There was an error setting the default adapter.
1	The default adapter was set to the new value.

2.8.4 Example Code

Below is a sample implementation of the SetDefaultAdapter subroutine. This code is provided for demonstration purposes only.

<u>Example 1: Initialize WinPCap and set default adapter ID</u>

```
int count;
int adapter_id;
int status;

count = WPCInit();

if(count < 0){
printf("There was an error initializing WinPCap.\n");
} else {
       printf("There are %d adapters on this computer\n", count);
}

adapter_id = GetDefaultAdapter( );
printf("The default adapter ID is %d\n", adapter_id);
// Note that adapter_id will be 0 if there is only one adapter
// For this example, let count=3, and the default adapter is 0
status = SetDefaultAdapter(1);
if(status == 1){
printf("The default adapter was changed successfully\n");
} else {
printf("There was an error setting the default adapter\n");
}
```

2.9 The GetAdapterMAC Subroutine

2.9.1 Description

The GetAdapterMAC subroutine is used to obtain the MAC address of a network adapter.

2.9.2 Declaration

The declaration in the calling program for the GetAdapterMAC subroutine:

```
extern "C" __declspec(dllimport) int GetAdapterMAC(int id, unsigned char *label);
```

2.9.3 Arguments and Return Value

The GetAdapterMAC subroutine requires two parameters. The first parameter specifies which adapter is being queried. If this parameter is passed as -1, then the default adapter is used. The second parameter is an output parameter and will have the MAC address of the adapter copied to it by the GetAdapterMAC subroutine. There must be enough space allocated for a MAC address to be copied to this parameter. Note that the MAC address type is unsigned char, not in human readable format, and will only require a string length of 6 unsigned chars to be allocated. The return values for the GetAdapterMAC subroutine are summarized in Table 2.10.1.

Table 2.10.1- Return Values for the GetAdapterMAC Subroutine

Return Value	Description
0	There was an error with an input parameter: the adapter id was out of range, or the label parameter was not allocated.
1	No error- the MAC address was copied to the output parameter.

2.9.4 Example Code

Below is a sample implementation of the GetAdapterMAC subroutine. This code is provided for demonstration purposes only.

Example 1: Initialize WinPCap and get the MAC address for the default adapter:

```
int count, adapter_id, status;
unsigned char label[6];

count = WPCInit();
if(count < 0){
printf(“There was an error initializing WinPCap.\n”);
} else {
       printf(“There are %d adapters on this computer\n”, count);
}

adapter_id = GetDefaultAdapter( );
printf(“The default adapter ID is %d\n”, adapter_id);
// Note the adapter ID will be 0 if there is only one adapter
status = GetAdapterMAC(adapter_id, &label);
if(status == 1)       printf(“The adapter MAC address was retrieved\n”);
else   printf(“There was an error getting the adapter MAC address.\n”);
```

2.10 The GetAdapterIP Subroutine

2.10.1 Description

The GetAdapterIP subroutine is used to obtain the IP address of a network adapter.

2.10.2 Declaration

The declaration in the calling program for the GetAdapterIP subroutine:

```
extern "C" __declspec(dllimport) int GetAdapterIP(int id, unsigned char *label);
```

2.10.3 Arguments and Return Value

The GetAdapterIP subroutine requires two parameters. The first parameter specifies which adapter is being queried. If this parameter is passed as -1, then the default adapter is used. The second parameter is an output parameter and will have the IP address of the adapter copied to it by the GetAdapterIP subroutine. There must be enough space allocated for a IP address to be copied to this parameter. Note that the IP address type is unsigned char, not in human readable format, and will only require a string length of 4 unsigned chars to be allocated. The return values for the GetAdapterIP subroutine are summarized in Table 2.10.1.

Table 2.10.1- Return Values for the GetAdapterIP Subroutine

Return Value	Description
0	There was an error with an input parameter: the adapter id was out of range, or the label parameter was not allocated.
1	No error- the IP address was copied to the output parameter.

2.10.4 Example Code

Below is a sample implementation of the GetAdapterIP subroutine. This code is provided for demonstration purposes only.

Example 1: Initialize WinPCap and get the IP address for the default adapter:

```
int count, adapter_id, status;
unsigned char label[4];

count = WPCInit();
if(count < 0){
printf(“There was an error initializing WinPCap.\n”);
} else {
       printf(“There are %d adapters on this computer\n”, count);
}

adapter_id = GetDefaultAdapter();
printf(“The default adapter ID is %d\n”, adapter_id);
// Note adapter_id will be 0 if there is only one adapter
status = GetAdapterIP(adapter_id, &label);
if(status == 1) printf(“The adapter IP address was retrieved\n”);
else printf(“There was an error retrieving the adapter IP address.\n”);
```

3 Summary

The BACnet Communications DLL enables programmers of HVAC-related computer programs to easily incorporate BACnet communications into their tools. It supports a command set that enables the most common types of communications used in data acquisition, monitoring, FDD, or other activities that require communicating with a BACnet enabled device. Use of the BACnet DLL removes a significant barrier for users who wish to write tools but do not have the programming experience to implement the BACnet protocol. The BACnet DLL is currently incorporated into several applications created at the National Institutes of Standards and Technology (NIST), and which are used both on the NIST campus and at other locations.

4 Future Work

While the BCD is already capable of being useful in a wide variety of applications, as BACnet expands there will be more applications where the BCD can be used. Some planned extensions of the BCD are inclusion of a Web Services interface, addition of new BACnet objects and properties, and extensions of the simplified interface to include new functionality.

5 References

[1] ASHRAE, ANSI/ASHRAE 135-2016, *BACnet: A Data Communication Protocol for Building Automation and Control Networks.* American Society of Heating, Refrigerating, and Air-Conditioning Engineers Inc. Atlanta, GA.
[2] http://www.winpcap.org