CASBACnetStack - Virtual Devices

Introduction

This document walks through the BACnet Virtual Devices Server Example CPP project found here: https://github.com/chipkin/BACnetVirtualDevicesServerExampleCPP

This project displays how to use the virtual network functions in the CAS BACnet Stack to setup one or more virtual networks and devices.

The example contains a simple memory database to store values used by the example. See CASBACnetStackExampleDatabase for more information.

The rest of this document will go through the BACnetVirtualDevicesServerExampleCPP.cpp file and describe the various sections for implementing Virtual Devices.

Implementation

This section describes the BACnetVirtualDevicesServerExampleCPP.cpp implementation for using the CAS BACnet Stack to create virtual BACnet devices on virtual BACnet networks.

Registering Required Callback Functions

The following code snippet shows the callback functions that will be used for this example. Please note that depending on what objects the virtual devices contain additional callback functions may be required. Please refer to the main CAS BACnet Stack quick start for more information on callback functions.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 98-114
// 3. Setup the callbacks.
// ------------------------------------------------------------------------
std::cout << "FYI: Registering the callback Functions with the CAS BACnet Stack"
<< std::endl;

// Message Callback Functions
fpRegisterCallbackReceiveMessage(CallbackReceiveMessage);
fpRegisterCallbackSendMessage(CallbackSendMessage);

// System Time Callback Functions
fpRegisterCallbackGetSystemTime(CallbackGetSystemTime);

// Get Property Callback Functions
fpRegisterCallbackGetPropertyCharacterString(CallbackGetPropertyCharString);
fpRegisterCallbackGetPropertyEnumerated(CallbackGetPropertyEnum);
fpRegisterCallbackGetPropertyOctetString(CallbackGetPropertyOctetString);
fpRegisterCallbackGetPropertyReal(CallbackGetPropertyReal);
fpRegisterCallbackGetPropertyUnsignedInteger(CallbackGetPropertyUInt);
```

See the Callback Section for a breakdown of the individual callback functions.

Setting up the Main Device

The main device that gets added using the CAS BACnet Stack AddDevice function will act as the virtual router. The next code snippet adds the device, enables any BACnet services needed, and enables any optional device properties.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 116-145
// 4. Setup the BACnet device.
// ------------------------------------------------------------------------
```

```
std::cout << "Setting up main server device. device.instance=[" <<
g_database.mainDevice.instance << "]" << std::endl;

// Create the Main Device
if (!fpAddDevice(g_database.mainDevice.instance)) {
    std::cerr << "Failed to add Device." << std::endl;
    return false;
}
std::cout << "Created Device." << std::endl;

// Enable the services that this device supports
// Some services are mandatory for BACnet devices and are already enabled.
// These are: Read Property, Who Is, Who Has
//
// Any other services need to be enabled as below.

std::cout << "Enabling ReadPropertyMultiple... ";
if (!fpSetServiceEnabled(g_database.mainDevice.instance,
CASBACnetStackExampleConstants::SERVICE_READ_PROPERTY_MULTIPLE, true)) {
    std::cerr << "Failed to enabled the ReadPropertyMultiple" << std::endl;
    return -1;
}
std::cout << "OK" << std::endl;

// Enable Optional Device Properties
if (!fpSetPropertyEnabled(g_database.mainDevice.instance,
CASBACnetStackExampleConstants::OBJECT_TYPE_DEVICE,
g_database.mainDevice.instance,
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_DESCRIPTION, true)) {
    std::cerr << "Failed to enable the description property for the Main Device"
<< std::endl;
    return false;
}
```

Then any objects for the virtual router are added. In this example, only the Network Port object that represents the Ipv4 BACnet IP port is added.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 147-156
```

```
// Add Main Device Objects
// ------------------------------------

// Add the Network Port Object
std::cout << "Added NetworkPort. networkPort.instance=[" <<
g_database.networkPort.instance << "]... ";
if (!fpAddNetworkPortObject(g_database.mainDevice.instance,
g_database.networkPort.instance,
CASBACnetStackExampleConstants::NETWORK_TYPE_IPV4,
CASBACnetStackExampleConstants::PROTOCOL_LEVEL_BACNET_APPLICATION,
CASBACnetStackExampleConstants::NETWORK_PORT_LOWEST_PROTOCOL_LAYER)) {
    std::cerr << "Failed to add NetworkPort" << std::endl;
    return -1;
}
std::cout << "OK" << std::endl;
```

Please note that all constant values can be found in CASBACnetStackExampleConstants.h

Add Virtual Network, Virtual Devices, and Objects in the Virtual Devices

Once the main device (the virtual router) has been added, you can now add virtual networks and virtual devices and objects.

The next code snippet shows what functions to call, and the order to call them. In this example, the example database contains all the information for the virtual networks and devices in a std::map. To add them to the CAS BACnet Stack, we iterate through the map to first add the virtual network, then add the virtual device, then finally add any objects and properties to the virtual device.

Here is the entire code snippet for this section. Each individual component will be discussed further below.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 158-196
// Add Virtual Devices and Objects
std::cout << "Adding Virtual Devices and Objects..." << std::endl;
std::map<uint16_t, std::vector<ExampleDatabaseDevice> >::iterator it;
for (it = g_database.virtualDevices.begin(); it !=
g_database.virtualDevices.end(); ++it) {
```

```
    // Add the Virtual network
    if (!fpAddVirtualNetwork(g_database.mainDevice.instance, it->first, it->first
* 10)) {
        std::cerr << "Failed to add virtual network " << it->first << std::endl;
        return -1;
    }

    std::vector<ExampleDatabaseDevice>::iterator devIt;
    for (devIt = it->second.begin(); devIt != it->second.end(); ++devIt) {
        // Add the Virtual Device
        std::cout << "Adding Virtual Device. device.instance=[" <<
devIt->instance << "] to network=[" << it->first << "]...";
        if (!fpAddDeviceToVirtualNetwork(devIt->instance, it->first)) {
            std::cerr << "Failed to add Virtual Device" << std::endl;
            return -1;
        }
        std::cout << "OK" << std::endl;

        // Enable Read Property Multiple
        if (!fpSetServiceEnabled(devIt->instance,
CASBACnetStackExampleConstants::SERVICE_READ_PROPERTY_MULTIPLE, true)) {
            std::cerr << "Failed to enable the ReadPropertyMultiple" <<
std::endl;
            return -1;
        }
        std::cout << "OK" << std::endl;

        // Add the Analog Input to the Virtual Device
        std::cout << "Adding Analog Input to Virtual Device. device.instance=["
<< devIt->instance << "], analogInput.instance=[" <<
g_database.analogInputs[devIt->instance].instance << "]...";
        if (!fpAddObject(devIt->instance,
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT,
g_database.analogInputs[devIt->instance].instance)) {
            std::cerr << "Failed to add AnalogInput" << std::endl;
            return -1;
        }
```

```
        std::cout << "OK" << std::endl;

        // Enable Reliability property
        fpSetPropertyByObjectTypeEnabled(devIt->instance,
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT,
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_RELIABILITY, true);
    }
}
```

Breaking down the code snippet above, for each virtual network that was setup in the example database we do the following:

1. Add the Virtual Network

First, we add the virtual network. This will add the network to the CAS BACnet Stack as a virtual network for the virtual router. It will also create in the main device a Network Port object that represents the virtual network. In this example, we assign the object instance of the Network Port Object to be the network * 10.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 162-166
// Add the Virtual network
if (!fpAddVirtualNetwork(g_database.mainDevice.instance, it->first, it->first *
10)) {
    std::cerr << "Failed to add virtual network " << it->first << std::endl;
    return -1;
}
```

2. Add Virtual Devices and Objects

Once the virtual network has been added, we can now add virtual devices and their objects and properties. In this example, we iterate through a vector of virtual devices and add them one at a time to the virtual network. For each device, we enable the services the virtual devices should support, add their objects and enable any properties that they should support. When adding the virtual device using the function AddDeviceToVirtualNetwork, a Network Port object is automatically added to the virtual device.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 168-195
std::vector<ExampleDatabaseDevice>::iterator devIt;
for (devIt = it->second.begin(); devIt != it->second.end(); ++devIt) {
    // Add the Virtual Device
```

```
    std::cout << "Adding Virtual Device. device.instance=[" << devIt->instance <<
"] to network=[" << it->first << "]...";
    if (!fpAddDeviceToVirtualNetwork(devIt->instance, it->first)) {
        std::cerr << "Failed to add Virtual Device" << std::endl;
        return -1;
    }
    std::cout << "OK" << std::endl;

    // Enable Read Property Multiple
    if (!fpSetServiceEnabled(devIt->instance,
CASBACnetStackExampleConstants::SERVICE_READ_PROPERTY_MULTIPLE, true)) {
         std::cerr << "Failed to enable the ReadPropertyMultiple" << std::endl;
         return -1;
    }
    std::cout << "OK" << std::endl;

    // Add the Analog Input to the Virtual Device
    std::cout << "Adding Analog Input to Virtual Device. device.instance=[" <<
devIt->instance << "], analogInput.instance=[" <<
g_database.analogInputs[devIt->instance].instance << "]...";
    if (!fpAddObject(devIt->instance,
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT,
g_database.analogInputs[devIt->instance].instance)) {
        std::cerr << "Failed to add AnalogInput" << std::endl;
        return -1;
    }
    std::cout << "OK" << std::endl;

    // Enable Reliability property
    fpSetPropertyByObjectTypeEnabled(devIt->instance,
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT,
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_RELIABILITY, true);
}
```

Send IAms and IAmRouterToNetwork

Once the setup of the virtual router and the virtual devices has been complete, send an IAm for the main device (virtual router) and for all the virtual devices. Then finally send an

IAmRouterToNetwork that describes what virtual networks have been added. All of these messages are broadcasted.

First, prepare the connectionString of where to send the broadcasted messages.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 198-205
// 5. Send I-Am of this device
// ------------------------------------------------------------------------
// To be a good citizen on a BACnet network. We should announce ourselves when we
start up.
std::cout << "FYI: Sending I-AM broadcast" << std::endl;
uint8_t connectionString[6]; //= { 0xC0, 0xA8, 0x01, 0xFF, 0xBA, 0xC0 };
memcpy(connectionString, g_database.networkPort.BroadcastIPAddress, 4);
connectionString[4] = g_database.networkPort.BACnetIPUDPPort / 256;
connectionString[5] = g_database.networkPort.BACnetIPUDPPort % 256;
```

Then, send the IAm message for the main device (virtual router)

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 207-211
// Send IAm for the Main Device
if (!fpSendIAm(g_database.mainDevice.instance, connectionString, 6,
CASBACnetStackExampleConstants::NETWORK_TYPE_IP, true, 65535, NULL, 0)) {
    std::cerr << "Unable to send IAm broadcast for mainDevice.instance=[" <<
g_database.mainDevice.instance << "]" << std::endl;
    return false;
}
```

Next, send the IAm messages for each of the virtual devices. Depending on how many virtual devices, you may want to delay sending some of the IAm messages to ease network traffic. From the spec:

> **H.2.2.2 Spread Out I-Am, I-Have Requests**
> The gateway device is responsible for sending the I-Am and I-Have requests for the non-BACnet devices. When the number of
> non-BACnet devices increases, so too will the number of I-Am and I-Have requests that the gateway will be sending. In order to
> ease network traffic, the gateway should space out I-Am and I-Have requests.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 213-222
// Send IAm for each virtual device
```

```
for (it = g_database.virtualDevices.begin(); it !=
g_database.virtualDevices.end(); ++it) {
    std::vector<ExampleDatabaseDevice>::iterator devIt;
    for (devIt = it->second.begin(); devIt != it->second.end(); ++devIt) {
        if (!fpSendIAm(devIt->instance, connectionString, 6,
CASBACnetStackExampleConstants::NETWORK_TYPE_IP, true, 65535, NULL, 0)) {
            std::cerr << "Unable to send IAm broadcast for
virtualDevice.instance=[" << devIt->instance << "]" << std::endl;
            return false;
        }
    }
}
```

Finally, send an IAmRouterToNetwork message that tells other BACnet routers on the network that the virtual networks are available.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 224-228
// Send IAmRouterToNetwork
if (!fpSendIAmRouterToNetwork(connectionString, 6,
CASBACnetStackExampleConstants::NETWORK_TYPE_IP, true, 65535, NULL, 0)) {
    std::cerr << "Unable to send IAmRouterToNetwork broadcast" << std::endl;
    return false;
}
```

Callback Functions

Implementing the Callback functions for virtual devices is basically the same way for a regular device. The only difference is using the deviceInstance parameter to determine from which virtual device the value is being requested. This section describes the Callback functions used in the example and highlights the parts that are specific for virtual devices.

The CallbackReceiveMessage, CallbackSendMessage, and CallbackGetSystemTime functions are all similar implementations for a standard BACnet device, so this document will focus on the CallbackGetProperty callbacks.

CallbackGetPropertyCharString

The CallbackGetPropertyCharString handles getting any character string properties. In this example this callback is getting the ObjectName and Description properties. The two helper

functions facilitate retrieving or creating the ObjectName based on the deviceInstance, objectType, and objectInstance properties.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 408-419
// Callback used by the BACnet Stack to get Character String property values from
the user
bool CallbackGetPropertyCharString(const uint32_t deviceInstance, const uint16_t
objectType, const uint32_t objectInstance, const uint32_t propertyIdentifier,
char* value, uint32_t* valueElementCount, const uint32_t maxElementCount,
uint8_t* encodingType, const bool useArrayIndex, const uint32_t
propertyArrayIndex)
{
    // Example of Object Name property
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_OBJECT_NAME) {
        return GetObjectName(deviceInstance, objectType, objectInstance, value,
valueElementCount, maxElementCount);
    }
    // Example of Device Description
    else if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_DESCRIPTION && objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_DEVICE) {
        return GetDeviceDescription(deviceInstance, value, valueElementCount,
maxElementCount);
    }
    return false;
}
```

CallbackGetPropertyEnum

The CallbackGetPropertyEnum handles getting any enumerated properties. In this example, the callback gets the Reliability and System Status properties. When retrieving the Reliability property of an AnalogInput, it checks to see if an AnalogInput object exists in the virtual device. If it does, it returns the specific reliability value stored in the example database.

Similarly, when retrieving the SystemStatus of the main device or a virtual device, the callback first checks what deviceInstance is being requested before responding with the proper SystemStatus value from the example database.

```cpp
// BACnetVirtualDevicesServerExampleCPP.cpp lines 422-461
// Callback used by the BACnet Stack to get Enumerated property values from the
user
bool CallbackGetPropertyEnum(uint32_t deviceInstance, uint16_t objectType,
uint32_t objectInstance, uint32_t propertyIdentifier, uint32_t* value, bool
useArrayIndex, uint32_t propertyArrayIndex)
{
    // Example of Analog Inputs Reliability Property
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_RELIABILITY) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT) {
            if (g_database.analogInputs.count(deviceInstance) > 0 &&
g_database.analogInputs[deviceInstance].instance == objectInstance) {
                *value = g_database.analogInputs[deviceInstance].reliability;
                return true;
            }
            return false;
        }
    }

    // Example of System Status
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_SYSTEM_STATUS &&
        objectType == CASBACnetStackExampleConstants::OBJECT_TYPE_DEVICE)
    {
        if (objectInstance == g_database.mainDevice.instance) {
            *value = g_database.mainDevice.systemStatus;
            return true;
        }
        else {
            std::map<uint16_t, std::vector<ExampleDatabaseDevice> >::iterator it;
            for (it = g_database.virtualDevices.begin(); it !=
g_database.virtualDevices.end(); ++it) {
                std::vector<ExampleDatabaseDevice>::iterator devIt;
```

```
                for (devIt = it->second.begin(); devIt != it->second.end();
++devIt) {

                    if (objectType == devIt->instance) {
                        *value = devIt->systemStatus;
                        return true;
                    }
                }
            }
            return false;
        }
    }

    // We could not answer this request.
    return false;
}
```

CallbackGetPropertyOctetString

The CallbackGetPropertyOctetString handles getting any OctetString properties. In this example, the callback gets the OctetString properties related to the Ipv4 network port object of the main device (virtual router). Note that none of the network port objects representing virtual networks or virtual devices are checked as they do not have Ipv4 properties.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 464-505
// Callback used by the BACnet Stack to get OctetString property values from the
user
bool CallbackGetPropertyOctetString(const uint32_t deviceInstance, const uint16_t
objectType, const uint32_t objectInstance, const uint32_t propertyIdentifier,
uint8_t* value, uint32_t* valueElementCount, const uint32_t maxElementCount,
const bool useArrayIndex, const uint32_t propertyArrayIndex)
{
    // Example of Network Port Object IP Address property
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_IP_ADDRESS) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
```

```
            memcpy(value, g_database.networkPort.IPAddress,
g_database.networkPort.IPAddressLength);
            *valueElementCount = g_database.networkPort.IPAddressLength;
            return true;
        }
    }
    // Example of Network Port Object IP Default Gateway property
    else if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_IP_DEFAULT_GATEWAY) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
            memcpy(value, g_database.networkPort.IPDefaultGateway,
g_database.networkPort.IPDefaultGatewayLength);
            *valueElementCount = g_database.networkPort.IPDefaultGatewayLength;
            return true;
        }
    }
    // Example of Network Port Object IP Subnet Mask property
    else if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_IP_SUBNET_MASK) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
            memcpy(value, g_database.networkPort.IPSubnetMask,
g_database.networkPort.IPSubnetMaskLength);
            *valueElementCount = g_database.networkPort.IPSubnetMaskLength;
            return true;
        }
    }
    // Example of Network Port Object IP DNS Server property
    else if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_IP_DNS_SERVER) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
            // The IP DNS Server property is an array of DNS Server addresses
```

```
            if (useArrayIndex) {
                if (propertyArrayIndex != 0 && propertyArrayIndex <=
g_database.networkPort.IPDNSServers.size()) {
                    memcpy(value,
g_database.networkPort.IPDNSServers[propertyArrayIndex - 1],
g_database.networkPort.IPDNSServerLength);
                    *valueElementCount =
g_database.networkPort.IPDNSServerLength;
                    return true;
                }
            }
        }
    }
    return false;
}
```

CallbackGetPropertyReal

The CallbackGetPropertyReal handles getting any Real properties. In this example, the callback gets the PresentValue property of the AnalogInput objects in the virtual devices. Part of the checks in this callback are to verify if the requested AnalogInput object exists in the virtual device and if it does exist, the value is filled out.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 506-521
// Callback used by the BACnet Stack to get Real property values from the user
bool CallbackGetPropertyReal(uint32_t deviceInstance, uint16_t objectType,
uint32_t objectInstance, uint32_t propertyIdentifier, float* value, bool
useArrayIndex, uint32_t propertyArrayIndex)
{
    // Example of Analog Input / Value Object Present Value property
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_PRESENT_VALUE) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_ANALOG_INPUT) {
            if (g_database.analogInputs.count(deviceInstance) > 0 &&
g_database.analogInputs[deviceInstance].instance == objectInstance) {
                *value = g_database.analogInputs[deviceInstance].presentValue;
                return true;
```

```
            }
            return false;
        }
    }

    return false;
}
```

CallbackGetPropertyUInt

The CallbackGetPropertyUInt handles getting any UnsignedInt properties. In this example, the callback gets the UnsignedInt properties related to the Ipv4 network port object of the main device (virtual router). Note that none of the network port objects representing virtual networks or virtual devices are checked as they do not have Ipv4 properties.

```
// BACnetVirtualDevicesServerExampleCPP.cpp lines 523-546
// Callback used by the BACnet Stack to get Unsigned Integer property values from
the user
bool CallbackGetPropertyUInt(uint32_t deviceInstance, uint16_t objectType,
uint32_t objectInstance, uint32_t propertyIdentifier, uint32_t* value, bool
useArrayIndex, uint32_t propertyArrayIndex)
{
    // Example of Network Port Object BACnet IP UDP Port property
    if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_BACNET_IP_UDP_PORT) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
            *value = g_database.networkPort.BACnetIPUDPPort;
            return true;
        }
    }
    // Example of Network Port Object IP DNS Server Array Size property
    // Any properties that are an array must have an entry here for the array
size.
    // The array size is provided only if the useArrayIndex parameter is set to
true and the propertyArrayIndex is zero.
```

```
    else if (propertyIdentifier ==
CASBACnetStackExampleConstants::PROPERTY_IDENTIFIER_IP_DNS_SERVER) {
        if (objectType ==
CASBACnetStackExampleConstants::OBJECT_TYPE_NETWORK_PORT && objectInstance ==
g_database.networkPort.instance) {
            if (useArrayIndex && propertyArrayIndex == 0) {
                *value = (uint32_t)g_database.networkPort.IPDNSServers.size();
                return true;
            }
        }
    }

    return false;
}
```

Version Table

Doc v.	Date	Author	Stack/Example V.	Notes
1.00	25/06/2020	ACF	3.17.0 / 0.0.2	Created document