SIMPLified 2 BACnet Transformer Worked Example

BACNET TO SIMPL DRIVER MIDDLEWARE

This example illustrates the use of the program generation features of SIMPLified 2, using one of the included transformer plugins – the BACnet Transformer. A collection of sample files are included with this worked example to aid in the generation of specific scenarios.

Use Case

An existing system is being upgraded to Crestron Home. The client already has a legacy, wired, Heatmiser heating control system and wishes this to be included in the final Crestron Home solution. Although we do have a solution for the newer Heatmiser Neo system, the legacy RS485 system is not supported and it would not be cost effective to have this added to the existing middleware product. An alternative solution is to use a standard Crestron processor, running a SIMPL Windows program which presents BACnet objects to the Crestron Home system, and connects, via SIMPL logic, to the existing Heatmiser RS485 drivers.

Each zone requires 4 BACNet objects (Set Point, Current Temp, Operating Mode and Heat Demand State) and there are 12 zones. This is a total of 12 sets of identical zone logic and 48 configured BACnet objects in the SIMPL Windows program. Not an insurmountable task by any means, but tedious, error prone work, and any updates could require repeat work. Not much fun.

The BACnet Transformer plugin and program generation of SIMPLified can make this task significantly easier and provide a foundational program capable of handling the task or provide a base to build upon.

Note

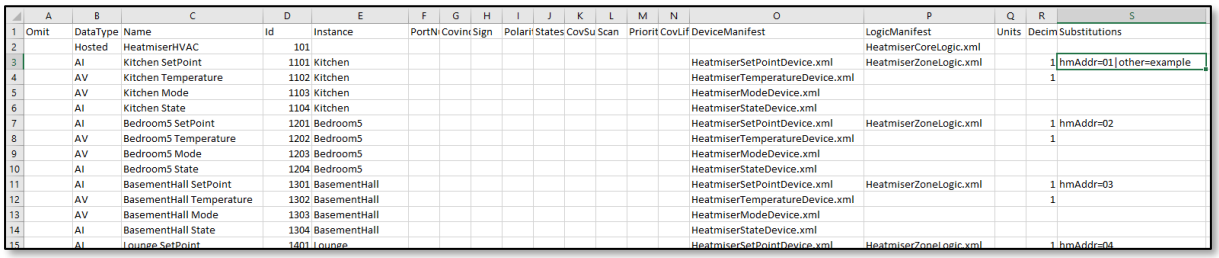
It is important to appreciate that this is an *example* use case, and the fact that – in this scenario – we're building a program to support Heatmiser is unrelated to the steps outlined in the subsequent pages. This technique could apply just as easily to any other situation that would benefit from a rapid, reliably method to of creating large quantities of BACnet objects within a SIMPL program. Connecting the associated logic is a further efficiency gain and could apply to any use case where BACnet integration with some other device logic is desired.

Step 1 – Define the BACnet objects in a CSV file

The BACnet transformer takes a CSV file as input and uses the information to populate a SIMPL windows hardware definition with the appropriate BACnet objects, connect up signals and optionally create associated logic.

We will need 4 BACnet objects per zone and a block of logic to connect the BACnet input and output values to each Heatmiser Thermostat.

The BACnet transformer uses a specific template for the CSV file and this is fully documented in the program generator user guide. An extract of the example file is shown below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Omit	DataType	Name	Id	Instance	PortN	Covin	Sign	Polari	States	CovSu	Scan	Priorit	CovLif	DeviceManifest	LogicManifest	Units	Decim	Substitutions
2		Hosted	HeatmiserHVAC	101												HeatmiserCoreLogic.xml			
3		AI	Kitchen SetPoint	1101	Kitchen										HeatmiserSetPointDevice.xml	HeatmiserZoneLogic.xml		1	hmAddr=01|other=example
4		AV	Kitchen Temperature	1102	Kitchen										HeatmiserTemperatureDevice.xml			1	
5		AV	Kitchen Mode	1103	Kitchen										HeatmiserModeDevice.xml				
6		AI	Kitchen State	1104	Kitchen										HeatmiserStateDevice.xml				
7		AI	Bedroom5 SetPoint	1201	Bedroom5										HeatmiserSetPointDevice.xml	HeatmiserZoneLogic.xml		1	hmAddr=02
8		AV	Bedroom5 Temperature	1202	Bedroom5										HeatmiserTemperatureDevice.xml			1	
9		AV	Bedroom5 Mode	1203	Bedroom5										HeatmiserModeDevice.xml				
10		AI	Bedroom5 State	1204	Bedroom5										HeatmiserStateDevice.xml				
11		AI	BasementHall SetPoint	1301	BasementHall										HeatmiserSetPointDevice.xml	HeatmiserZoneLogic.xml		1	hmAddr=03
12		AV	BasementHall Temperature	1302	BasementHall										HeatmiserTemperatureDevice.xml			1	
13		AV	BasementHall Mode	1303	BasementHall										HeatmiserModeDevice.xml				
14		AI	BasementHall State	1304	BasementHall										HeatmiserStateDevice.xml				
15		AI	Lounge SetPoint	1401	Lounge										HeatmiserSetPointDevice.xml	HeatmiserZoneLogic.xml		1	hmAddr=04

Many of the fields in the CSV are optional, and the BACnet transformer will read the first line of the CSV file to establish which column contains which field – the name of each column heading is therefore critically important.

In this example, we define the parameters for the BACnet host, and then a series of BACnet objects. These are Analogue Inputs (AI) and Analogue Values (AV) and specified under the "DataType" column.

The "Instance" column defines the value that will be substituted for "bacnetDevice" in the signal and logic definitions.

The "DeviceManifest" column specifies that an external Manifest file is to be loaded and used to define the signal connections for the hardware object and any default parameter values. These values can then be overridden from the CSV file if they are defined (e.g. "Decimals"in this example.)

The "LogicManifest" column specifies a single logic folder for each thermostat. This logic folder actually covers the logic for all BACnet objects of the thermostat and signal connections are resolved once the program is generated.

The "Substitutions" column provides a way for the CSV file to override signal, comment and parameter values with row specific values. In this example, we are replacing the value of "hmAddr" with the two-character hexadecimal address value for the Heatmiser module.

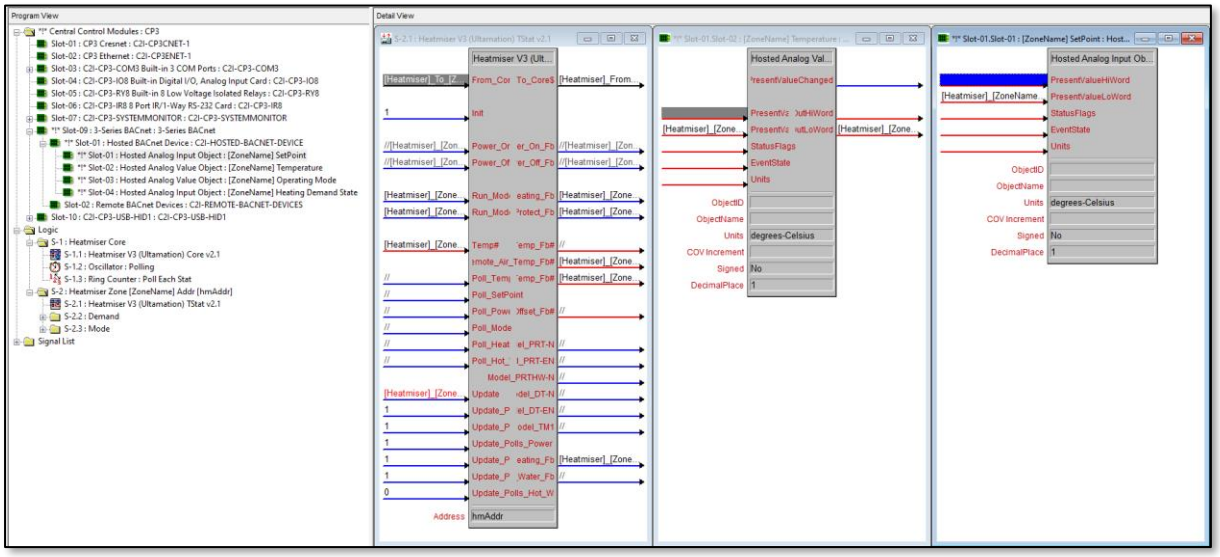
Name value substitution pairs are defined by NAME=VALUE and multiple values are separated using the pipe 'I'symbol. So multiple substitutions can be specified such as:

```
hmAddr=01|other=example
```

Step 2 – Create the template program for the zones and central logic in SIMPL

Now that we have the base data defined for the BACnet generator, we must provide Manifest files (essentially, SIMPL programming logic "stubs") as defined in the CSV so that the program generator can build the logic and tie it to the BACnet devices.

To do this, we create a "template" SIMPL program. This program has a definition for one thermostat (hardware and logic) and any supplementary scaffolding that is required for the final program. This gives us a convenient place to make edits to maintain the program.



The program doesn't need to compile – and, in fact, will not since we have incomplete symbols in the hardware definition. This is fine as the missing parameters will be populated by the BACnet Transformer plugin during program build.

The Heatmiser zone logic includes some simple interfacing logic between the BACnet and Heatmiser world, but this could be as sophisticated as you wish.

The program does include substitution parameters; shown here are "hmAddr" and "ZoneName" which, as will be seen in the next section, are marked for substitution in the manifest export.

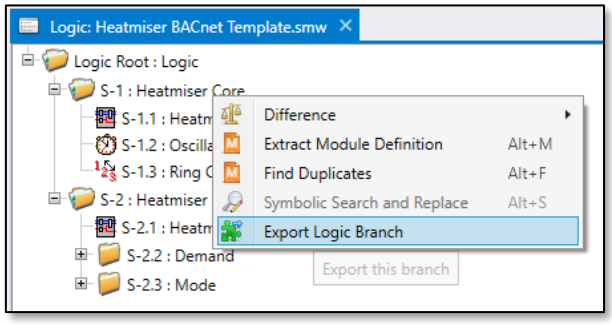
The final part of the program is the central logic that connects all of the zones together on the Heatmiser side – this is simply the standard programming logic and is included here so that the program generator can include this "one off" logic in the final build.

Step 3 – Use SIMPLified to export the BACnet object and SIMPL logic templates

Now that we have a template program for a single heating zone and the central logic, we use SIMPLified to export the logic and device definitions to XML Manifests. The XML Manifest is the standard form of serialisation for the SIMPLified program generator and we will create a file for each of the key elements:

- The core Heatmiser logic
- The Heatmiser thermostat logic for a single zone
- The BACnet objects (4 No.)

Open the template program in SIMPLified and select each of the elements in the program tree as shown below:



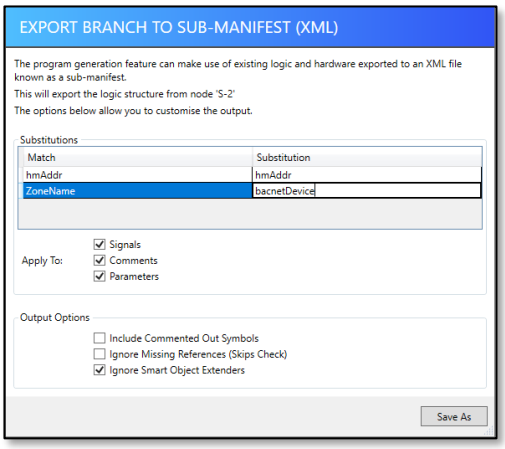
You need to tell SIMPLified what substitution markers to look for in the program template and which substitution keys to replace them with. During program generation, these keys will be replaced with literal values which are defined in the CSV (as described in step 1) or the RESERVED key "bacnetDevice" which is replaced with a value that is either auto-generated by the BACnet transformer, or taken from the "Instance" field of the CSV file.

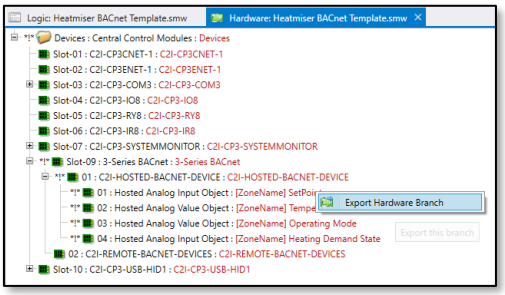
In our example, we write the Manifest export files to the same directory as the CSV file and the SIMPL template program. In addition to creating the Manifest XML files, SIMPLified will attempt to create a catalogue of dependent modules in a subdirectory so that the program generation can also pull in these files.

You can define your own structure for this, but having a single "Config" folder keeps everything together.

Repeat the process for each of the BACnet objects required of a single zone. This is accomplished by opening the Device Tree view for the current program within SIMPLified 2.

The exported manifests are XML documents.
e.g.



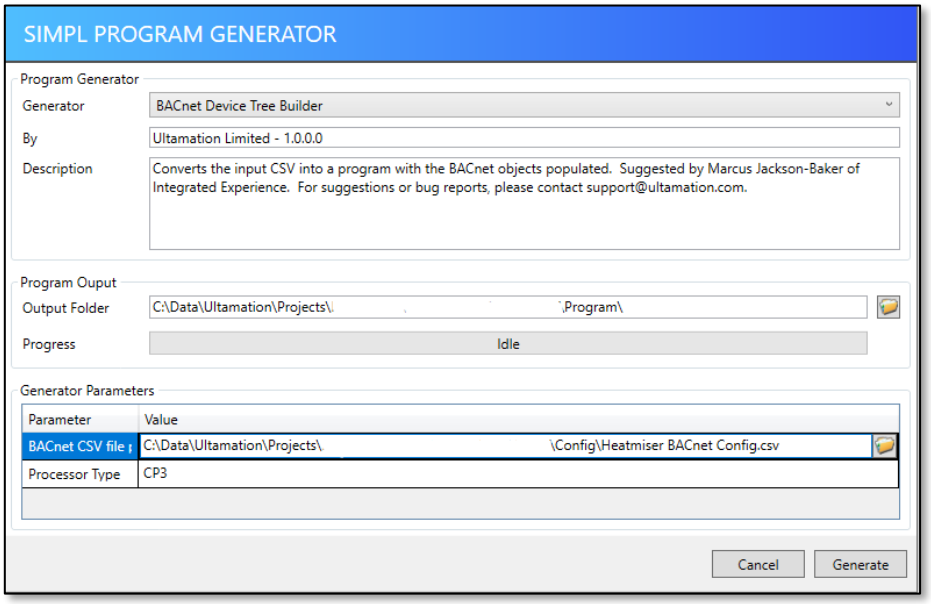



```
<?xml version="1.0" encoding="utf-8"?>
<SimplManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.ultamation.com">
  <DeviceTree>
    <BACnet>
      <Host name="Placeholder">
        <DeviceCode>0</DeviceCode>
        <Devices>
          <HostedDevice type="Hosted Analog Input Object" name="[%bacnetDevice%] SetPoint">
            <DeviceCode>3051</DeviceCode>
            <Parameters>
              <Parameter name="Units">62</Parameter>
              <Parameter name="Signed">0d</Parameter>
              <Parameter name="DecimalPlace">1d</Parameter>
            </Parameters>
            <Connections>
              <SignalIn cue="PresentValueLoWord">[Heatmiser]_[%bacnetDevice%]_Temp_x10_Fb#</SignalIn>
            </Connections>
          </HostedDevice>
        </Devices>
      </Host>
    </BACnet>
  </DeviceTree>
</SimplManifest>
```

Step 4 – Run the program generator with the BACnet Transformer

We now have all of the elements required to build a program automatically.

Within SIMPLified 2, select the Program Generation tool. This will offer you a selection of Generators (at the time of writing, we ship two free transformer plugins with SIMPLified). Select the BACnet transformer.



Choose the output folder for your auto-generated program. We suggest you keep this SEPARATE from you config folder.

Select the BACnet CSV file location, and enter the type of processor you will be building for. The processor type must match the name of the processor model EXACTLY as it appears in SIMPL Windows (i.e. Note that some processor models include a period '.' at the end of the model name).

Click Generate and, if all goes well, SIMPLified will build a program for you. You will then be asked if you wish to compile the resulting program – answer NO.

Step 5 – Final edits

Although the program will be largely complete, there are a few modifications we need to make before it can be compiled and uploaded successfully.

At present, the program generator does not support connection of the COM port, so we need to manually configure the COM port (RS485, 4800 8-N-1) and connect it to the Heatmiser core module.

Add then…

Step 6 – Compile and upload

Compile the SIMPL Windows program (under normal conditions, you should get a clean compile, but see below for the worked example) and upload to the Creston processor as normal.

NOTE: In the files provided with this example, we have omitted the Heatmiser Marshaller which is a central part of the Heatmiser RS485 module suite. This is included in the full Heatmiser RS485 integration module that is available for purchase at the Ultamation online shop.