NATIVE SERIES™

MATRIXBBC
PROGRAMMERS GUIDE

AMERICAN AUTO-MATRIX®
SMART BUILDING SOLUTIONS®

MatrixBBC Programmers Guide

Part # - 1E-04-00-0147

WORLD HEADQUARTERS

American Auto-Matrix
One Technology Lane
Export, Pennsylvania 15632-8903 USA
Tel (1) 724-733-2000
Fax (1) 724-327-6124
Email aam@aamatrix.com
www.aamatrix.com

Updated 10/5/2012

. Corresponds with product launch (firmware revision v1.02)

This manual describes the operation and configuration of MatrixBBC.

This document is divided into the following sections:
. One: *Overview*, describing the product platform.
. Two: *Device Setup*, describing the setup and configuration of device and communications.
. Three: *Programs and Files*, describing SPL programming.
. Four: *Scheduling*, describing the setup and configuration of schedules and calendars.
. Five: *Alarm Routing*, describing how to route input and output alarms to workstations and other devices.
. Six: *Data Storage*, describing the setup and configuration of Analog and Binary Values objects.
. Seven: *Data Manipulation*, describing the setup and configuration of data manipulation objects.
. Eight: *Data Movement*, describing the setup and configuration of data movement objects.
. Nine: *Expansion I/O*, describing the setup and configuration of expandable STATbus IOX modules.
. Ten: *Inputs Setup*, describing the setup and configuration of inputs.
. Eleven: *Outputs Setup*, describing the setup and configuration of outputs.
. Twelve: *Control Loop*, describing how to use and implement control loops.
. Thirteen: *Miscellaneous*, described the use of objects found under the Miscellaneous category.

This document contains certain style and formatting conventions for conveying information in a clear and concise manner:
. Menu commands appear with a “>” symbol between levels. For example: *File>Open*.
. *Italics* indicate options within software.

SECTION 1: OVERVIEW

This section provides general information regarding the MatrixBBC as well as a short review of the fundamental concepts of the BACnet protocol.

IN THIS SECTION

1.1 WHAT IS THE MATRIXBBC?

The MatrixBBC is a BACnet Building Controller product platform designed to provide global area control capabilities. As a BACnet-compliant building controller, the product provides the ability to store and execute logic, perform internal and external scheduling, trend collection, as well as other features described throughout this document. The MatrixBBC is designed to provide hardware and software flexibility using on-board I/O and STATbus - AAM's innovative sensor networking technology.

The MatrixBBC can be used in a wide variety of applications that require either area control capabilities, stand-alone Ethernet-enabled I/O control capabilities, sa well as full peer-to-peer network capabilities with other BACnet devices.

Flexible by software design, the MatrixBBC supports dynamic creation of objects. The following table below provides a list of the maximum amount of objects that the device will support.

Table 1-1 Maximum Object Count

Object Type	Limit
Universal Inputs	144
Analog Outputs	72
Binary Outputs	72
Schedules	32
Calendars	32
Trend Logs	256
Analog Values	1000
Binary Values	1000
Programs	64
Notification Class	10
Season	1
Input Select	64
Remap	64
Netmap	64
Logic	64
Math	64
Motor Control	64
Scale	64

Table 1-1 Maximum Object Count

Object Type	Limit
Piecewise Curve	8
Enthalpy	64
Thermostatic Control Loop	64
PID Control Loop	64
Staging	16
Comm Status	1
Manufacturing	1
Broadcast	8

1.2 FUNDAMENTAL CONCEPTS OVERVIEW

This section of the user manual reviews standard fundamental concepts and provides an explanation of the prerequisite information necessary to know prior to installing this product.

1.2.1 BACNET MS/TP OVERVIEW

BACnet MS/TP (**M**aster **S**lave **T**oken **P**assing) is an EIA-485 network layer intended for use with lower-level devices such as Unitary Controllers. In comparison to BACnet/IP and BACnet/Ethernet, MS/TP is more cost-effective to implement due to lower cost of wiring. Given the MS/TP network is a serial-based network, devices may be configured to communicate at different baud rates specified by BACnet. Therefore it is essential to know information regarding the BACnet network you are connecting to prior to installing and implementing the MatrixBBC.

1.2.2 MS/TP NETWORK TOKEN PASSING

BACnet MS/TP uses token passing to allow devices to communicate on the network. Token passing is controlled by each device, which contains an internal memory list of other MS/TP peers connected to the network. The token is passed in order of the MAC Address (Unit ID) from lowest to highest. In most MS/TP networks, each device is configured to be a master. Given all devices may be a master, MS/TP may appear and react slower than traditional building automation protocols. However, configuring your network for faster baud rates will help provide better bandwidth and transport speed of network messaging.

Token passing is a communications scheme that allows connected devices to inter-communicate with one another. A network “token” is passed from unit to unit on the network in a round-robin fashion by order of the MAC Address (lowest to highest) to provide a transport to access the network. When a unit possesses the token, it may perform any network activity for which it is responsible. When finished, the token is then passed onto the next device. At any time, the unit that possesses the token is the only device permitted to initiate communications with another device on the network or to request information from it. A device that receives the token may or may not need to perform network functions (e.g. read values from a remote device, broadcast information, etc.). If not, it will simply pass the token along the network.

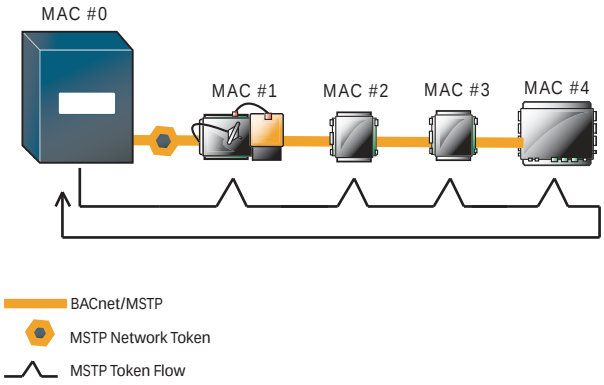

Figure 1-1 MS/TP Token Passing Example

Because each device can be an MS/TP master, it is important to realize that each MS/TP network should be optimized. Later sub-sections of this manual explain this process.

1.2.3 BACNET MS/TP LAN WIRING

Similar to EIA-485 standards, BACnet MS/TP networks support a maximum network distance of 4000 feet maximum with 18-AWG, 2-wire, shielded-twisted-pair cabling. This device is designed with half-watt serial drivers, allowing up to a maximum of 64 devices to be connected to a single MS/TP network bus.
If you are connecting the MatrixBBC to an existing MS/TP network consisting of third-party devices, consult third-party vendor documentation regarding MS/TP network considerations.

1.2.4 MS/TP DEVICE ADDRESSING

BACnet MS/TP devices contain two unique addresses. One device address is known as a Device Instance, and the other is a MAC Address.

The Device Instance is an address assignment that is used to identify the BACnet device on a global BACnet network. When a device is connected to a global BACnet network consisting of multiple data layers joined together using routers, the Device Instance is used to uniquely identify the device on a global basis. The valid range for the device instance in a BACnet device is 0 to 4,194,302. The MatrixBBC controller must be configured for a unique, non-conflicting Device Instance. In the event that multiple devices are assigned the same Device Instance, both devices will simply not communicate on the BACnet network, or could be subject to mis-directed messaging (a message intended for Device-A may be routed to Device-B)

The MAC Address is an address assignment used within the BACnet MS/TP segment to permit a device to actively communicate on the BACnet MS/TP network. Valid MAC Address assignments range from 0 to 127 and are typically assigned in a logical and incremental order to permit faster token passing between devices. The MAC Address of a BACnet MS/TP device must be a unique, non-conflicting value that exists on the local MS/TP network. In the event that multiple devices are assigned with the same MAC Address, the effects can be far more detrimental than that of a conflicting Device Instance; potentially resulting in a failure of the entire local MS/TP network. In the event that the MatrixBBC determining its MAC Address may be a duplicate, the MatrixBBC will inform the user that a duplicate MAC Address has been detected and will not perform client communications until resolved.

In most cases, the MatrixBBC should have it's MAC Address configured for a value of zero (0). Given it's capabilities as a router and a device that will likely perform the most network activity, this will allow it to re-generate the network token should a drop occur.

1.2.5 COMMUNICATION BAUD RATES

As a serial based protocol, BACnet MS/TP supports the following four baud rates: 9.6kbps, 19.2kbps, 38.4kbps, and 76.8kbps.

Each device communicating on an MS/TP network must be configured for the same baud rate at all times. In the event that the MatrixBBC's communication baud rate is incorrect for the network it is connected to, the MatrixBBC will inform the user that a different baud rate has been detected and will not perform client communications until resolved.

1.2.6 NETWORK OPTIMIZATION

In BACnet MS/TP devices, specific device properties are available to permit optimization of network communications. By adjusting the Device properties max-master and max-info-frames, users can adjust the token passing abilities of devices. The functionality of these two properties is described as follows:

. ***Max-Master*** - defines the highest unit ID of a MSTP master that is connected to the network. This value specifies to what maximum address a token may pass. For example if you have 64 devices addressed in logical order, this value would be assigned to 64. This value should be set to the same value across all devices connected to an MSTP network.

. ***Max-Info-Frames*** - defines the amount of data frames that a MSTP master can use the token before passing onto the next device. This value is typically set by the factory, but can be modified if necessary. In the event a device does not need to keep the token for the amount of frames specified, AAM devices will automatically pass the token onto the next device.

SECTION 2: DEVICE SETUP

This section describes software configuration of the device itself, including communications and setup of alarm and event information to a front-end, building controller, or operator workstation. In order to setup the MatrixBBC, a web-browser must be used to access general device setup. To configure items such as BACnet Time Synchronization, Daylight Savings, and Device Address Binding, NB-Pro must be used.

IN THIS SECTION

2.1 LOGGING INTO THE MATRIXBBC

Within the MatrixBBC, the general setup and configuration for all BACnet communications is achieved through an embedded web-server built into the product. The web-server contains pages that allow technicians to configure items such as MS/TP network configuration, BACnet routing information, as well as other items.

To access the web-server of the MatrixBBC, perform the following steps:

1. Using a standard web-browser, enter the default IP address of your MatrixBBC. The default IP address is 192.168.1.250. Navigate to the device.

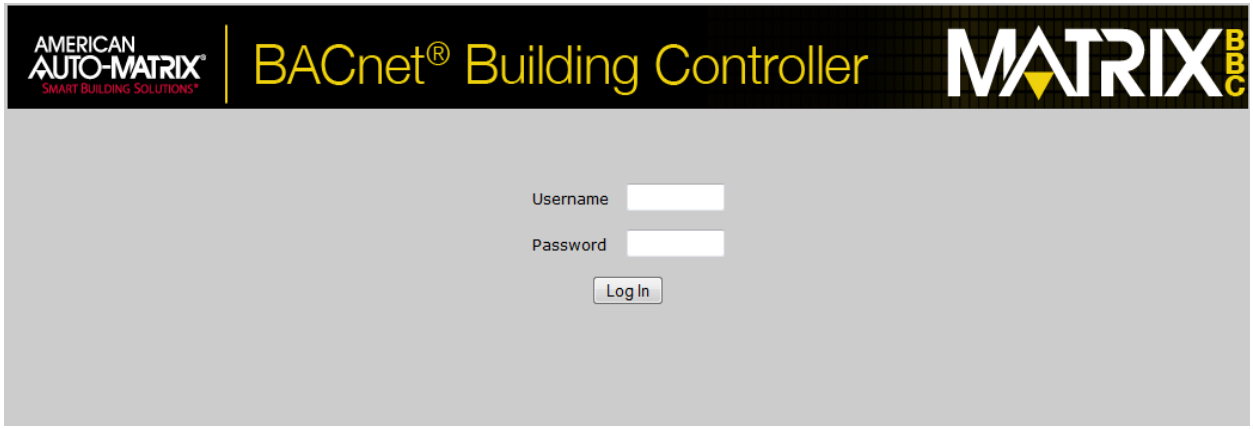

Figure 2-1 - MatrixBBC Login Page

2. When navigated successfully, you will be greeted with the login page. Enter your username and password and click the *Log In* button. The default, case-sensitive credentials are as followed:

. Username - aamuser
. Password - default

2.1.1 ABOUT THE MATRIXBBC CONTROL PANEL

The MatrixBBC Control Panel contains several web-pages that permit technicians to manage and setup the core hardware platform of the controller. Common items configured and administered through this embedded web-site include the following:

. Licensing - used to manage your MatrixBBC's licensed features.
. Communication Setup - used to configure BACnet network communications for the MatrixBBC.
. System Administration - used to administer the hardware, perform diagnostic troubleshooting, and perform features such as backing up and restoring the programming of the MatrixBBC.
. System Logs - used to retract diagnostic logs for advanced troubleshooting with AAM Technical Services.

Each portion of the Control Panel is discussed in further details within this section.

2.2 LICENSING

The Licensing folder contains a page to view and manage your current license.

2.2.1 LICENSE

The License area provided the ability to view the current license of your MatrixBBC, as well as provide the means to upload/download license files. By default, the MatrixBBC is licensed by AAM when the product is shipped. As new features and capabilities are made available, they may potentially be licensed as optional features. This area provide the means to manage your license.

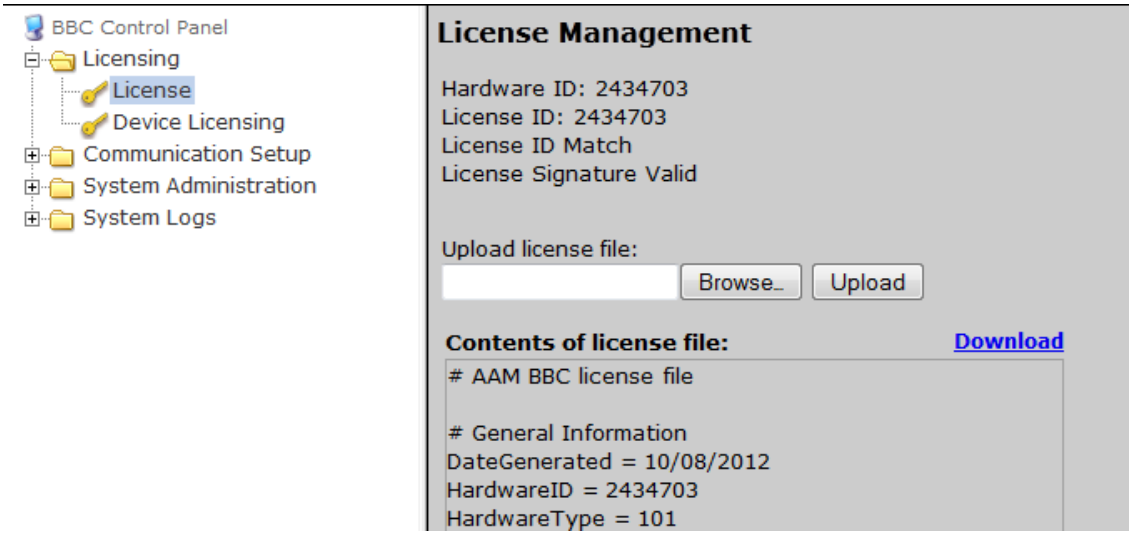

Figure 2-2 - MatrixBBC License Page

You may download a copy of your license by clicking the blue Download link shown above. Should your MatrixBBC be upgraded by license, you may use the Browse and Upload buttons to upload a new license to the target. In order to make license changes effective, it is recommended that you restart the MatrixBBC through the System Services page found under System Administration.

2.3 COMMUNICATION SETUP

The Communication Setup page is used to setup all BACnet network communication for the MatrixBBC. This area contains a page for EIA-485 Port Configuration, as well as a sub-folder for BACnet details.

2.3.1 PORT CONFIGURATION

The Port Configuration page is used to enable and disable MS/TP network communications for each RS-485 port. To enable BACnet MS/TP network communications, simply set the Protocol for each port to BACnet MSTP. To disable a port, set the selector to Not Assigned. Click Submit when finished and follow any additional steps required by the embedded web-site.

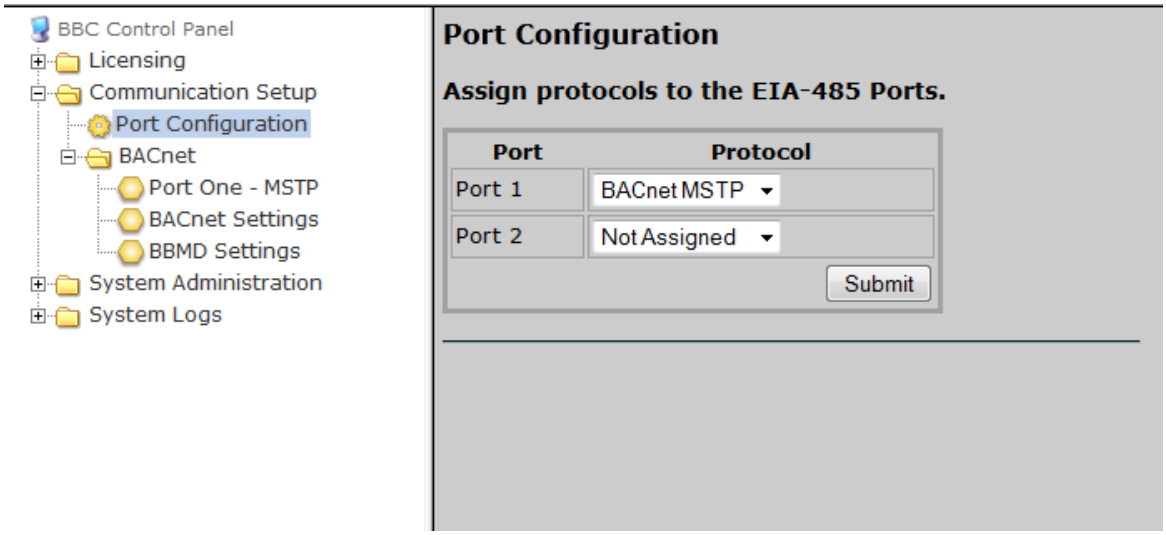

Figure 2-3 - MatrixBBC Port Configuration

2.3.2 PORT ONE/TWO - MSTP

The Port One/Two - MSTP page is used to configure BACnet MS/TP communication settings for the specific port - including MAC Address, Baud Rate, and token control properties.

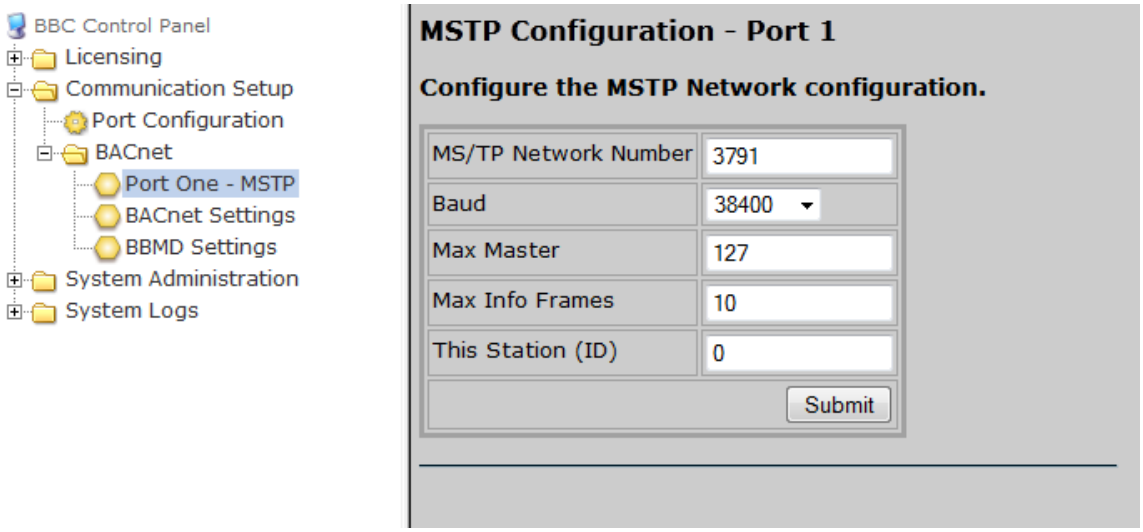

Figure 2-4 - MS/TP Port Configuration Properties

The following table below provides configuration notes for each parameter.

Table 2-1 BACnet MS/TP Port Configuration Notes

Property	Notes
MS/TP Network Number	Defines the network number for this MS/TP network bus. Valid ranges are 1-65534, and must be unique for each MS/TP network system-wide.
Baud Rate	Defines the network speed of the MS/TP network.
Max Master	Defines the highest maximum MS/TP master node addressed on the network.
Max Info Frames	Defines the highest number of frames permitted to be used by the MatrixBBC before the network token is passed.
This Station (ID)	Defines the MAC Address/Unit ID for the port. By default, this is set for zero (0) and should remain at this assignment.

2.3.3 BACNET SETTINGS

The BACnet Settings page is used to configure all aspects of data routing, read/write retires, and another parameters within the MatrixBBC. The following tables below provide details on configuration for each area and it's properties.

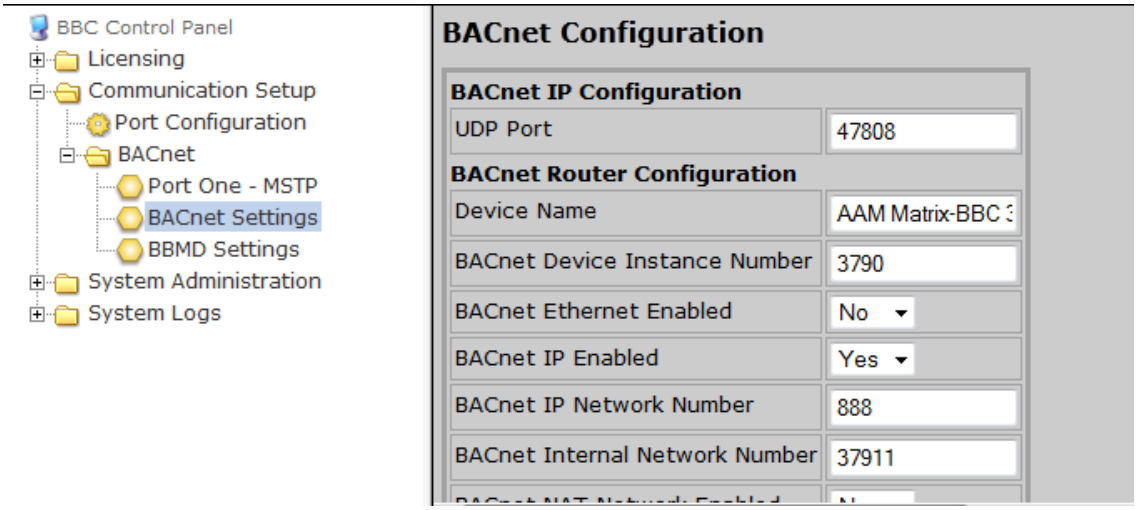

Figure 2-5 - MatrixBBC BACnet Configuration Settings

Table 2-2 BACnet/IP Configuration Notes

Property	Notes
UDP Port	Defines the UDP port that general BACnet/IP communications will occur on. By default, most BACnet/IP devices uses Port 47808.

Table 2-3 BACnet Router Configuration Notes

Property	Notes
Device Name	Defines the network identifier name for this target as a BACnet device.
BACnet Device Instance Number	Defines the Device ID for this target as a BACnet device. Valid ranges are from 0 - 4194302.
BACnet Ethernet Enabled	Defines if BACnet/Ethernet routing is enabled. In order for AspectFT to display and interact with BACnet/Ethernet devices, this must be set to Yes. NOTE - On a local network, only one device should be configured to have BACnet/Ethernet enabled. Having more than one device enabled for BACnet/Ethernet can result in cyclic routing of packets.
BACnet IP Enabled	Defines if BACnet/IP routing is enabled.
BACnet IP Network Number	Defines the BACnet/IP network number this device either hosts or is joined to. Valid ranges are 1-65534.
BACnet Internal Network Number	Defines the internal network number to allow AspectFT to interact with objects local to itself. This network number must be unique across all devices and cannot conflict with any established network numbers for IP, Ethernet, or MS/TP. Valid ranges are 1-65534
BACnet NAT Network Enabled	Defines if BACnet BBMD NAT functionality and accessibility is enabled. When enabled, this feature permits NB-Pro to access a site remotely in a more efficient manner when NAT routers are involved. If enabled, the following properties must be configured to ensure proper operation: . BACnet NAT Network Number - Defines a unique network number for NAT access. This network number must be unique from other networks within your BACnet inter-network. Valid ranges are 1-65534. . BACnet NAT External IP Address - Defines the external IP address for the NAT router that the device will forward packets through. . BACnet NAT External Port - Defines the UDP Port assignment that BACnet packets will be forwarded through. This cannot conflict with the standard BACnet IP UDP Port Setting. If you intend to connect to the device remotely using NB-Pro, this port would be used in your NB-Pro Connection Settings.
BACnet APDU Timeout	Defines the APDU time out for BACnet communications. This value defaults to 3000ms and is considered optimal.
Router Debug Level	Defines level of debugging for the device. This should be set to zero (0).
BACnet Debug Level	Defines level of debugging for general BACnet communications. This should be set to zero (0).

2.3.4 BBMD SETTINGS

The BBMD Settings area is used configure BBMD table setup for BACnet networks. Within this section, two BBMD tables are available - one for standard BBMD configuration, and another for BBMD over NAT. Either or must be enabled in order to add entries into the corresponding tables.

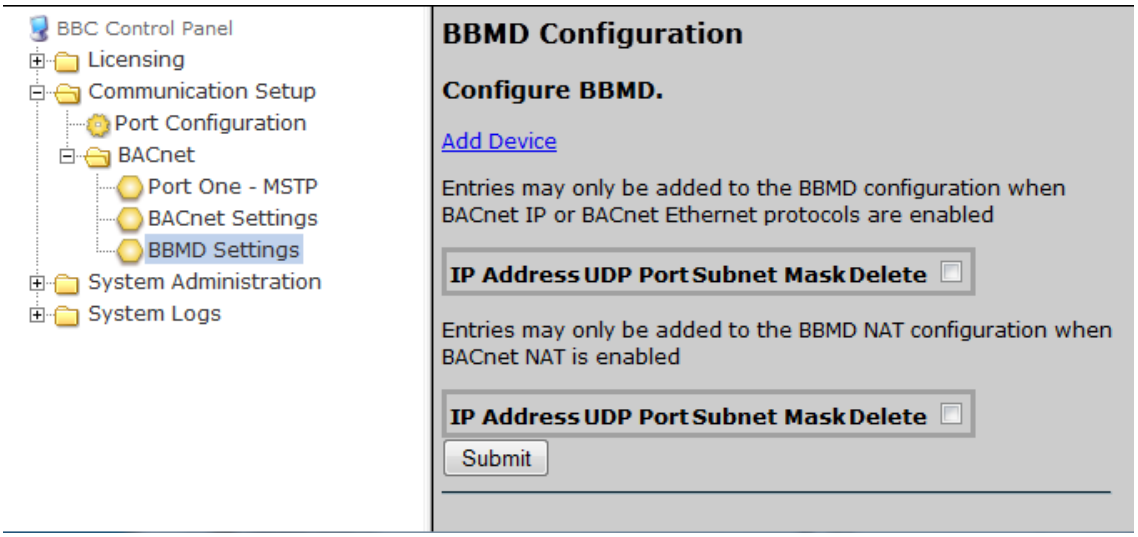

Figure 2-6 - BBMD Configuration

When entering addresses into the table, it is recommended that two-hop distribution methods be used, where an entry should contain a mask of 255.255.255.255 (indicating two-hop, global). This reduces the potential for standard IP routing issues that could occur on an IT infrastructure.

2.4 SYSTEM ADMINISTRATION

The System Administration area is used to administer the hardware elements of the MatrixBBC, including Ethernet address setup, and other variables.

2.4.1 SYSTEM SERVICES

The System Services area is used to perform a reboot of the controller itself, as well as restart Ethernet and web services. To restart a specific item, select the item from the drop-down and click *Submit*.

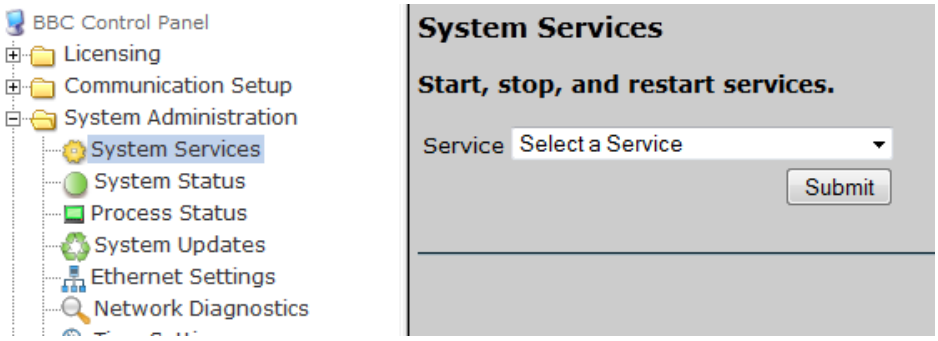

Figure 2-7 - System Services

2.4.2 SYSTEM STATUS

The System Status page provides an overview of all information contained within the system, provided revision information, as well as memory and load status.

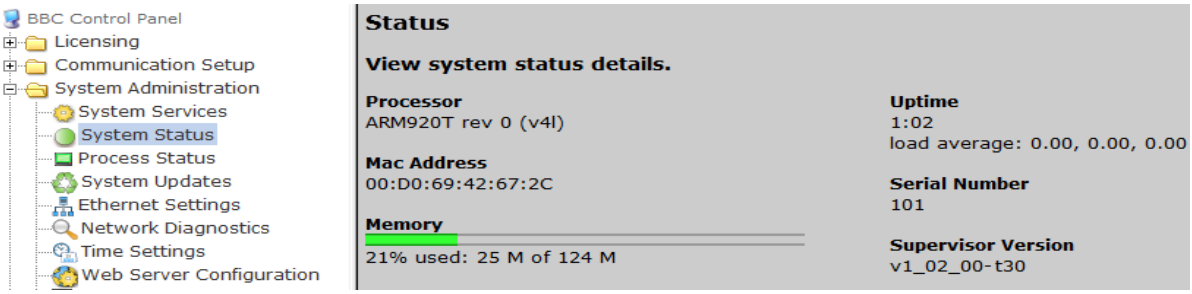

Figure 2-8 - System Status

2.4.3 PROCESS STATUS

The Process Status page provides a view of all system services that are running. This page is helpful for troubleshooting issues related to CPU usage and memory allocation.

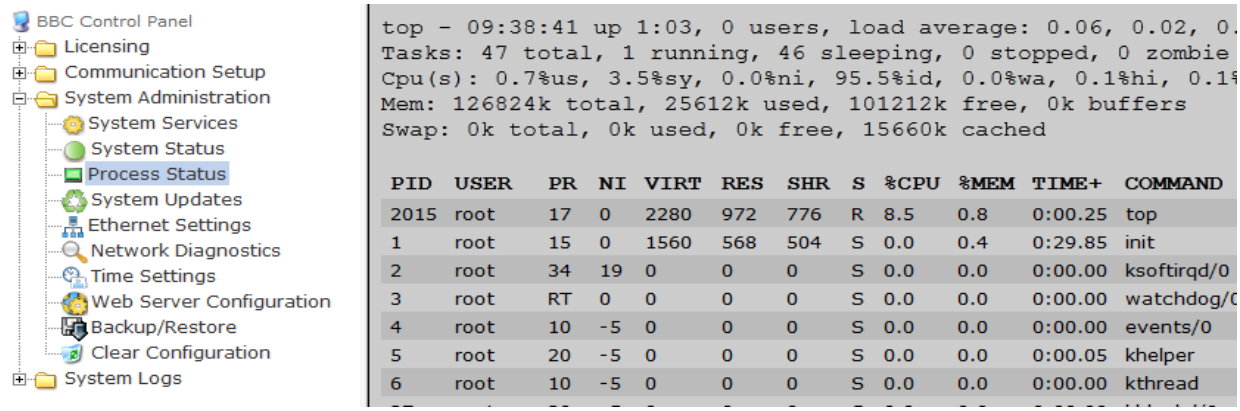

Figure 2-9 - Process Status

2.4.4 SYSTEM UPDATES

The System Updates page is used to apply firmware and maintenance updates to the MatrixBBC when needed. This page provides a simple wizard for applying system updates.

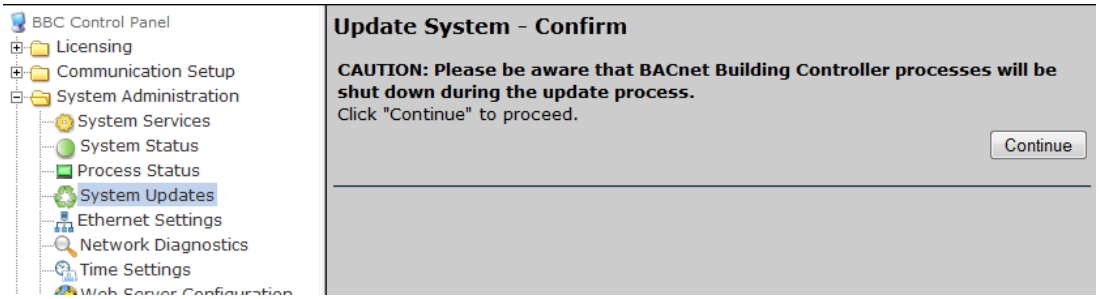

Figure 2-10 - System Updates

2.4.5 ETHERNET SETTINGS

The Ethernet Settings page is used to assign a static IP address to the hardware, and can also be used to configure dynamic addressing provided that an address reservation has been applied to a DHCP server.

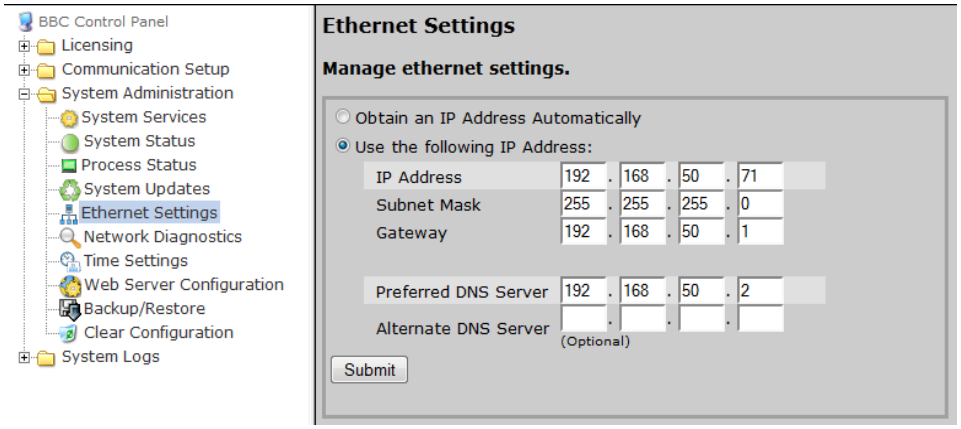

Figure 2-11 - Ethernet Settings

2.4.6 NETWORK DIAGNOSTICS

The Network Diagnostics page provides technicians with the ability to perform basic, common network diagnostic tools from the BBC. Utilities are available to perform the following functions:

. Ping
. Traceroute
. DNS Test
. MySQL Test

2.4.6.1 PING

Ping provides the ability to perform a PING test against a specific address or resolvable name. Ping tests are commonly used to verify connectivity to a particular device or server. To perform Ping test, simply enter the address or resolvable name into the field provided and click Start Ping Test. A result window will be displayed for the test indicating the sequence number, the time to live, and the response time.

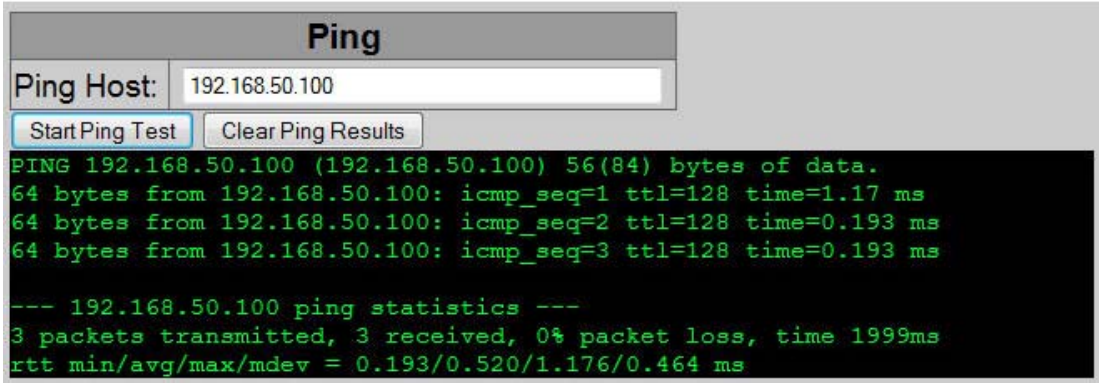

Figure 2-12 - Ping Test

2.4.6.2 TRACEROUTE

Traceroute provides information for displaying a route (path) and measuring transit delays of packets across the network. To perform a traceroute test, simply enter the address or resolvable name into the field provided and click Start Traceroute. A result window will be displayed for the test indicating each hop and route.

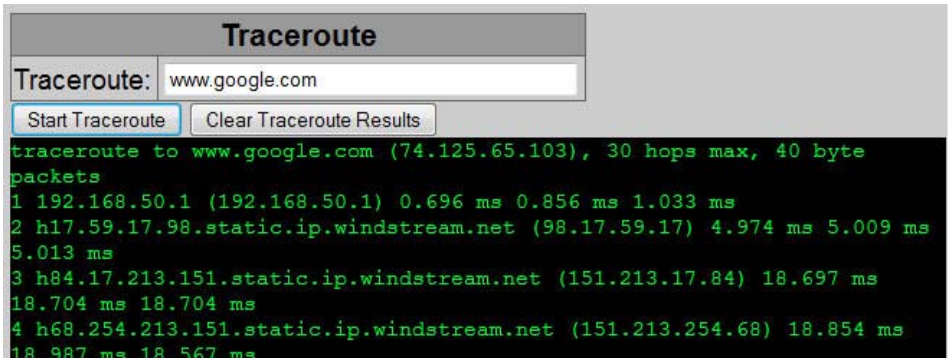

Figure 2-13 - Traceroute Test

2.4.6.3 DNS TEST

DNS Test can be used to verify name resolution of an assigned resolvable name. Simply enter a resolvable name into the provided field and click the Start DNS Test button.

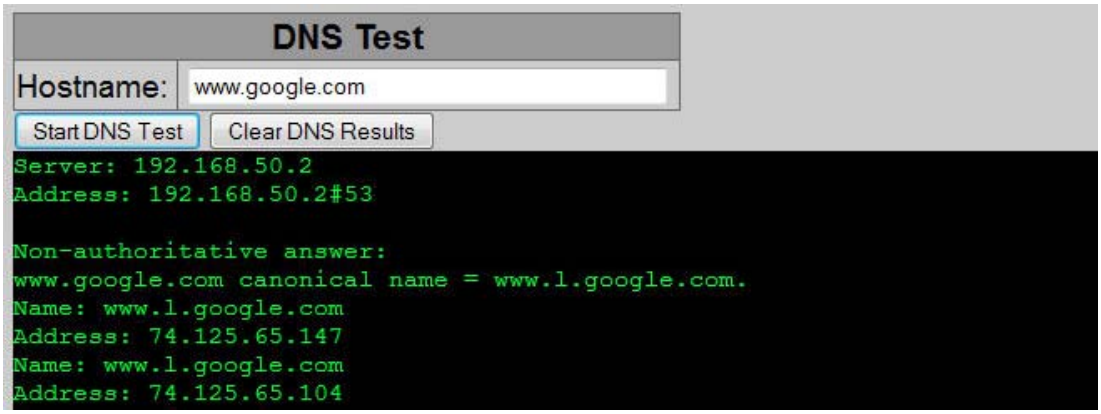

Figure 2-14 - DNS Test

2.4.6.4 MYSQL

MySQL Test can be used to verify access to a MySQL Database either locally on an AspectFT server-based target, or a remote MySQL database. Enter a host name (IP address or resolvable name), along with the username, password, and table name. If connection is successful, table names accessible by the provided credentials will be echoed back in the result window.

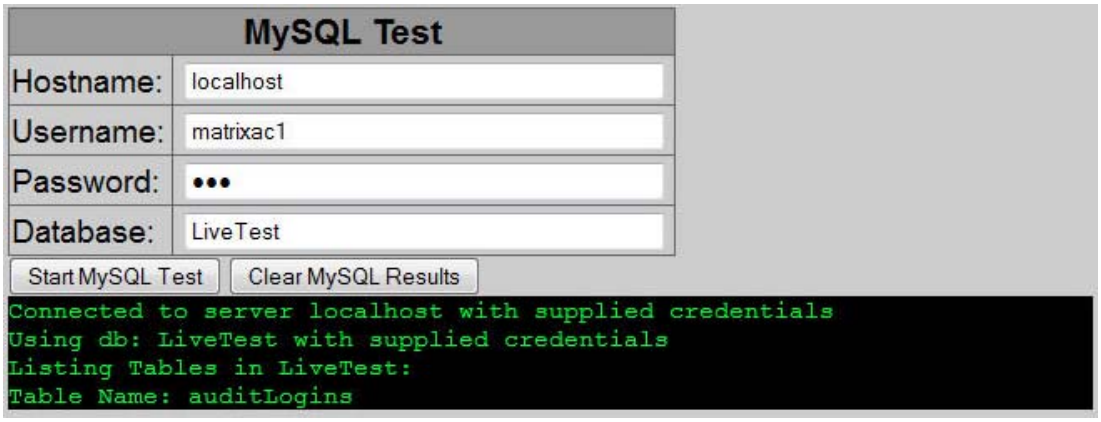

Figure 2-15 - MySQL Test

2.4.7 TIME SETTINGS

The Time Settings page is used to configure time related parameters for the MatrixBBC.

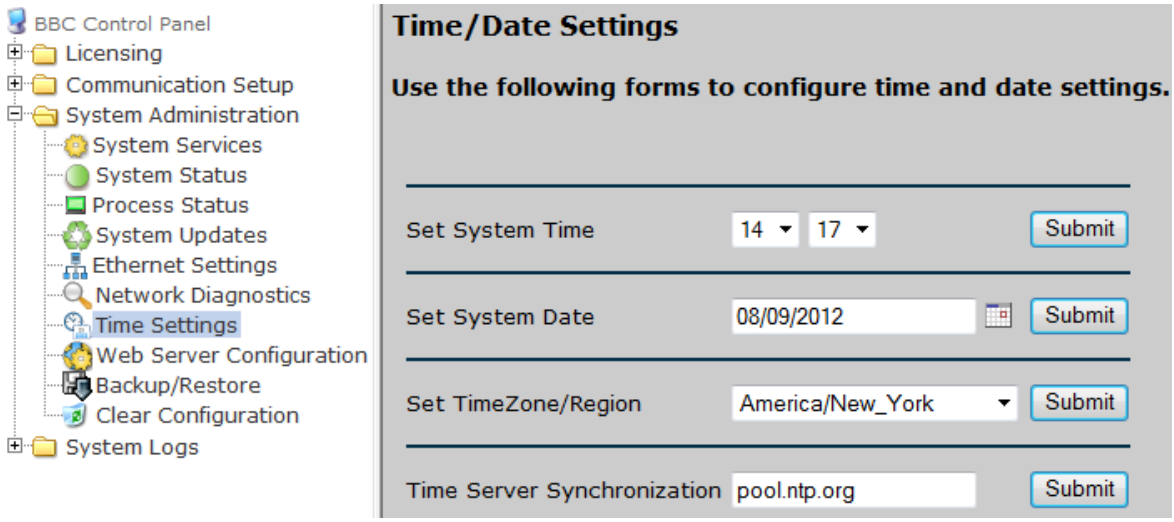

Figure 2-16 - Time and Date Settings

Table 2-4 Time and Date Settings Notes

Property	Notes
Set System Time	Used to set the system's clock time in military format.
Set System Date	Used to set the system's clock date.
Set TimeZone/Region	Used to set the time-zone for the system clock. This should be set to ensure that the MatrixBBC observes daylight savings for your area.
Time Server Synchronization	Defines an NTP server address. This feature requires the MatrixBBC to have some form of connectivity to an NTP server either within your local area network, or via the Internet. The default address is an Internet-based address for time synchronization. This feature is separate from BACnet Time Synchronization.

2.4.8 WEB SERVER CONFIGURATION

The Web Server Configuration page is used to set the port in which the BBC Control Panel is accessed. By default, the MatrixBBC is available using Port 80, however, this can be changed to better secure the system from unauthorized access. A configurable Device Label, which is displayed above the login fields of the MatrixBBC page, is also available to help identify the device.

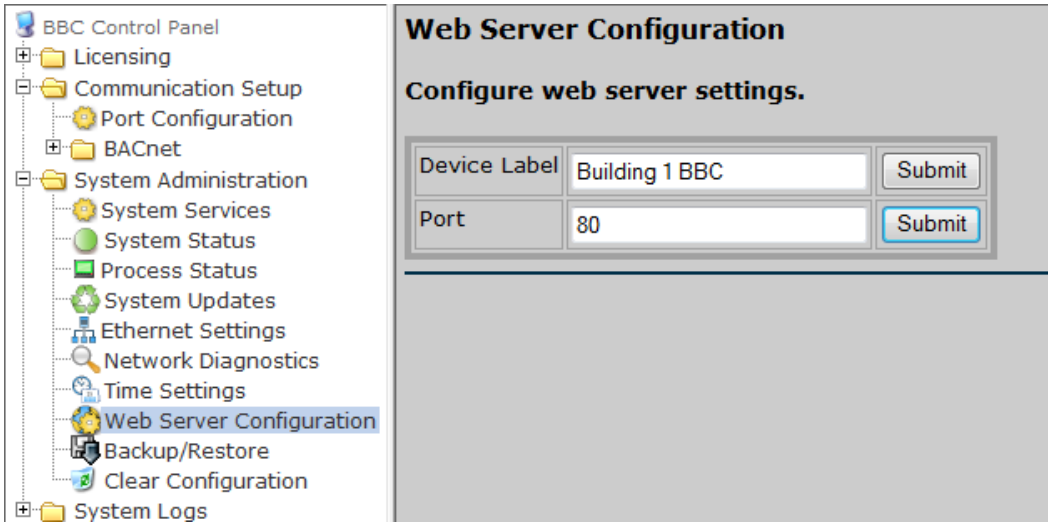

Figure 2-17 - Web Server Configuration

2.4.9 BACKUP/RESTORE

The Backup/Restore page is used to backup and restore all MatrixBBC programming that has been performed using NB-Pro. Using this feature, all created objects and uploaded SPL programs will be saved as a single file and downloaded using your web browser.

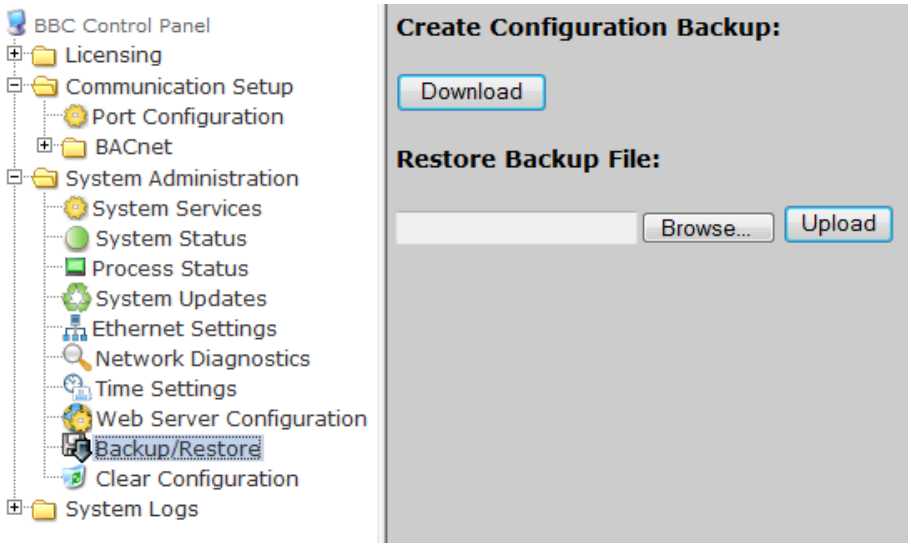

Figure 2-18 - Backup/Restore

To perform a backup of your MatrixBBC, simply click the *Download* button. This will generate the download of a single file which contains all programming for the device.

To perform a restoration of your programming, click *Browse* to find the file on your computer. Once found, click the *Upload* button. In order for your programming to be restored, you must reboot the MatrixBBC via the System Services page.

2.4.10 CLEAR CONFIGURATION

The Clear Configuration page is used to clear the entire database configuration of the MatrixBBC in the event that database corruption, or accessibility to the MatrixBBC cannot occur. To clear the current configuration, click the *Clear* button.

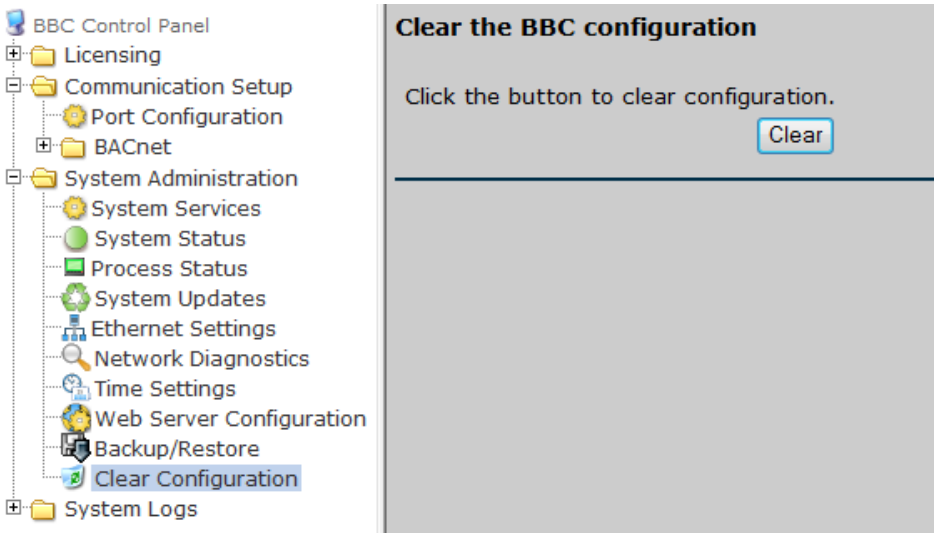

Figure 2-19 - Clear Configuration

This process will produce the same result as a Default Enable command performed on current generation American Auto-Matrix controllers.

2.5 BACNET TIME SYNCHRONIZATION SETUP

By nature, all MatrixBBC controllers are capable of receiving and interpreting BACnet time-synchronization messages from other BACnet devices that act as a time master.

More so, MatrixBBC controllers can be configured to transmit time-synchronization messages to other BACnet devices or networks; effectively making the MatrixBBC a network time master. This may be necessary if you have a network of lower-level devices that do not contain real-time clocks by default (such as NB-VAV, select NB-ASC, or NB-SD devices).

Programming the MatrixBBC to perform this function consists of:

. Configuring time-synchronization-recipients
. Configuring the time-synch-interval

2.5.1 CONFIGURING TIME-SYNCHRONIZATION-RECIPIENTS

The Instance option allows enables the MatrixBBC to send BACnet time-synchronization messages directly to a single BACnet device by referencing its the intended recipient's device instance. Simply enter the device instance number and click Add. To send the reference to the MatrixBBC, click the *Update Value* button on the editor bar in NB-Pro.

The Address option enabled the MatrixBBC to send time-synchronizations to a a single BACnet device, a specific BACnet network, broadcast a time to a single BACnet network, or broadcast to the global network.

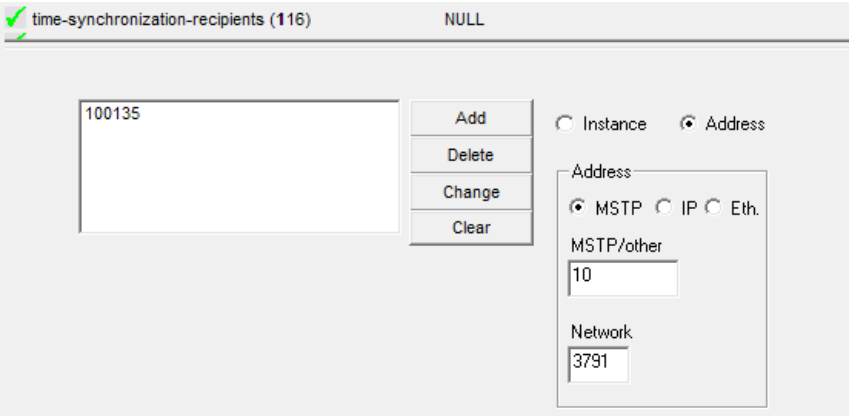

Figure 2-20 - Entering Address References for Time-Synch-Recipients

Within the Address option, you have the ability to specify the address you wish to send to as BACnet MS/TP, BACnet/IP, or BACnet over Ethernet.

The following minor sections provide examples of common configurations that are likely used based on standard network topologies in BACnet.

2.5.1.1 MS/TP NETWORK TIME SYNCHRONIZATION

To send a local MS/TP network time-synchronization:

1. Select MS/TP form the Address area.
2. Enter a value of 255 in the MSTP/other field.
3. Enter a value of 0 in the Network field.
4. Click *Add* to add the entry to the list.
5. Click *Update Value* in NB-Pro to send the value to the controller.

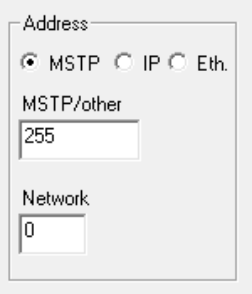

Figure 2-21 - MS/TP Network Synchronization Example

2.5.1.2 IP BROADCAST SYNCHRONIZATION

To send a global BACnet/IP network time-synchronization:

1. Select IP from the Address area.
2. Enter 255.255.255.255 in the IP Address field.
3. Enter 65535 in the Network field.
4. Enter your BACnet/IP Port Number into the Port field.
5. Click *Add* to add the entry to list.
6. Click *Update Value* in NB-Pro to send the value to the controller.

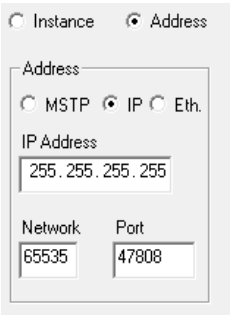

Figure 2-22 - IP Broadcast Synchronization Example

2.5.1.3 ETHERNET BROADCAST SYNCHRONIZATION

To send a global BACnet over Ethernet network time-synchronization:

1. Select Eth from the Address area.
2. Enter FF:FF:FF:FF:FF:FF in the Ethernet MAC field.
3. Enter 65535 in the Network field.
4. Click *Add* to add the entry to list.
5. Click *Update Value* in NB-Pro to send the value to the controller.

Figure 2-23 - Ethernet Broadcast Synchronization Example

2.5.2 CONFIGURING THE BROADCAST TIME SYNC INTERVAL

The frequency of how often time synchronization messages are sent to the network is controlled through the ***time-synch-interval*** property. This property specifies how often, in minutes, the MatrixBBC will send a time-synchronization to the entries defined in ***time-synchronization-recipients***.

Table 2-5: Broadcast Time Synch Interval Details

Property	Valid Range	Notes
time-synch-interval	1 - 1080 (minutes)	A value of 0 disables time-synchronization message transmissions.

2.6 DAYLIGHT SAVING

The MatrixBBC can be programmed to automatically update its clock based on daylight saving time. There are several properties in the controller that correspond to daylight saving, and they can be configured to meet rules in your regional location. By default, daylight saving configuration is disabled.

(RD) Daylight Saving Start Day, ***(RM) Daylight Saving Start Month***, and ***(ST) Daylight Saving Start Time*** control when daylight saving starts in your regional location. Several day options are provided based on universal daylight saving schedules, as well as each month of the year, and a configurable time.

(ND) Daylight Saving End Day, ***(NM) Daylight Saving End Month***, and ***(ET) Daylight Saving End Time*** control when daylight savings ends in your regional location. Similar to the starting properties, the same options are provided for defining end parameters.

Once these parameters are configured, the controller will automatically track when to begin and end daylight saving based on its real-time clock.

CAUTION

While this feature is a 'cut-over' from NB-GPC product technology, the embedded Linux operating system already manages this provided it has connectivity to the Internet in some form.

2.7 MANUALLY CONFIGURING DEVICE ADDRESS BINDINGS

In BACnet, there can be situations where a device may not know of the existence of another device. Some common examples include MS/TP devices that are configured as slaves, or even devices that co-exist on a BACnet-network not reachable using broadcast addressing to resolve the location of the device. While most properly configured BACnet global-networks may never need to worry about manually addressing defining the location of a device, the situation could occur.

The MatrixBBC provides support to allow a programmer to manually define the address and location of a device through the ***device-address-binding*** property. Up to a maximum of 24 manual address bindings can be defined within this property.

To manually define a device, perform the following steps:

1. In NB-Pro, select the ***device-address-binding*** property.
2. Fill in the Device Instance into the first text field, then provide address information in the separated box as shown below.
3. Repeats steps 2 to add additional bindings.
4. Click *Update Value* to send the configuration to the device.

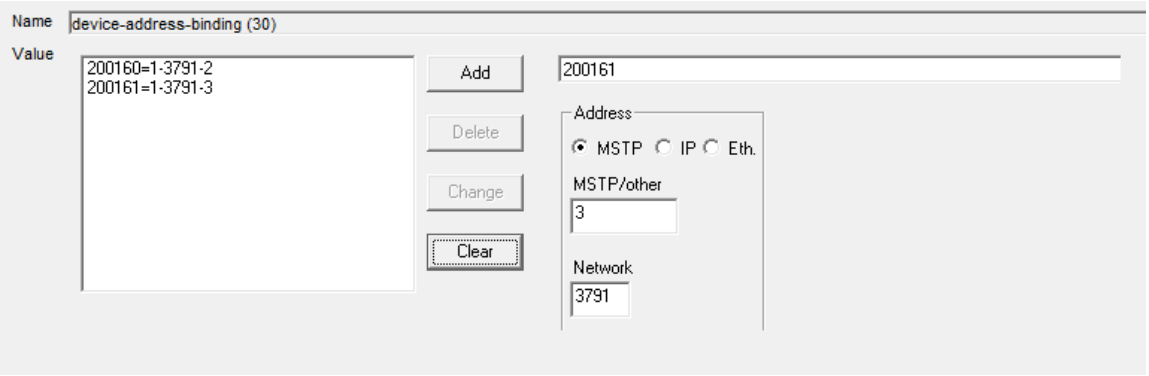

Figure 2-24 - Device Address Binding Configuration

2.8 BACNET MS/TP SLAVE PROXY

The MatrixBBC supports the ability to act as an master proxy for BACnet MS/TP slave devices connected to either RS-485 port. In BACnet, MS/TP devices configured as slaves are unable to respond to the automated discovery processes supported by the protocol, given they do not accept or use the network token for communications.

When configured, the MatrixBBC will answer an auto-discovery request on behalf of each slave connected to it. This permits client software and front-ends such as NB-Pro and AspectFT the ability to perform object-discoveries and "see" the devices on the network.

BACnet MS/TP Slave Proxy support is configured using NB-Pro. The following steps detail how to enable this feature.

2.8.1 ENABLING MS/TP SLAVE PROXY

To enable MS/TP Slave Proxy, perform the following steps using NB-Pro:

1. Access the Device object of the MatrixBBC and find the **slave-proxy-enable** property.
2. By default, slave-proxy is disabled on the MatrixBBC. This property contains two binary flags. The first binary flag corresponds to MS/TP Port 1, and the second binary flag corresponds to MS/TP Port 2.

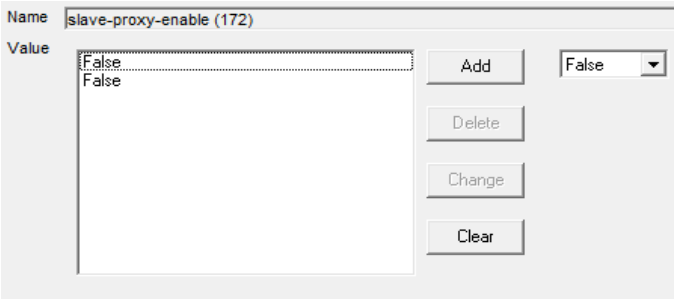

Figure 2-25 - Slave-Proxy-Enable Default Configuration

3. Select the corresponding binary flag for the port. Using the supplied combo box, select True to enable the port. Click *Change* to make the change effective in the list. Repeat these steps for enabling slave-proxy for the other port.

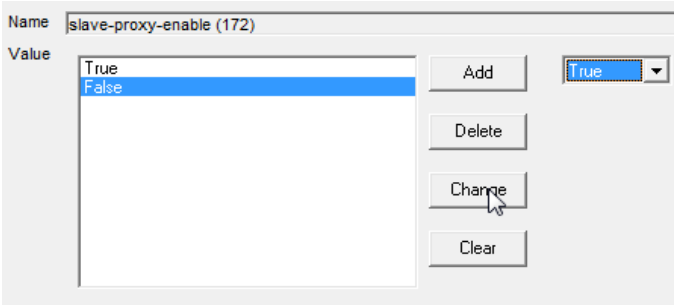

Figure 2-26 - Modifying Slave-Proxy Configurations

4. Click *Update Value* to write the configuration change onto the MatrixBBC.

2.8.2 CONFIGURING THE MANUAL SLAVE ADDRESS BINDING

To allow the MatrixBBC to forward I-Am messages onto the network on behalf of each MS/TP slave node connected to an enabled MS/TP network, you must configure the **manual-slave-address-binding** property to define each slave node that the MatrixBBC provides access to.

To configure, perform the following steps using NB-Pro:

1. In the Device object, locate the **manual-slave-address-binding** property.
2. Fill in the Device Instance into the first text field, then provide address information in the separated box as shown below.
3. Repeats steps 2 to add additional bindings.
4. Click *Update Value* to send the configuration to the device.

MatrixBBC will periodically check for the true existence of an MS/TP slave for the binding you reference in this table. Once verified, the device will then be inserted into the **slave-address-binding** property as a verification that the device is now under proxy.

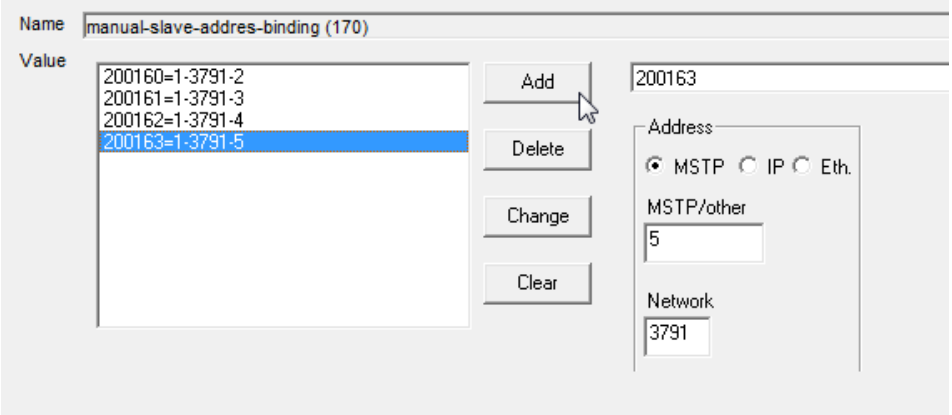

Figure 2-27 - Configuring the Manual-Slave-Address-Binding Table

CAUTION

The device you are referencing must be configured as an MS/TP slave device, otherwise proxy services will not be provided for that device.

NB-ASC Product Family devices can participate provided the product is utilizing firmware revision v6.08 or newer and has been appropriately configured for slave mode.

NB-GPC Product Family devices can participate provided the product is utilizing firmware revision v2.06 or newer and has been appropriately configured for slave mode.

SECTION 3: PROGRAMS AND FILES

This section describes File and Program objects within the MatrixBBC and how they represent certain configuration and programmatic items within the platform. A complete reference of how to use and write SPL programs is also reviewed in this section. The MatrixBBC supports a maximum of 64 Program objects.

IN THIS SECTION

3.1 OVERVIEW

File objects are used to store compiled data for the MatrixBBC to use and execute. There are three specific applications where File objects are utilized.

1. In cases where an SPL Program has been loaded into a File object, a corresponding Program object invokes the file to execute the sequence and provides detailed feedback to users regarding the status of the program.
2. In cases where a site requires customization, LOGO files can be loaded into File objects which allowed connected SBC-STAT devices to display custom logos, list files, or other data.
3. In cases where users wish to apply firmware updates to the MatrixBBC, a designated File object exists that allows users to upload a firmware file.

The following information discusses each application.

3.2 SPL PROGRAMMING

The MatrixBBC supports the ability to accept line-by-line custom programming using the SPL Programming Language. Programs are written in the SPL Editor (located within NB-Pro), and are then uploaded to File objects that exist for each corresponding Program. As the program is loaded, a corresponding Program object provides status feedback regarding the program. The relationship between File and Program Objects is illustrated in the table below.

Table 3-1: File and Program Object Correlation Examples

File Object	Program Object
File, Instance 1	Program, Instance 1
File, Instance 2	Program, Instance 2
File, Instance 3	Program, Instance 3
File, Instance 4	Program, Instance 4
File, Instance 5	Program, Instance 5
File, Instance 6	Program, Instance 6
File, Instance 7	Program, Instance 7
File, Instance 8	Program, Instance 8

3.2.1 CREATING PROGRAMS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. The MatrixBBC will support a maximum of 64 Program objects. Program objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Program object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MI) Max Program Objects** property. By default, this value is set to 0, indicating no Program objects exist.
3. Write the number of total Program objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Programs, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Program objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

3.2.2 LOADING PROGRAMS INTO MATRIXBBC

When a program is initially uploaded to the MatrixBBC using NB-Pro's Upload/Download utility, the program will not execute until a user directly affects the current **program-state**. To initially execute the program, set the program-state property to Load.

Once a program has been loaded, the program will continue to run. In the event that the MatrixBBC controller is reset (via warm-start, restart button, power cycle, etc.), the program will start automatically without user intervention.

3.3 INTRODUCTION TO SPL

The MatrixBBC leverages extension of the SAGE Programming Language to perform custom programming sequences. SPL support has been included in the MatrixBBC to allow users to create line-by-line, BASIC-like written sequences that cannot be carried out using the library of built-in logic objects.

The SPL Programming language support for the MatrixBBC mirrors functionality found in MatrixBBC products and supports a large number of features, summarized below:

. Unlimited number of program lines up to a maximum code block size of 8kb per program.
. Up to 255 custom program properties (initialized using the PROP statement)
. Indirect references within the program up to 256 references
. Floating point math and type coercion
. Re-entrant subroutine calling ability for multiple programs
. Up to sixteen (16) program registers capable of 32 bit primitive BACnet datatype values.
. Six (6) Level expression stack to resolve nested expressions
. Asynchronous read/write of standard and non-standard BACnet object properties

3.4 THE PARTS OF SPL PROGRAMS

An SPL program consists of a collection of structures that are used by the SPL compiler and execution system. An SPL program consists of the following items:

. The program source (SPL) file, which is the written sequence or code.
. The Program Logic Block (PLB) file, which is the compiled source code loaded to the controller.
. The LIST (LST) file for troubleshooting loaded programs that may be aborted or "stuck"

These components are explained in the following sections of this chapter.

3.5 PROGRAM NAMES

SPL programs must have an associated name that is limited only by the operating system. The valid characters for program names are shown below:

. A-Z (uppercase letters “A” through “Z”)
. a-z (lowercase letters “a” through “z”)
. 0-9 (numbers “0” through “9”)
. _ (under bar)
. (space)
. . (period)
. $ (dollar sign)

Program names are case-insensitive, meaning lowercase letters are treated the same as uppercase letters in program object names (e.g., "abc" is the same as "ABC").

3.6 THE .SPL, .PLB AND .LST FILES

SPL programs must reference a Program Logic Block (PLB). The PLB is a binary data file that contains compiled binary pseudocode in a form which can be executed by the controller. This file is created by the SPL Compiler after you create, edit and compile an ASCII text file (i.e., an SPL source file) which contains program logic statements that are easily edited and understood by programmers.

ASCII SPL source code files have the .SPL extension and Binary Program Logic Block files (PLBs) have the .PLB extension.

The name given to the SPL file can be as long as allowed by the operating system on your computer. However, the controllers impose a limitation on file length. When a PLB is loaded into the MatrixBBC, the file name is shortened to 8 characters. If the name was originally longer, only the first 8 characters will be used for the program name.

The source file contains SPL program logic statements which are discussed in detail later in this section. SPL source code can be created and/or edited by using the SPL editor built into NB-Pro. The number of lines in an SPL source file is unlimited. Compiled PLBs cannot exceed 8kb in size.

NOTE

If any errors are generated during the compiling process, a PLB is not created.

You may choose to have the SPL Compilers optionally create a list file during the compile process. The list file is an ASCII text file that contains the source code statements along with the pseudocode and their respective *relative* locations in memory and any error messages generated by the compiler. List files are useful in debugging the execution of SPL programs.

3.7 PROPERTIES AND REGISTERS

Each SPL program may have up to 255 user-defined properties for storing local data for control sequences. These properties, created as non-standard properties, can be addressed at an operator workstation or web-server.

All programs have 16 registers (**%A**-**%P**) and a number of program control properties. To an operator, program registers always begin with a percent sign (**%**) and program control attributes always begin with a dollar sign (**$**). Registers can also be used to hold or present data to an end-user.

3.8 COMPILER CONTROL STATEMENTS

Compiler control statements are non-executable directives to the SPL compiler that it uses to control the generation and format of SPL compiler listings and PLBs. The various SPL compiler control statements are summarized below:

#TITLE "*titletext*"
#NOLABELS
#LABELS
#BBC

Compiler control statements always begin with the pound sign character (#). They must begin in the left-most column.

#TITLE "*titletext*"

The title directive is used to put the specified ***titletext*** at the top of each page of a compiler list file in order to help identify the program logic. The text string ***titletext*** must be enclosed in double quotation marks (**"**) and can be up to 79 characters long.

#NOLABELS
#LABELS

The no labels directive commands the SPL compiler to *not* generate pseudo-code for statement labels in the PLB file. Using this command results in smaller, faster executing PLBs, but eliminates the ability to visually locate labels in the PLB files during troubleshooting. The **#LABELS** command turns on the inclusion of label pcodes in the PLB. In order to conserve RAM and optimize execution, **#NOLABELS** is the default for **#SOLODX, #SOLOGX** and **#BBC**. #**LABELS** remains the default for #**SAGE**. The inclusion of label pcodes can be turned on/off throughout the SPL source file.

#BBC

The **#BBC** commands identify the target platform for the resulting Program Logic Block (PLB), which in this case is the MatrixBBC. A summary of statements, terms, operators and features supported by the compiler for each target is shown in Table 1-1. The **#BBC** command can appear any place in the SPL source file, but it is recommended that they appear early in the source file, i.e., directly after the **#NOLIST** and/or **#PLBxx** commands if they are included.

When it executes, the SPL compiler requires a block of RAM in addition to what the executable SPL module uses. The additional block is used to build the PLB. The amount required depends on the maximum anticipated size of the PLB.

3.9 COMMENTS

All lines in SPL programs that have a semicolon (**;**) in the left most column are comments. They are for documentation purposes only and are ignored at compile time. The generous use of comment statements within your programs will make them more readable and easier to troubleshoot, especially if *you* are not the person doing the troubleshooting.

As a guideline, the top lines of programs should be reserved for program identification comments. This area may contain information such as:

. the name of the program
. the date the program was written
. the name of the author
. what the program does
. the meaning/use of program properties
. the meaning/use of program registers
. any assumptions made by the author
. any input variables used by the program
. any output values calculated by the program
. an *edit trail* indicating any modifications made to the program logic, when they were made, and by whom
. in general, any information that may prove useful to someone looking at the program for the first time

In addition to using comments at the beginning of your program logic, it is also helpful to create a series of comment lines prior to any program segments with logic that may need special explanation. The extra effort you take in adding useful comments to your programs is well worth the benefits you (or someone else) will reap in the future.

3.10 LABELS

SPL programs are composed of one or more *statements* which define the actions and logical operations that the program is to take when it is executed. SPL program statements are grouped into *lines* which contain a single program statement. These lines may be labeled with symbolic names to identify them. Typically this is done so that **GOTO** and other branching statements can refer to them.

Labels cannot use any of the reserved names that identify SPL statements. Labels are case-insensitive, meaning that the label **ABC** is treated the same as the label **ABc** or **abc**. Labels must begin in the left most column of the line. Labels may contain up to eight characters or up to eight characters and numbers. Labels can consist of the following:

. **A** through **Z**
. **a** through **z**
. **0** through **9** (not as the first character)
. **$**, **.** and _ characters.

Labels **cannot** begin with the numbers **0**-**9**. If a line contains any statement following the label, then the statement must be separated from the label by one or more **TAB**s or spaces. Labels may optionally end with a colon character (**:**), which is not counted in the length of the label.

The lines of an SPL program may be labeled with a symbolic name to identify that line, typically so that **GOTO** and other branching statements can refer to it. Labels may be symbolic and numeric, or symbolic and can be up to 8 characters long. Symbolic labels may not use any of the reserved names that identify SPL statements.

The following program fragment demonstrates several labels:

```
                C=[AI1.PRESENT_VALUE]
LABEL1
                A=[AV1.PRESENT_VALUE]
                B=A+3.0
                IF B >32.0 THEN LABEL003
                C=5
LABEL2:         IF [AI13.PRESENT_VALUE] >72.0 THEN LABEL3
                SWAIT 30
                GOTO LABEL2
```

3.11 EXPRESSIONS

Expressions are symbolic formulas which represent a chain of arithmetic calculations on data from various sources. Expressions are used to convey values for parameters in many of the primitive statements in the SPL language. SPL expressions can contain an arbitrary number of *terms* and *operators* which may represent mixed mode arithmetic (i.e., integer, floating point and fixed point). SPL automatically performs type coercion on mixed mode values.

Expression evaluation is performed from left to right. Up to six levels of nesting (i.e., the use of parentheses) may be used in expressions to define an order or *precedence* for evaluation. The expression within the innermost set of parentheses is evaluated first from left to right. This procedure continues outward until the expression within the outermost set of parentheses is evaluated from left to right.

Expressions may contain constants, variables, registers, named object attributes, references, tables, built-in functions and arithmetic and logical operators.

In a very general sense, expressions are composed of terms and operators. In the simplest case, an expression is simply a term with no operators. An expression is defined as follows:

expression ::= term or
expression ::= term operator expression

An expression may also be nested within parentheses and used as a term anywhere within an expression. Up to six levels of parentheses may be used.

The syntax for a nested expression is shown below.

(expression)

The evaluation of nested expressions occurs first from the innermost set of parentheses and continues outward. Expressions that are at similar levels of nesting are simply evaluated from left to right.
The code below shows some complex SPL programming examples using nested expressions.

```
A=MAX(MEAN(B,C,D), MEAN(E,F,G))
H=SQRT((B**2)+(C**2))
[.AA]=([.BB]/255)*[AI1.PRESENT_VALUE]+C
```

Because expressions may contain terms that are objects on networks, an expression does not necessarily have to be completely resolved before execution is passed to another program. Such network accessing is processed asynchronously, causing only the program using the value to be delayed until the value has been fetched. This provides for a fair method in dealing with network-intensive programs, and not penalizing other programs by waiting for a network device object value.

Terms in expressions may be one of several possible types, indicating one of several possible sources of a value to be used during expression evaluation. In general, each term has a data type as well as a value. Data types identify the way in which the values are represented numerically, and may imply additional hidden operations or coercions to be performed when arithmetic operations are required between dissimilar types. The types of terms that can be used in expressions are as follows:

. constants
. named terms
. registers
. program control attributes
. user-defined program attributes
. named object attributes
. BACnet object properties
. references
. virtual attributes
. tables
. functions
. nested expressions

Each type of expression term is explained in detail in the following sections.

3.12 PROGRAM STATEMENTS OVERVIEW

This section is intended to familiarize you with all of the SPL programming statements by organizing them into logical groups based on the functions that they perform. Program statements fall into the following categories:

. attribute definitions and references
. assignment statements
. iteration control, program branching and subroutines
. program delays
. printing, logging and alarming
. job execution
. spooling
. trending control
. program execution control
. execution error control
. debugging statements

Attribute definitions and references are used to declare user-defined program attributes and save the values of attributes to the program's attribute initial value (INI) file.

Assignment statements are used to assign the value of an expression to a variable. This type of program statement is characterized by the use of an equal sign (=).

Iteration control, program branching and subroutines are statements perform a statement or group of statements some number of times, change the order in which the logic is executed, or transfer program control to another portion of the program (a subroutine).

Program delays are statements which suspend program execution, either for a set amount of time or until certain conditions are met.

Printing, logging and alarming refers to statements that give you the ability to print information to a port, log information to a file, or generate formatted alarms of definable alarm classes.

Job execution refers to statements that give you the ability to execute any SAGE^{MAX} job from within the SPL program execution environment.

Spooling refers to commands which offer the ability to send specified files to the printer.

Trending control refers to program statements that can control the execution of trends from a program.

Program execution control refers to statements that can start, stop and prepare programs to be executed.

Execution error control refers to program statements that allow you to define a course of action for PEX when network access errors occur.

Debugging statements refer to programming statements that can be used to aid in the diagnosis of program logic errors.

Each SPL programming statement is individually explained, including sample SPL statements in the following pages.

Table 3-2 Program Statements

Format	Description
variable = expression	assignment statement
ERRORABORT	trap condition - abort on errors
ERRORWAIT	trap condition - wait until no error
symbol EQU *expression*	symbolic equate statement
GOSUB *label*	go to internal subroutine
GOTO *label*	unconditional branch
IF *expr* THEN *label*	conditional branch if expr is true
IF *expr* THEN *label1* ELSE *label2*	conditional branches if expr is true or false
LOOP *register*,*label*	iteration control
MWAIT *expression*	wait a certain amount of minutes
ON *expression* GOTO *label0*,*label1*...*labeln*	indexed conditional branches
ONERROR *label*	trap condition - branch if error occurs
PROP *progproperty,BACnetDatatype*	declares a BACnet property for BACnet based devices
RETURN	return from a subroutine
SECTION *number*	section marker used for debugging
STOP	halt execution of this program
SWAIT *expression*	wait a certain amount of seconds
WAIT (*expression*)	wait until an expression is true, then go on

3.13 ASSIGNMENT STATEMENTS AND EQUATES

3.13.1 STANDARD VALUE ASSIGNMENT

variable = expr

SPL allows various forms of value assignment statement. In each case, a variable on the left side of the **=** (equal sign) is assigned the new value dictated by the expression on the right side. The right side expression produces a value and a data type. Because automatic data type *coercion* may occur during evaluation, the data type of the expression may not match the data type of the variable on the left side. In this case the data type and value from the expression *may* have to be coerced into the variable's data type according to certain rules. The table below summarizes the conversions in general:

Left Side	Right Side	Effect
fixed	float	left=FIX(right)
float	fixed	left=FLOAT(right)
fixed*	fixed	left=RETYPE(right)

*Different fixed data type.

Integers and time data types are treated as fixed types. The table above does not reflect that there are 20 distinct types of fixed types, i.e. 10 decimal point positions each for signed and unsigned types.

When the left side variable is a local program attribute, coercion of the expression into the proper data type is done automatically by PEX. When the left side variable is a register, unless the data type of the result is converted according to the table above, the data type of the register is automatically changed to the data type of the result. When the left side variable is any other type of object, the result *must* be converted according to the table or incorrect values may be assigned to the variable. For example, if the left side variable is a point whose data type is F9H (xxxxxxx.xxx) and the expression has a data type of F7H (xxxxxx.xxxx) then a RETYPE (F9H) must be done so that the value assigned to the variable is not 10 times too large (in this case.)

There are several forms of assignment statements that may be used in SPL. These are summarized below:

register = *expression*	A = B+C
;*programattribute* = *expression*	;CV = B+C
namedobject = *expression*	OAT = B+C
namedobject = *expression*	[ZONE TEMP] = B+C
namedobject = *expression*	[1STFLOOR] = B+C
*****objecttype******namedobject* = *expression*	\VR\OAT = B+C
*namedobject***;***attribute* = *expression*	[LOOP;SP]= B+C
*****objecttype******namedobject***;***attribute* = *expression*	\PT\LOOP;SP = B+C
REF(*expression***)** = *expression*	REF(6) = B+C
&*tablename***(***expression***)** = *expression*	&CLAIREX(29) = B+C

UNS(*x1,x2,x3,x4,x5,attribute***)** = *expression*	UNS(1,0,0,0,FB00h,CV) = B+C

At first there would appear to be a conflict in syntax between local attribute references that are used as variables on the left hand side of assignment statements, e.g., **;AT=***expression*, and comments since both begin with semicolons. The difference in syntax between the two is that comments begin in the left most column and local program attribute references used as variables must have at least one leading space or tab. In order to avoid confusion, local program attribute references can be enclosed in brackets, i.e. **[;AT]**=*expression*.

3.13.2 EQU

symbol EQU expression

EQU (Equate) provides a simple method to assign substitute names to commonly used point references in an SPL program, providing the ability to easily read and interpret an SPL program in a more basic form. **EQU** is a symbolic equate in its rawest form.

EQU statements must be defined in a program *before* they are used, because the compiler considers all terms that are not SPL keywords, numeric values or SPL symbols to be object names. The symbol part of the **EQU** statement can be up to 16 characters in length, which must all be printable characters (A-Z, 0-9, !, @, etc.) and cannot begin with a digit. The right-hand side of the expression can be up to 32 printable characters and can be a programmatic expression or point data location (e.g. FE01;CV).

The code below illustrates the use of the **EQU** statement in an SPL programming example:

```
FANSTATUS EQU [BI1.PRESENT_VALUE]
FANOUTPUT EQU [BO1.PRESENT_VALUE@8]

;start of program
L0:  SWAIT 1
   FANOUTPUT = 1
   SWAIT 5
   FANOUTPUT = 0
```

3.14 ITERATION, BRANCHING AND SUBROUTINES

3.14.1 GOTO STATEMENT

GOTO *label*

where:

label is the label of the point to which program execution will be switched

The **GOTO** statement is an unconditional branch statement that causes program logic to jump to some other location that is identified by a label.
The code below illustrates the use of the **GOTO** statement and shows a sample SPL programming example. It may increase the readability of your program logic if you add a blank line after **GOTO** statements.

```
          :
L1:       C = A+B
          GOTO L3

L2:       C = B-A
L3:       D = C*2
          :
```

3.14.2 IF... THEN... {ELSE...} STATEMENT

IF *expr* THEN *label1* {ELSE *label2*}

where:

expr is the logical expression which determines conditional branching behavior
label1 is the label to jump to if *expr* evaluates to *true*
label2 is the label to jump to if *expr* evaluates to *false* (optional)

The **IF... THEN...** statement is a conditional statement that causes the program logic to jump to some other location identified by a label if a certain condition is true. If the condition is false, execution *falls through* to the next sequential statement. If the optional **ELSE** statement is included, then program execution will jump to the label following the **ELSE** statement if the condition evaluates to *false*.

The code below illustrates the use of the **IF... THEN... ELSE...** statement and shows its usage in an SPL programming example.

```
          IF (DAYOFWEEK==SUN) THEN L3
L0:       IF (A>B) THEN L1 ELSE L2
L1:       C=A+B
          GOTO L3

L2:       C=B-A
L3:       D=C*2
           :
```

3.14.3 ON... GOTO... STATEMENT

ON *expr* GOTO *label0*,*label1*,*label2*,*label3*,....

where:

expr is the expression which determines which label is selected
label0,label1,label2,label3,.... are the labels of the sections to which program control can be switched

The **ON/GOTO** statement is a conditional statement that identifies a series of indexed labels to which PEX transfers control based on the value of an expression. The code below illustrates the use of the **ON/GOTO** statement.

```
            ATTR ER,07
                 :
            ON INT(B-10) GOTO L0,L1,L2
            :ER=1
            PRINT 13,226,"Unsuccessful."
            GOTO DONE

L0:         D = (C+1)/2
            GOTO MERGE

L1:         D = (C+20)/2
            GOTO MERGE

L2:         D = (C+50)/2
MERGE:      PRINT 13,226,"Success. D=%?%",D
DONE:       STOP
```

The indices of the **ON/GOTO** statement are zero-based. In addition, if an index evaluates to a number that is greater than the number of indices, program execution continues with the next line of the program.

3.14.4 LOOP STATEMENT

LOOP *register*,*label*

where:

register is the number of times the loop is to be executed
label is the program label to which execution will jump

The **LOOP** statement is an iteration control statement that performs a “decrement register and jump if not zero” function using a specified register and a program label. The **LOOP** statement is a combination of an assignment statement (e.g., **A = A-1**) and a conditional statement (e.g., **IF A>0 THEN Continue**).

The code below illustrates the proper use of the **LOOP** statement in a sample SPL programming example.

```
A = 100
B = 0
```

```
CALC:   B = REF (A-1)+B
        LOOP A, CALC
        REF (100)=B/100
```

3.14.5 GOSUB STATEMENT

GOSUB *label*

where:

label is the text label which specifies the starting point of the subroutine

The **GOSUB** statement is used to call a subroutine *in the current PLB*. A **RETURN** statement is used to terminate the internal subroutine and return execution control to the statement directly following the **GOSUB**. The subroutine name is actually a label for which all the naming conventions apply.

The code below illustrates the syntax of the **GOSUB** statement and shows its use in a sample SPL program segment:

```
        ATTR AR,0FAH
        A=65
READIT: D=&DuctDiam(A-1)
        GOSUB AREACALC
        &DuctArea(A-1) = ;AR
        LOOP A, READIT
              :
AREACALC: ;AR = PI*(D*D)/4
        RETURN
```

3.14.6 RETURN STATEMENT

RETURN

The **RETURN** statement is used in conjunction with the **CALL** and **GOSUB** statements.

3.15 PROGRAM DELAYS

3.15.1 SWAIT AND MWAIT STATEMENTS

SWAIT *expr*

where:

expr is the number of seconds to delay program execution

MWAIT *expr*

where:

expr is the number of minutes to delay program execution

The **SWAIT** and **MWAIT** statements are used to cause a timed delay in program execution. These statements each have a single argument which represents a number of seconds or minutes (respectively) that must pass before program execution continues. The time delay can be viewed as it counts down from the **$D** program control attribute. This attribute shows all time delays in seconds. Once the delay reaches zero, the next program statement is executed.

The code below illustrates the proper use of the **MWAIT** and **SWAIT** statements. Sample SPL programming examples are also shown.

```
L0:     IF (SWITCH==1) Then L1
        MWAIT 5
        GOTO L0

L1:     [PROG1;MN]=[PROG2;SP]+5.0
```

3.15.2 WAIT STATEMENT

WAIT *expr*

where:

expr is the logical expression that will determine when the **WAIT** will finish

The **WAIT** statement is a conditional statement that halts further program execution until the expression specified in the argument is true.

The code below illustrates the syntax of the **WAIT** statement and shows a sample SPL programming example.

```
L0:     WAIT $ALARMS
        CALL Notify

L1:     CALL Clear, STICK
        IF $ALARMS THEN L1 ELSE L0
```

3.16 EXECUTION ERROR CONTROL

3.16.1 ERRORABORT STATEMENT

ERRORABORT

The **ERRORABORT** statement is an error control statement that causes the program executor to abort the program when any trappable or non-trappable error is detected. (See also *Section 3.16.2*:*ERRORWAIT Statement* and *Section 3.16.3*:*ONERROR Statement*).

There can be multiple **ERRORABORT** and **ERRORWAIT** statements within a program. This allows the aborting of errors to be turned on and off. Unless an **ERRORWAIT** statement is included in a program, the **ERRORABORT** statement is in effect.

All errors that are not trappable (e.g., no such object name, invalid operation, etc.) will always cause the program to be aborted.

The code below illustrates the syntax for the **ERRORABORT** statement and shows its use in a simple SPL program segment:

```
ERRORABORT
[AI1.PRESENT_VALUE]=55.255
```

3.16.2 ERRORWAIT STATEMENT

ERRORWAIT

The **ERRORWAIT** statement is an error control statement that allows the programmer to specify what PEX should do when it encounters a trappable error. If the **ERRORWAIT** statement is included in a program and PEX detects a trappable error, then the statement that caused the trappable error is re-executed forever until the error condition no longer exists

There can be multiple **ERRORWAIT** and **ERRORABORT** statements within a program. This allows the aborting of errors and error waiting to be staggered throughout the program. Unless an **ERRORWAIT** statement is included in a program, the **ERRORABORT** statement is in effect.

The code below illustrates the syntax of the **ERRORWAIT** statement and shows it being used in an SPL program segment:

```
ERRORWAIT
[AI1.PRESENT_VALUE]=55.255
      :
```

3.16.3 ONERROR STATEMENT

ONERROR *label*

where:

label is the label of the code to be executed when a trappable error occurs

The **ONERROR** statement identifies a label to which PEX transfers control whenever it detects a *trappable* error (see **Appendix B**). The **ONERROR** statement is in effect only for the statement that precedes it. (See also *Section 3.16.1*:*ERRORABORT Statement* and *Section 3.16.2*:*ERRORWAIT Statement*). When an error is detected, the error code is placed in the program's **$E** control attribute by PEX. **ONERROR** statements take precedence over **ERRORWAIT** statements. The **$E** program control attribute should be reset to zero before leaving the error code handler.

The code below illustrates the syntax of the **ONERROR** statement and shows an SPL programming example.

```
Getit:    ;$E = 0
          A = ZONE_TEMP;CV
          ONERROR Err
L1:       B=A+10
               :

Err:      IF (;$E<>5) THEN END
          A=72.0
          GOTO L1

End:      STOP
```

NOTE

The **ONERROR** statement can only be used with trappable errors such as a timeout, a CRC or checksum error, NAK responses, data rejection, temporarily blocked states, dialer busy states and failed to connect errors. Any other program execution errors cause the program to abort.

3.17 DEBUGGING STATEMENTS

3.17.1 SECTION STATEMENT

SECTION *number*

where:

number is the number designation given to the section

The **SECTION** statement is a debugging statement that stores the *number* argument in the **$S** program control attribute of the program. This command can be placed strategically at multiple locations in the program to be debugged. By using unique *numbers* in the statements, you can track the progress of the program through various logical sections by monitoring the **$S** program control attribute.

The code below illustrates the syntax of the **SECTION** statement and shows an SPL programming example:

```
        SECTION 1
        A = REF (0)
L1:     SECTION 2
        B = B + REF (A-1)
        LOOP L1
        SECTION 3
            :
```

3.18 PROGRAM CONTROL PROPERTIES

All program control properties provide the means to exmaine, monitor, and troubleshoot a compiled program. The following table provides a high level overview of each property, its use, and capabilities.

Table 3-3 Program Control Attributes

Control Properties	Meaning	
program-state	Program State Enumerations 0=Idle 1=Loading 2=Running 3=Waiting 4=Halted 5=Unloading	Program State defines the current activity of a program which may be executing logic. ***Idle*** - defines that the program object is not actively executing compiled programming code. This state may appear when no program has been uploaded to the corresponding file region, when the program has not been loaded into memory to run. ***Loading*** - defines that a request has been made from the network through the program-change property to load the program for execution. This state will continue to appear until the program has been fully loaded into upper memory for execution. ***Running*** - defines that the program is actively executing logic. ***Waiting*** - defines that the program is currently in a waiting condition. This can occur upon the execution of an SWAIT, MWAIT, or the wait of a response from a remote controller for information. ***Halted*** - defines that the program has halted the execution of logic. This can occur if a compiled program aborts due to a trappable error, an error in developed logic, or other local matter which can be troubleshot using other properties corresponding to the program object. A program may also be halted through a human request by direct editing of the **program-change** property. ***Unloading*** - defines that the program is being actively unloaded from upper memory. This occurs when an Unload request is made to the **program-change** property.
program-change	Program Change Enumerations 0=Ready 1=Load 2=Run 3=Halt 4=Restart 5=Unload	Program Change enumerations are used to user command a program into a specific state. For example, this property can be written to restart a program, manually halt a program, etc. ***Ready*** - defines that the program is ready to have information loaded for execution or debugging. This is a read-only state and cannot be commanded. ***Load*** - used to load a program into upper memory for execution. ***Run*** - used to force a program to run logic. This command is commonly only used for Single-Step Mode Debugging, or can be used to command a program to run when it executes a STOP statement. ***Halt*** - used to stop a program from executing logic. ***Restart*** - used to restart a program from the very first line of logic. ***Unload*** - used to unload a program from upper memory.

Table 3-3 Program Control Attributes

reason-for-halt	Reason for Halt properties provide a generalized reason for a program that may have been halted. Reason for Halt Enumerations 0 = Normal 1 = Load Failed 2 = Internal 3 = Program 4 = Other ***Normal*** - defines that the program is in a healthy condition to execute logic. ***Load Failed*** - defines that the program failed to load due to resource constraints, or a possible OS issue. ***Internal*** - defines that an internal error caused the program to halt. Further properties listed below can be used as an aide in troubleshooting why the program halted. ***Program*** - defines that the program halted due to a Halt command being written to the **program-change** property. ***Other*** - defines that an external condition caused the program to halt
description-of-halt	Provides a generic, English-readable reason for why the program has halted.
program-location	Provides a string-based message which provides the current location or portion of code that the program object is executing. Note that the 'Sec #' portion of this property provides the section number of logic being executed, provided that you have implemented SECTION statements into your program logic. The second portion of the string is a hex offset which can be referenced in the.LST file of your compiled program for troubleshooting.
description	Provides a gen earl description that can be assigned to your program.
$instance-of	Provides a reflection of the name of the program that is loaded.
status-flags	Provides general health information for the program object.
reliability	Provides reliability information regarding the health of the program object.
out-of-service	Always reports false and cannot be modified.
profile-name	Provides a profile-name reference for the MatrixBBC.
($1) Enable Single Step Mode	Debugging point for troubleshooting and testing your program. When set to Yes (1), and a program has been loaded, lines of your SPL program are executed one line at a time. Once a line has been executed, use the program-change property and set it to Run (2) to have the program execute the next line of code. Once you have finished debugging your SPL program, set this property to No (0) to allow the program to execute all lines automatically.
($D) Delay Time Remaining	Provides a count down timer when the program currently executes and observes an SWAIT or MWAIT statement. Countdown information is displayed as time in seconds.
($E) Error Code	Provides a generic error code for troubleshooting.
($F) Device Instance Not Found	Provides feedback of a device that may not be reachable using remote fetching commands when executing. If your program uses an ERRORWAIT statement and a timeout occurs when fetching information from a remote device, this property will reflect the device instance for the BACnet device that cannot be communicated with. If your program uses ONERROR statements, the last device failed to be communicated with will be reported through this property.

Table 3-3 Program Control Attributes

($W) Trappable Error Action	Provides read-only status as to whether or not the program is subject to abortion if an error is encountered. Trappable errors are commonly catch-able when using ERRORWAIT and/ or ONERROR statements in your program. If these commands are present, this property will reflect that the program will Wait on Error, otherwise, Abort on Error.
($N) Number of Program Properties	Provides feedback relative to the number of custom properties that have been generated or created through use of the PROP statement within a loaded program.

The **$S** control attribute is another special control attribute which reflects the current section number of the program. The **SECTION** statement is used to set a portion of the **program-location** property to reflect the section number that the program is currently executing. This control attribute can be used in diagnosing errors in your program logic. By using **SECTION** statements at strategic locations in the logic (e.g., before and after loops, conditional statements, calls to subroutines, etc.), you can check the progress of the program execution through examination of the program-location property. The example below illustrates the use of **SECTION** statements.

```
SECTION 1
CALL INITIALIZE
SECTION 2
CALL CALC_LOOP
SECTION 3
CALL PROCESS_LOOP
SECTION 4
CALL SUBMIT_JOB
SECTION 5
STOP
```

3.19 USING SPL WITH BACNET OBJECTS

SPL has features designed specifically for creating program that work with BACnet devices. From within your program, you may define custom properties. These properties can be used within the program and are also visible to other controllers on the BACnet network just like any other property in the controller. Statements exist which allow you to reference properties that exist on the host controller as well as on other controllers on the network. Functions exist that allow you can generate object identifiers during program execution. These features combine seamlessly to allow you to work with any BACnet properties from within your SPL program.

3.20 FUNDAMENTALS OF SPL IN BACNET

The following section illustrates standard fundamentals for writing SPL programs for the MatrixBBC. While the information in the previous sections provide explicit information behind the underlying functionality of each statement and its usage, this section will provide a more simplistic approach to learning how to write SPL.

While the following information provides only a few statements, the majority of existing functions in SPL can obviously be used with BACnet.

3.20.1 THE PROP STATEMENT

The **PROP** statement is equivalent to using *ATTR* in PUP-based devices, where **PROP** allows users to create local program properties initialized to one of the twelve (12) primitive BACnet data types supported throughout the standard. User-defined properties must be declared before any other SPL statements with the exception of compiler control statements such as **#BBC**. The syntax to define a property is:

PROP *propertyname, datatype=xxx*.

where

. ***propertyname*** is a numeric or two-letter reference for the property.
. ***datatype*** is a keyword for one of the twelve primitive BACnet datatypes such as REAL, UNSIGNED, NULL, BOOLEAN, etc. Note that BACnet does not support PUP datatypes (e.g. 0FEh, 254, etc).
. ***=XXX*** is an initial value assignment (this can be placed in optionally).

In addition to being available as arguments for assignment and expression statements, all declared properties are visible to the BACnet network in the form of non-standard properties of the Program Object associated with the program. If you wish to access these properties using a BACnet device manufactured outside of American Auto-Matrix, please make certain that the device supports the ability to address non-standard objects and properties.

3.20.2 PROP STATEMENT EXAMPLES

The following provides examples of how to use the **PROP** statement, with information on how to initialize values for each specific datatype assignment.

3.20.2.1 FLOATING POINT DATATYPE CREATION

For floating point datatypes, use **REAL**. **REAL** is a 32-bit IEEE floating point value, typically used for present-value in Analog Input, Analog Output, and Analog Value objects, as well as set points, and any other type of point that is of a floating nature (contains a decimal).

```
PROP 10001, REAL = 65.0
```

3.20.2.2 UNSIGNED INTEGER DATATYPE CREATION

. For unsigned Integer datatype, use **UNSIGNED**.

```
PROP 10002, UNSIGNED = 11
```

3.20.2.3 SIGNED INTEGER DATATYPE CREATION

. For signed Integers, use **INTEGER**.

```
PROP 10003, SIGNED = 6
```

3.20.2.4 TEXT PROPERTY DATATYPE CREATION

. For text properties (character strings), use the term **CHARSTRING**.

```
PROP 10004,CHARSTRING = 0,64,"THIS IS MY TEXT PROPERTY"
```

In the value declaration, the value of zero (0) defines the character string set used for the text property. This value must always be zero (0). The value of 64 limits the size of the text property value to 64 characters. All text properties must have a value defined in order for the program to compile. Text properties are primarily to be used for read-only applications, and cannot be assigned different values from within your SPL program.

3.20.2.5 BITSTRING PROPERTY DATATYPE CREATION

For bitstring properties, use the term **BITSTRING**.

```
PROP 10005,BITSTRING = 5,0b10101
```

In the value declaration, the value of five (5) defines the number of bits for the initialized value. If you attempt to define more bits than the size setting, your program will not compile. All bitstring properties must have a value defined in order for the program to compile. The MatrixBBC will support up to a maximum of 32 bits in a bitstring value for any property.

When a custom bitstring is viewed by a client or other front-end, all 32-bits will be returned.

3.20.2.6 TIME PROPERTY DATATYPE CREATION

For time properties, use the term **TIME**.

```
PROP 10006,TIME = 15:30:00.00
PROP 10007,TIME = 16:00
```

Time properties can be declared values in *Hour:Minute* format, or *Hour:Minute:Second.Millisecond* format. Most applications used in American Auto-Matrix Native Series products use *Hour:Minute* format.

3.20.2.7 DATE PROPERTY DATATYPE CREATION

For date properties, use the term **DATE**.

```
PROP 10008, DATE = 0d20051225
```

Date property values are initialized uniquely in BACnet, when compared to PUP applications. The general format is *0dyyyymmdd*, where *yyyy* is the year, *mm* in the month, and *dd* is the day-of-the-month. The example provided above represents December 25, 2005.

3.20.2.8 ENUMERATED PROPERTY DATATYPE CREATION

For enumeration properties, use the term **ENUM**.

```
PROP 10009, ENUM = 2
```

Enumerated property values are typically used for multiple choice assignments in standard BACnet properties such as the Units property, or present-value of Multi-State object type.

3.20.2.9 NULL PROPERTY DATATYPE CREATION

For **NULL** properties, use the term **NULL**.

```
PROP 10010, NULL
```

A **NULL** Datatype is typically used in SPL to assist with relinquishing control of an Analog Output or Binary Output that was written to at a certain priority. No initial value can be given to a **NULL** property because the datatype reflects no assigned value.

3.21 WORKING WITH OBJECT PROPERTIES

Accessing objects and properties in BACnet using SPL can be done in a variety of methods. The following section reviews the various methods of how to access objects and properties.

3.21.1 REFERENCING OBJECTS

SPL can access objects specifically by pre-defined object references. **Appendix E2** provides a table of supported BACnet Objects, and their predefined SPL object references.

3.21.2 REFERENCING PROPERTIES

SPL can address standard properties by using pre-defined property references. **Appendix E3** provides a table of the standard BACnet properties, and their pre-defined SPL property references. For non-standard properties in the MatrixBBC, you may use either the two-letter reference for the property (e.g. SP, AE, etc), or the numeric BACnet property identifier.

3.21.3 ADDRESSING OBJECT PROPERTIES

When addressing object properties in SPL, the following format must be used:

[*objectID.property*]

where

. objectID references the pre-defined object reference and its Object Instance number
. property references the pre-defined property reference or numeric BACnet property identifier.

A period (**.**) must separate the objectID and property references.

The following examples are illustrated:

```
[AI1.PRESENT_VALUE]
;references Analog Input with Instance of 1
[BI2.PRESENT_VALUE]
;references Binary Input with Instance of 2
[DE800818.SYSTEM_STATUS]
;references Device object with device instance of 800818
```

In many applications, you will typically deal with an object's PRESENT_VALUE property, as this is the most commonly accessed property in BACnet devices.

However, when you are working with proprietary properties (sometimes referred to as non-standard), SPL requires you to reference the BACnet identifier number for the proprietary property of the object you are referencing. In AAM controllers, property identifiers for proprietary properties can be found in the controller's respective user manual. When you are addressing proprietary properties for a MatrixBBC controller or an NB-GPC controller (whether local or remote over an MS/TP network connection), you may simply use the two-letter alias that is assigned to it. However, if you are working with an ASC-family device or a third-party BACnet controller that contains proprietary properties, you will need to use the BACnet identifier number.

If you choose to work with the numeric BACnet property identifier or are addressing proprietary properties, the following can be used:

```
[AI1.47410]
[BI2.85]
[DE800818.16520]
```

3.21.4 ADDRESSING USER-DEFINED PROPERTIES

To reference user-defined properties created at the top of your program, the following format must be used:

[*.property*]

where

. .property is the property identifier declared.

By referencing no object, SPL will look inside its own program for the property reference.

```
#BBC
;
            PROP 10001,REAL
            PROP 10002,REAL
;
L0:         [.10001] =13.00
            [.10002] = 16.25
```

3.21.5 PEER-TO-PEER ADDRESSING

SPL allows users to perform peer-to-peer transactions on the MSTP sub-network, as well as BACnet/IP or BACnet/Ethernet network(s) that the controller resides on. To address an object property from a remove device, the following format must be used:

[####*.objectID.property*]

where

. **####** is the Device Instance of the Device you wish to access.
. ***ObjectID*** is the Object Type and Instance.
. ***property*** is the property of the object.

The following example illustrates this function:

```
#BBC
;
L0:         A = [12345.AI1.PRESENT_VALUE]
```

When accessing object properties from remote devices, users should place SWAIT statements of about 3 seconds between each peer-to-peer network transaction that is made. This allows for the device to receive the token from the network. If you declare ERRORWAIT, SPL will trap an MSTP communication timeout if encountered.

If you wish to access non-standard properties in ASC family devices remotely, you must always use the numeric BACnet property identifier. Numeric BACnet property identifiers for each property can be found in the back of the corresponding device you are using, or through various utilities in NB-Pro.

3.21.6 WRITING VALUES TO OBJECT PROPERTIES

Writing values to object properties is dependent on the datatype of the property you are working with. By general nature, BACnet SPL can write to numeric based data types by simply placing an equal sign after the object property and declaring your value. Datatypes that are acceptable to the right-side of the equal sign are as follows:

. REAL
. UNSIGNED
. INTEGER
. TIME
. DATE
. BOOLEAN
. ENUM
. DOUBLE

The following example provides this action:

```
[AI1.PRESENT_VALUE]=75.2
[AV2314.PRESENT_VALUE]=64.0
```

Similar to proprietary PUP applications, you can utilize traditional variable assignment routines. Keep in mind that some datatypes need to be equated to a user-defined property configured for the same datatype if one wishes to modify its value through SPL. The primary datatype that must follow this format is BITSTRING.

The following example illustrates how to write to BITSTRING datatypes:

```
#BBC
;
            PROP 10001,BITSTRING=8,0b10101010
;
L0:         [GPCSCHED1.AD] = [.10001]
```

3.21.6.1 WRITING WITH COMMAND PRIORITIZATION

In BACnet, it is possible for many different devices to try to modify the same device's object property values. If multiple devices tried to write to the same object property, errors could occur and values could be set incorrectly. To avoid this, BACnet uses priority arrays to determine the order in which property changes will be performed.

A priority array assigns the unique levels of priority to the different types of devices that could write to a device. There are 16 prioritization levels with 1 being highest, and 16 being lowest. A complete list of BACnet Priority Array Levels and their uses is given below:

Table 3-4: Priority Array Levels

Priority Level	Application	Priority Level	Application
1	Manual-Life Safety	9	Available
2	Automatic-Life Safety	10	Available
3	Available	11	Available
4	Available	12	Available
5	Critical Equipment Control	13	Available
6	Minimum On/Off	14	Available
7	Available	15	Available
8	Manual Operator	16	Available

Valid Objects that need to be commanded with Priority Array are as follows:

. Analog Output
. Analog Value (if commandable)
. Binary Output
. Binary Value (if commandable)

To write to one of the above objects using Priority Array, you must place an at sign (@) followed by the level of priority you wish to write with inside the object property reference. If the (@) is not specified in the syntax, priority level 16 is assumed. The following example illustrates:

```
#BBC
;
L0:         [AO1.PRESENT_VALUE@2]=100.0
            [BO1.PRESENT_VALUE@2]= 1
```

To relinquish control, you must equate the object property at the same priority to a user-defined property with a NULL datatype. The following example illustrates:

```
#BBC
            PROP 10013,NULL
L0:         [AO1.PRESENT_VALUE@2]=[.10013]
            [BO1.PRESENT_VALUE@2]=[.10013]
```

3.21.7 DATA TYPE SENSITIVITY WITH BACNET SPL

In comparison to PUP applications, datatypes in BACnet are mostly 32-bit, which results in sensitivity when writing data through SPL. In SPL, the following items are the most common error when writing BACnet SPL.

. When writing to floating point values, you must include a decimal place. If you do not include a decimal place, your SPL program could abort.
. When performing math functions in SPL, you must use the same datatypes. For example, if you try to add an unsigned value of 15 to a time of 15:00 to get 15:15, this will not work. You must add two times together in order to come to a realistic result. Operating outside of this rule will result in aborted SPL programs
. Follow the rules listed with each datatype listed within this manual. For example, you can only write to bitstring properties by equating a property to a user-defined property.

3.21.8 EQU FUNCTION LIMITATIONS IN BACNET SPL

When using the EQU function with BACnet-based SPL programs, you may use the function to reference commonly accessed objects within your program. Unlike PUP-based SPL, you cannot use EQU functions to write to commandable objects such as Analog Outputs, Analog Values, Binary Outputs and Binary Values in the MatrixBBC. While the EQU statement can accommodate addressing commandable object types, this functionality is limited to being used for read commands, rather than write commands.

3.22 OBJECT SYNTAX REFERENCE

The following table provides a syntax reference for how objects are addressed in MatrixBBC products. Object aliases are available not only for interacting with GPC logic blocks that can be created in the MatrixBBC, but also objects that reside on remote GPC v2.0 products, as well as remote GPC v1.x products and NB-ASC controllers. Careful consideration should be taken into account relative to how are addressing objects.

Table 3-5 Object Syntax Reference

Object Type	SPL Syntax Reference
Analog Input	AI
Analog Output	AO
Analog Value	AV
Binary Input	BI
Binary Output	BO
Binary Value	BV
Multi-State-Input	MSI
Multi-State-Output	MSO
Multi-State-Value	MSV
Calendar	CAL
Device	DE
Event Enrollment	EE
File	FI
Group	GR
Loop	LP
Notification Class	NC
Program	PG
Schedule	SC
Average	AVG
Trend Log	TR
Life Safety Point	LSP
Life Safety Zone	LSZ

Table 3-5 Object Syntax Reference

Object Type	SPL Syntax Reference
UI Summary (GPC v2.0)	GPCBTLUISUMMARY
DO Summary (GPC v2.0)	GPCBTLDOSUMMARY
AO Summary (GPC v2.0)	GPCBTLAOSUMMARY
DI Summary (GPC v2.0)	GPCBTLDISUMMARY
SSB Summary (GPC v2.0)	GPCBTLSSBSUMMARY
Schedule Summary (GPC v2.0)	GPCBTLSCHEDULESUMMARY
Input Select (GPC v2.0)	GPCBTLINPUTSELECT
Mix Max Average (GPC v2.0)	GPCBTLMINMAXAVG
Math (GPC v2.0)	GPCBTLMATH
Logic (GPC v2.0)	GPCBTLLOGIC
Remap (GPC v2.0)	GPCBTLREMAP
Piecewise Curve (GPC v2.0)	GPCBTLPCURVE
Scaling (GPC v2.0)	GPCBTLSCALING
Netmap (GPC v2.0)	GPCBTLNETMAP
Enthalpy (GPC v2.0)	GPCBTLENTHALPY
Staging (GPC v2.0)	GPCBTLSTAGING
Comm Status (GPC v2.0)	GPCBTLCOMMSTATUS
Season (GPC v2.0)	GPCBTLSEASON
Statbus (GPC v2.0)	GPCBTLSTATBUS
PID Control (GPC v2.0)	GPCBTLPID
Pulse Pair PID (GPC v2.0)	GPCBTLPULSEPAIR
Thermostatic Control (GPC v2.0)	GPCBTLTHERMCTRL
Timers (GPC v2.0)	GPCBTLTIMERS
Broadcast (GPC v2.0)	GPCBTLBROADCAST
STATbus (GPC v1.x)	GPCSTATBUS
UI Summary (GPC v1.x)	GPCUISUMMARY

Table 3-5 Object Syntax Reference

Object Type	SPL Syntax Reference
DI Summary (GPC v1.x)	GPCDISUMMARY
AO Summary (GPC v1.x)	GPCAOSUMMARY
DO Summary (GPC v1.x)	GPCDOSUMMARY
Occupancy (GPC v1.x)	GPCOCCUPANCY
Motor Control (GPC v1.x)	GPCMOTORCONTROL
Thermostatic Control (GPC v1.x)	GPCTHERMCTRL
PID Control (GPC v1.x)	GPCPID
Scaling (GPC v1.x)	GPCSCALING
Piecewise Curve (GPC v1.x)	GPCPCURVE
Logic (GPC v1.x)	GPCLOGIC
Math (GPC v1.x)	GPCMATH
Min Max Average (GPC v1.x)	GPCMINMAXAVG
Input Select (GPC v1.x)	GPCINPUTSELECT
Broadcast (GPC v1.x)	GPCBROADCAST
ASC Economizer	ASCECONOMIZER
ASC PID Control Loops	ASCPID
ASC Occupancy Control	ASCOCCUPANCY
ASC Proof of Flow	ASCPROOFOFFLOW
ASC Broadcast	ASCPULSE
ASC Flow Setpoints	ASCBROADCAST
ASC Reheat Control	ASCREHEAT
ASC Valve Control	ASCVALVECTRL
ASC Damper Control	ASCDAMPER

3.23 ADVANCED BACNET SPL FUNCTIONS

3.23.1 THE OID FUNCTION

OID(*objecttype,instexpr*)

where:

objecttype is a numeric object identifier number or SPL object reference
instexpr is an expression for instance

The **OID** function is used to compute object identifier numbers from within an SPL program. In many instances the desired object type will be known, but the object instance will be determined during the program's execution. This would occur, for example, if you knew you wanted to read from an analog input, but you wanted the particular input chosen when the program is run.

The **OID** function will return the object identifier number for the object specified by the object type and instance number entered into the *objecttype* and *instexpr* arguments. The *objecttype* argument can either be the numeric object identifier number for that type of object or the SPL keyword used to refer to that type. The *instexpr* argument can be any expression which results in a positive integer value. A complete list of both the standard BACnet object types as well as the AAM proprietary object types, their object identifier numbers for each, and the SPL Object References for each are given in **Appendix E**.

As an example, to compute the object identifier number for the third instance of the analog output object you could use the numeric value for the object identifier number and write

```
OID(1,3)
```

or you can use the SPL keyword AO to specify the analog output

```
OID(AO,3)
```

Similarly, you could use a program register to decide which object to use. If you wanted to look up the object identifier number for the analog output whose instance number was stored in the B register for the program you could write

```
OID(AO,B)
```

The **OID** function can be combined with the **BACNET** statement to programmatically read properties from, or write properties to, any controller on the MSTP sub-network.

3.23.2 THE BACNET STATEMENT

The **BACNET** statement is used to reference a property on the BACnet network. Your programs can read values from, or write values to, the referenced property using the **BACNET** statement. The syntax for the **BACNET** statement is as follows:

BACNET(*devexpr,objexpr,propexpr,instexpr*)

where

devexpr is an expression whose value specifies the device object instance of the device containing the property to be read or written.

objexpr is an expression specifying the object identifier number of the object whose property is to be read.
propexpr is an expression for the identifier number for the chosen property.
instexpr specifies an array index for use in cases when array properties are being read.

When working with the **BACNET** statement the special value -1 (0xFFFFFFFF) can be used for *devexpr* and *instexpr*. When used in *devexpr*, the value -1 means "this device" and is used to refer to properties present in the device in which the SPL program is running. When used in *instexpr*, a value of -1 indicates that no array index is provided. A value of -1 should be entered for *instexpr* whenever you are referencing a single value. The *instexpr* term is only used for bit string, character string, and octet string properties.

3.23.2.1 READING VALUES WITH THE BACNET STATEMENT

The **BACNET** statement can be used to read values from any device connected to the MSTP sub-network. To read a value, you must specify the device, object identifier number, property identifier number, and index of the property to be read. Each of these values may be functions or expressions, allowing you to determine the property to be read at the time of executions.

For example, if you wanted to read the value of the property who's identifier number was 85 from an object located on the same controller who's identifier number was 127.

```
BACNET(-1,127,85,-1)
```

Here the *devexpr* is -1 because the object is on the same device, *objexpr* is the object identifier number 127, 85 is the identifier number of the property we wish to read, and the value of -1 is included because the property is not an array and, therefore, has no index value.

The combination of the **OID** and **BACNET** statements is particularly useful. The **OID** function can be used as the *objexpr* argument to the **BACNET** statement, allowing you to specify any property without having to know identifier numbers ahead of time. If you wished to read the **present_value** of Analog Output 1 then you would write:

```
D=OID(AO,1)
BACNET(-1,D,85,-1)
```

Here we have used the **OID** function as an argument in the **BACNET** statement. You can also use expressions in the **OID** function. If, in the example above, instead of Analog Output 1, you were interested in reading the value of the Analog Output who's instance number was stored in A, you would write:

```
D=OID(AO,A)
BACNET(-1,D,85,-1)
```

Expressions can also be used in the *devexpr* and *instexpr* arguments. If you had, for example, 10 *NB*-VAV controllers with device numbers 10 through 19, you could average the measured flow (Flow Control:**present_value**) using the following code.

```
        A=0
        B=10
        C=OID(AI,6)
L1:     A=A+BACNET(9+B,C,85,-1)
        LOOP B,L1
        A=A/10
```

In this program, the value of B is used to increment the device from which the flow is being read and A is the average flow.
(10/5/2012)
Similarly, you could use expressions in arguments to the **BACNET** statement to choose the property being read. If you had, for example, twelve inputs devices connected to your MatrixBBC measuring space temperatures, you could read the current values and calculate an average space temperature using a program similar to the one above or you could simply use the values in the Universal Input Summary Objects. In this case, the current values would be in the **(VD) Current Measured Input 13** through **(VO) Current Measured Input 24** properties. The code to do this would look like this:

```
        A=0
        B=12
L1:     A=A+BACNET(-1,UISUMMARY0,VC+B,-1)
        LOOP B,L1
        A=A/12
```

Here we are using B to increment the property identifier number. The value VC+B is used because we are interested in the values of properties **VD** (identifier number 54852) through **VO** (identifier number 54863). VC has a value of 54851 and we know that B will vary from 12 to 1 as the program executes the **LOOP** statement. The loop will therefore count from 54863, the identifier number for **VO**, down to 54852, the identifier number for **VD**.

3.23.2.2 WRITING VALUES WITH THE BACNET STATEMENT

The syntax used for writing values using the **BACNET** statement is very similar to that used for reading with one additional parameter. When writing values, you must include a priority array level for the write. This value is appended after the *instexpr* argument in the **BACNET** statement. The complete syntax would look like

BACNET(*devexpr,objexpr,propexpr,instexpr,priority*)

where

devexpr is an expression whose value specifies the device object instance of the device containing the property to be written.
objexpr is an expression specifying the object identifier number of the object whose property is to be written.
propexpr is an expression for the identifier number for the chosen property.
instexpr specifies an array index for use in cases when array properties are being written.
priority is an expression for the priority array level for the write command

When writing values using the **BACNET** statement, you may use expressions for any of the arguments in the same way that you could when reading values. For example, if you wanted to turn on all of the digital outputs on, you would write the following:

```
        B=12
L1:     C=OID(BO,B)
        BACNET(-1,C,85,-1,7)=1
        LOOP B,L1
```

. In this example, the **present_value** property for each of the twelve Digital Output objects is set to one. This value is written with a priority of 7.

3.24 TROUBLESHOOTING YOUR SPL PROGRAM

One of the most common call topics that AAM Technical Services receives is regarding SPL programming issues realtive to halted programs. In many cases, halted programs can be troubleshot using some of the features discussed within this section of the manual. This section of the program is provided to re-iterate information in a helpful, summarized fashion.

3.24.1 USING SECTION STATEMENTS

The SECTION statement is a debugging statement that can be placed strategically throughout a program to track its execution process from the program-location property. Figure 3-1 illustrates an example of how SECTION statements can be used throughout a program. Observe that under SECTION 2, the program incorrectly addresses an Analog Input (AI), where the author may have intended to reference an Analog Output (AO).

```
#BBC
;
      PROP  10001,REAL
      PROP  10002,UNSIGNED
      PROP  DESCRIPTION,CHARSTRING=0,32,"SIMULATION PROGRAM"
;
L0:   SWAIT 5
      SECTION 1
      IF [AI1.PRESENT_VALUE] > 72.0 THEN HEAT ELSE L0
;
;
HEAT  SECTION 2
      [AI2.PRESENT_VALUE]=100.0
      SWAIT 1
      SECTION 3
      GOTO L0
```

Figure 3-1 - Example of an SPL Program with Error within Section 2

When the program is loaded and commanded to run, the program will Halt due to this. Figure 3-2 illustrates that this program has halted with the description-of-halt property indicating an error of Property:Write-Access-Denied. While this description may provide a general indication of the program attempting to write to something that is not necessarily writable, it does not provide an exact location or reference. However, the program-location property provides a string message that includes the SECTION number that provides a more general area as to where the error is located at within the program.

✓ object-type (79)	Program (16)
✓ program-state (92)	Halted (4)
✓ program-change (90)	Ready (0)
✓ reason-for-halt (100)	Internal (2)
✓ description-of-halt (29)	Property: Write Access Denied
✓ program-location (91)	Sec 2: 0x0052

Figure 3-2 - Example of NB-Pro Monitoring Halted Program

SECTION statements can be used in defensive manner to help troubleshoot programs after a system has been fully commissioned, and is suggested for crafted line-by-line programs that you may write occasionally.

3.24.2 USING SINGLE-STEP MODE

Single Step Mode is a utility built into each program object that can be used to initially test and/or debug your SPL program. Single Step Mode executes one line of defined program code in a sequential manner. As each line is executed, the program-change property will be set to Halt.

To enable Single-Step Mode on a Program Object, locate property **($1) Enable Single-Step Mode?**. Set this value to *Yes <Single Step Mode> (1)*.

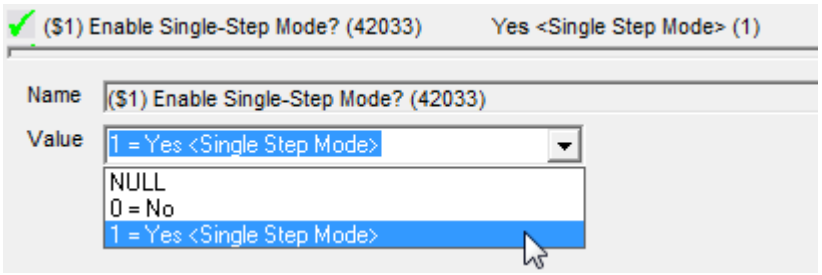

Figure 3-3 - Enabling Single-Step Mode on a Program Object

With Single-Step Mode enabled, you may command a program to Run or Restart in this mode through the program-change property.

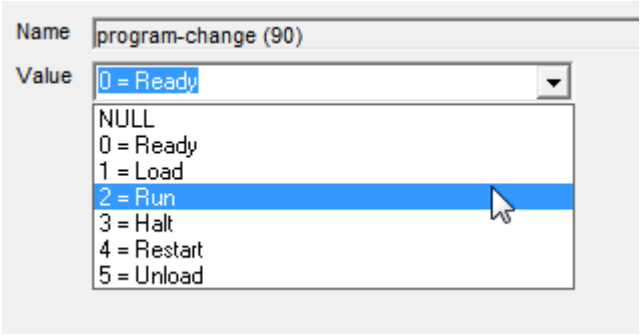

Figure 3-4 - Commanding the program-change Property

As each line is ran, the program will be immediately halted. If the program line has been executed successfully, the **reason-for-halt** property will report **Normal (0)**, and **description-of-halt** will reflect you are in Single Stepping mode as shown in Figure 3-5.

object-identifier (75)	Program [16], Instance 1
object-name (77)	LOGIC\ASCFETCH
object-type (79)	Program (16)
program-state (92)	Halted (4)
program-change (90)	Ready (0)
reason-for-halt (100)	Normal (0)
description-of-halt (29)	Single stepping
program-location (91)	Sec 0: 0x0023

Figure 3-5 - Single Step Mode - Line Executed Successfully

To execute the next line, simply set the program-change property to a value of Run (2). This will command the program to run the next line of logic while remaining in Single-Step Mode.

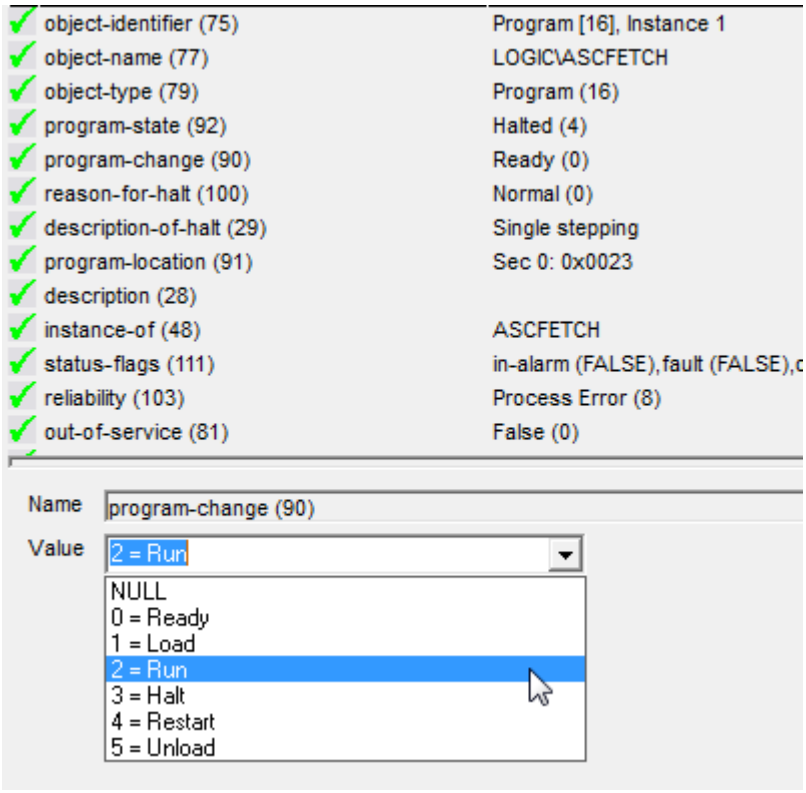

Figure 3-6 - Commanding the Program to Run

After commanding the program, you may eventually find a situation where your program may halt. When it does, information will be provided in a manner as if you were using the SECTION statement.

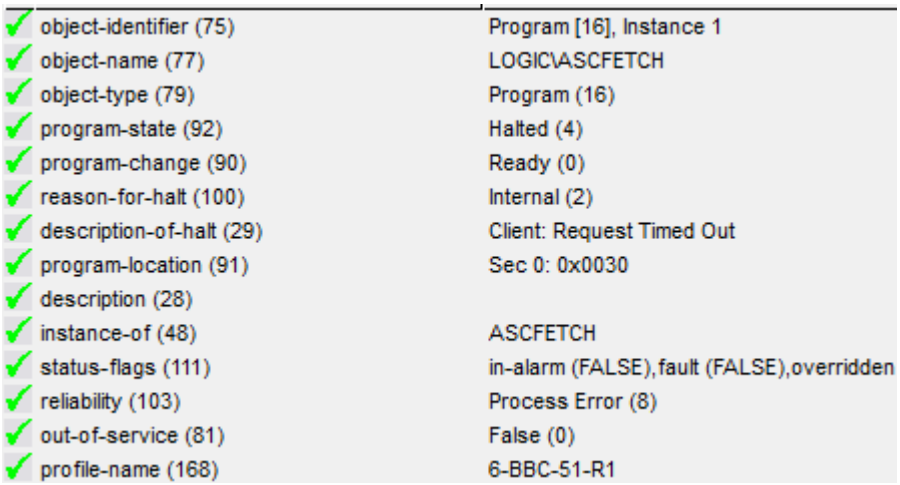

Figure 3-7 - Single Step Mode - Program Halted with Error

3.24.3 REFERENCE THE .LST FILE

The .LST file is a file that is generated by the SPL Compiler when a program successfully compiles for use with a device. The existence of this file is one in which is gives technicians and programmers a comprehensive way to locate which specific program line or command that results in a halted program.

```
SPLW Compiler v1.02                    Fri  05-Oct-2012  10:27:15    Page 001


 **** SPLW Compiler v1.02 (c) American Auto-Matrix, 2003 ****


      Source: C:\USERS\ED\DESKTOP\BBC LOAD UP\PROGRAM 2\ABORT.SPL
      Logic:  C:\USERS\ED\DESKTOP\BBC LOAD UP\PROGRAM 2\ABORT.PLB  (65535 bytes max)
      List:   C:\USERS\ED\DESKTOP\BBC LOAD UP\PROGRAM 2\ABORT.LST

        Off   PLB Header Bytes          Definition
        ----  ------------------------  ----------
        0000  ?? ??                     (Next PLB)
        0002  4C 4F 47 49 43 5C 41 42   LOGIC\ABORT
              4F 52 54 00 00 00 00 00
              00 00
        0014  ?? ??                     (Nbr of Users)
        0016  00 00                     (Ptr to Attrs)
        0018  9A 00                     (Size of PLB)
        001A  07 00                     (Type: 1=SAGE v1.xx, 2=SAGE v2.00,
                                               3=SOLO/DX, 4=SOLO/GX, 5=GPC)
        001C  FF FF                     (One Second Offset SOLO/GX & GPC only)
        001E  5C 00                     (Ptr to Properties GPC only)

SPLW Compiler v1.02                      Fri  05-Oct-2012  10:27:15    Page 002


Line    Off   Pcodes                    Statement
-----   ----  ------------------------  ---------
00001 (0x0001)                                  #BBC
00002 (0x0002)                                  ;
00003 (0x0003)                                       PROP  10001,REAL
00004 (0x0004)                                       PROP  10002,UNSIGNED
00005 (0x0005)                                       PROP  DESCRIPTION,CHARSTRING=0,
32,"SIMULATION PROGRAM"
00006 (0x0006)                                  ;
00007 (0x0007)   0020  03 09 FE 05 00           L0:  SWAIT 5
00008 (0x0008)   0025  02 01 00                      SECTION 1
00009 (0x0009)   0028  0A 00 00 20 00 81 01 00       IF [AI1.PRESENT_VALUE] > 72.0
THEN HEAT ELSE L0
                 0030  00 00 55 00 00 00 09 0B
                 0038  E0 00 00 90 42 00
00010 (0x000a)                                  ;
00011 (0x000b)                                  ;
00012 (0x000c)   003E  02 02 00                 HEAT SECTION 2
00013 (0x000d)   0041  81 02 00 00 00 55 00 00       [AI2.PRESENT_VALUE]=100.0
                 0049  00 00 0B E0 00 00 C8 42
                 0051  00
00014 (0x000e)   0052  03 01 00                      SWAIT 1
00015 (0x000f)   0055  02 03 00                      SECTION 3
00016 (0x0010)   0058  0B 20 00                      GOTO L0
00017 (0x0011)
00018 (0x0012)                                  ;
00018 (0x0012)   005B  00                            STOP
SPLW Compiler v1.02                      Fri  05-Oct-2012  10:27:15    Page 003
```

The .LST file can be opened with any standard text editor. Upon first opening the file, the information may appear to be cryptic to users. However, this section of the manual is provided to assist you with understanding the file format and how you can leverage it for advanced troubleshooting.

3.24.3.1 REFERENCING .LST COLUMN DATA

Within the .LST File, there are four columns of information that provide detailed information regarding the program that can essentially assist troubleshooting your compiled SPL program. These four columns are described in further detail in the table below.

Table 3-6 The Four Columns for Troubleshooting SPL via .LST File Method

Column	Notes
Line	The Line section contains the SPL Source Code Document Line number. This corresponds to the line number of the SPL source that is referenced.
Off	The Off section specifies memory offset which are referenced internally by the product's Program Executor (PEX).
Pcode	The Pcode section contains a detailed breakdown of the single pseudocode (pcode) memory offset for a specific line of SPL programming. This pcode is referenced by a program as it is executed and reported through the **program-location** property. This column
Statement	The Statement section provides the raw typed SPL source code developed by the technician.

```
Line    Off   Pcodes                    Statement
-----   ----  ------------------------  ---------
00001 (0x0001)                                    #BBC
00002 (0x0002)                                    ;
00003 (0x0003)                                         PROP  10001,REAL
00004 (0x0004)                                         PROP  10002,UNSIGNED
00005 (0x0005)                                         PROP  DESCRIPTION,CHARSTRING=0,
32,"SIMULATION PROGRAM"
00006 (0x0006)                                    ;
00007 (0x0007)   0020  03 09 FE 05 00             L0:  SWAIT 5
00008 (0x0008)   0025  02 01 00                        SECTION 1
00009 (0x0009)   0028  0A 00 00 20 00 81 01 00         IF [AI1.PRESENT_VALUE] > 72.0
THEN HEAT ELSE L0
                 0030  00 00 55 00 00 00 09 0B
                 0038  E0 00 00 90 42 00
00010 (0x000a)                                    ;
00011 (0x000b)                                    ;
00012 (0x000c)   003E  02 02 00                   HEAT SECTION 2
00013 (0x000d)   0041  81 02 00 00 00 55 00 00         [AI2.PRESENT_VALUE]=100.0
                 0049  00 00 0B E0 00 00 C8 42
                 0051  00
00014 (0x000e)   0052  03 01 00                        SWAIT 1
00015 (0x000f)   0055  02 03 00                        SECTION 3
00016 (0x0010)   0058  0B 20 00                        GOTO L0
00017 (0x0011)
00018 (0x0012)                                    ;
00018 (0x0012)   005B  00                              STOP
SPLW Compiler v1.02                    Fri  05-Oct-2012  10:27:15    Page 003
```

Figure 3-8 - .LST FIle Displaying the Line, Off, Pcodes, and Statement Columns

In the event that a deployed program halts or aborts, the location of which the program halted is advertised in the **program-location** property. This string will define a hexadecimal offset. This offset directly corresponds with Pcode column information. In the sample shown below, this program halted at offset 0x0028. We will examine the .LST file and locate this specific line and command.

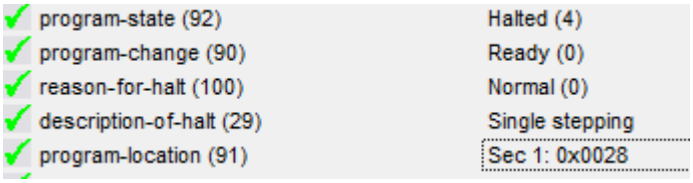

Figure 3-9 - Example of Halted SPL Program with Stated Location

3.24.3.2 PARSING THE .LST FILE

The following shows an example of how the .LST file is parsed for information. Notice that under the Pcodes column, the hexadecimal location is provided. As shown below in the red box, the line of code begins with hex offset 0028, which indicates that the program halt is being caused by this statement (which is an attempt to write to an Analog Input that is likely out of service).

Pcodes are counted in hex from left to right start after the offset location, which could be 0020, 0025, or 2280. The first offset is the listed memory Pcode, then is incremented up from there for each statement on a line.

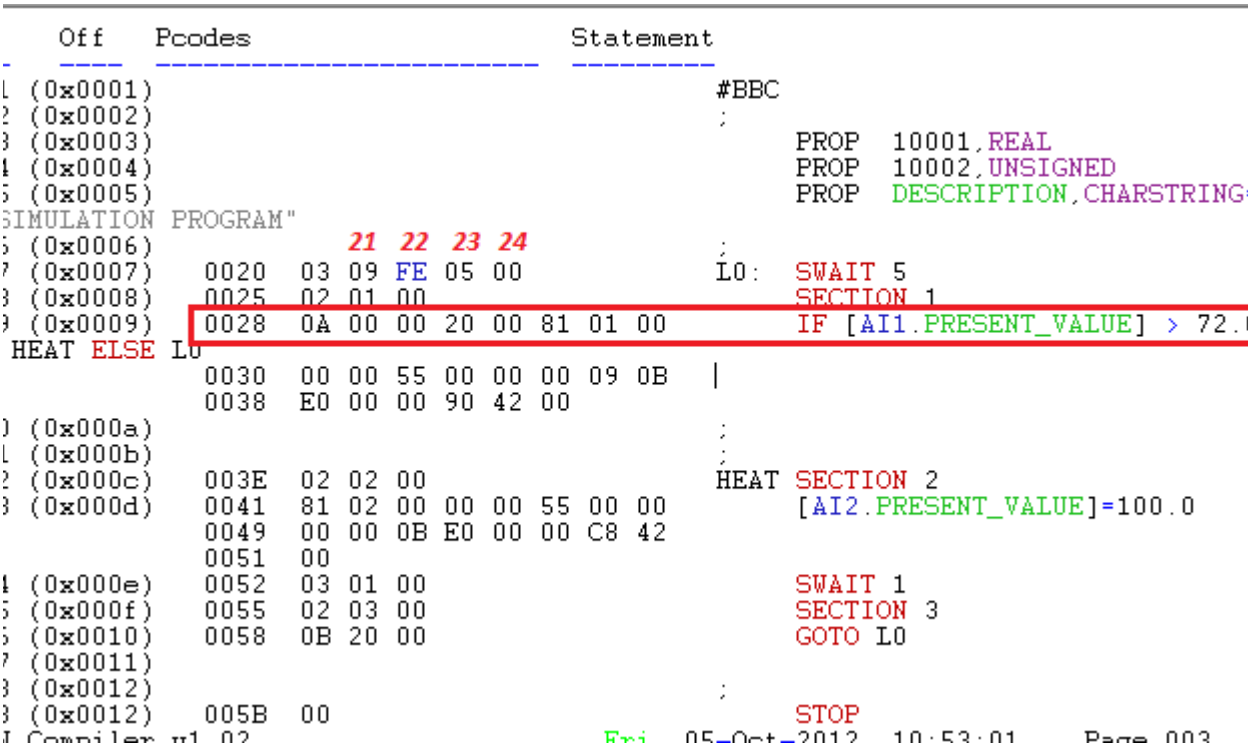

Figure 3-10 - Locating the Pcode in the .LST File

SECTION 4: SCHEDULING

This section provides general information regarding the use of BACnet Schedule and BACnet Calendar objects, and how they may be used to setup general schedule occupancy, as well as advanced, programmatic scheduling of control values within the MatrixBBC.

IN THIS SECTION

4.1 SCHEDULING OVERVIEW

The MatrixBBC supports BACnet Schedule and Calendar objects to permit programmatic scheduling of values within the controller, as well as manipulation of values in other devices on the BACnet network. Using these objects together, it is also possible to perform complex overrides of normal daily schedules based on events defined in a Calendar object, or other system actions based on object statuses. The MatrixBBC supports a maximum of 32 Schedule objects.

4.1.1 ABOUT SCHEDULE OBJECTS

Schedule objects are setup to define a periodic schedule that may recur during a range of dates, with optional exceptions at arbitrary times on arbitrary dates. The Schedule object also serves as a binding between these scheduled times and the writing of specified values to specific properties of specific objects at those times.

Schedules include two different scheduling methods: weekly scheduling and exception scheduling. Both types of days can specify scheduling events for either the full day or portions of a day, and a priority mechanism defines which scheduled event is in control at any given time.

The current state of the Schedule object is represented by its present-value property, which is normally calculated using configured time/value pairs entered into the weekly-schedule and/or exception-schedule properties. Schedules also include a default value (known as schedule-default) for use with no schedules are in effect.

4.1.2 ABOUT CALENDAR OBJECTS

Calendar objects are used to define days, dates, date ranges, and other periods that may indicate a special event. Calendar objects are commonly used in conjunction with a Schedule object's exception-schedule property to configure a higher priority schedule mode that deviates from the normal schedule calculations controlled by the weekly-schedule property of a Schedule object. The MatrixBBC supports a maximum of 32 Calendar objects.

4.1.3 CREATING SCHEDULES IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Schedules objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC supports a maximum of 32 Schedule objects.

To create a Schedule object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MD) Max Schedule Objects** property. By default, this value is set to 0, indicating no Schedule objects exist.
3. Write the number of total Schedule objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Schedules, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Schedule objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

4.1.4 CREATING CALENDARS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Calendar objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC supports a maximum of 32 Calendar objects.

To create a Calendar object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MC) Max Calendar Objects** property. By default, this value is set to 0, indicating no Calendars objects exist.
3. Write the number of total Calendar objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Calendars, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Calendar objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

4.2 SCHEDULE OBJECT CONFIGURATION

The following sections provide details on how to configure a Schedule to change values in the MatrixBBC.

4.2.1 DETERMINE YOUR SCHEDULE APPLICATION

BACnet Schedule objects are intended to provide programmability and flexibility for your area control routines. Schedules created within the MatrixBBC can be used to adjust software parameters within the MatrixBBC (e.g. control loops, data storage objects, etc.). In addition to internal control, external control over remote BACnet objects can be performed with Schedules.

Within the MatrixBBC, Control Loops (Analog PID Control, Pulsed Pair PID Control, and Thermostatic Control) can be controlled using AAM's classic four-mode scheduling system. Each schedule mode can have a specific setpoint assigned that allows the control loop to reference the schedule and operate on a specific setpoint parameter when the Schedule's **present-value** is equal to one of the four scheduled modes. Application Specific Controllers manufactured by American Auto-Matrix also support four-mode scheduling (Occupied, Unoccupied, Warmup, Night Setback). These schedule modes typically refer back to specific actions that may occur in the application when the schedule enters into one of the four specified states.

The mode assignment is as follows:
0 = Unoccupied
1 = Warm Up
2 = Occupied
3 = Night Setback

To provide support for this scheduling mechanism, a Schedule object can be configured for an Unsigned data type and be configured to support four-mode scheduling.

4.2.2 CONFIGURE THE SCHEDULE DATATYPE

Before you can instruct a Schedule object exactly what it will control, you must first determine the datatype you will schedule with. Determining the datatype will depend on exactly what you wish to control based on a schedule. The Schedule object can be configured to schedule with one specific datatype. If you need to schedule with multiple data types, you must use multiple schedules.

Schedule objects can be configured to schedule with the following data types:
. Boolean
. Unsigned
. Integer
. Real
. Enumerated
. Date
. Time
. Object ID

Data types and examples of commonly scheduled properties are provided in the following table below.

Table 4-1: Schedule Datatype Example Usage

Datatype	Example Use
Boolean	. Any object's **out-of-service** property . The log-enable property of a Trend Log object . Any standard object's **(EA) Enable Alarming** property.
Unsigned	. Adjust the time-delay of an alarm for an object configured for alarm/event support. . Change the function of a Data Manipulation object (e.g. Function of a Math object). . Used to utilize the four-mode schedule application for control loops within the MatrixBBC.
Integer	Modify a signed integer property in an SPL program or in an external device through use of a Netmap object.
Real	. Command the present-value of an Analog Output or Analog Value . Adjust a set point of a PID Control Loop, Thermostatic Control Loop,etc.
Enumerated	. Command the present-value of a Binary Output, or a Binary Value object . Change the Notification Class assignment for a specific object enabled for alarming.
Date	Modify a date property in an external device through the use of a Netmap object.
Time	Modify a time property in an external device through the use of a Netmap object.
Object ID	Modify the object-ID portion of an object-property reference of a logic object within the MatrixBBC (such as a Math object, a PID Control Loop, Thermostatic Control loop, etc.)

To configure the datatype of a Schedule object:

1. Select **schedule-default** and set the property type to the data type you wish to configure the schedule for and click *Update Value*.

-or-

2. Select **(DT) Schedule's Default Data Type**, and select a datatype from the list and click *Update Value*.

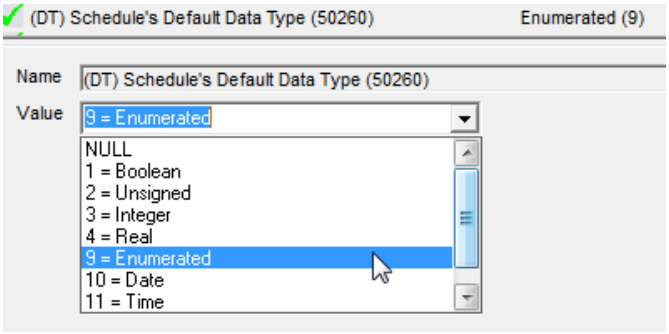

Figure 4-1 - Schedule Default Datatype Configuration

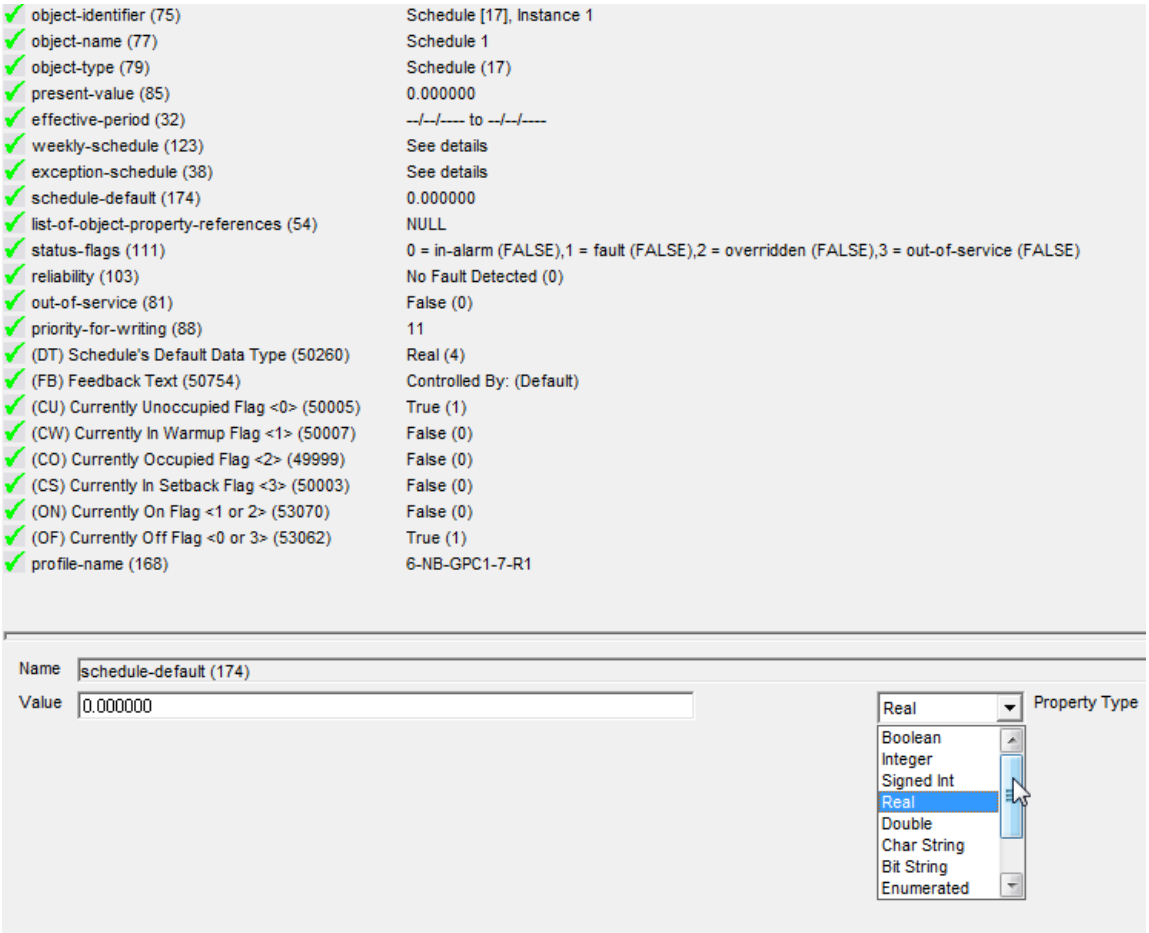

Figure 4-2 - Schedule Datatype Configuration Using the Schedule-Default Property

4.2.3 CONFIGURE THE EFFECTIVE PERIOD

By default, a Schedule is configured to be effective (actively calculating schedule data) at all times. In some situations, there may be cases where you may want a programmed Schedule to only be affective during specific date ranges, day ranges, or other special periods where a date may need to be wildcard for customization. This is achieved through configuring the **effective-period** property.

The effective-period defines a range which can be configure for:

. Month/Date/Year - a specific month, date, and year (e.g. January 25th, 2010 to January 27th, 2010).
. Month/Date - a specific month and date on any year (e.g. December 22th to December 26th).
. Day of Week - a specific day of the week on any month and any year (e.g. Monday through Thursday).
. Month - a range of months for any year (e.g. January through March)
. Month/Day of Week - a specific day of week on a specific month of any year (e.g. Monday in November through Wednesday in November).
. Any - any range not defined above. This is a custom, wildcard selection.

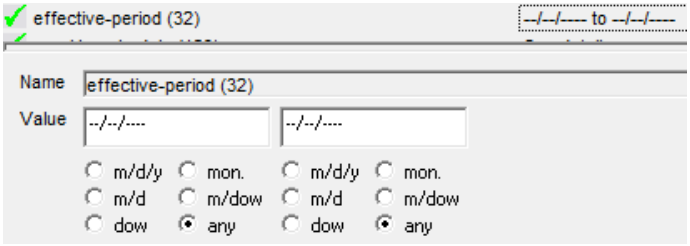

Figure 4-3 - Configuring the Effective-Period

NOTE

When the effective-period is set for it's default value of any to any (---/--/---- to --/--/----), the Schedule's configuration will remain in effect indefinitely.

When the MatrixBBC's **local-date** (see Device object) is within the confines of the defined **effective-period**, the Schedule object will operate and control. No calculations will occur when the **local-date** is outside of the defined range.

4.2.4 CONFIGURE THE LIST OF OBJECT-PROPERTY REFERENCES

The **list-of-object-property-references** defines what object-properties the Schedule object will write values to when a time,value pair entry is processed. For example, if you wish for a group of Binary Outputs to be active at a specific time, you must reference each Binary Output object's present-value you wish to control within this property.

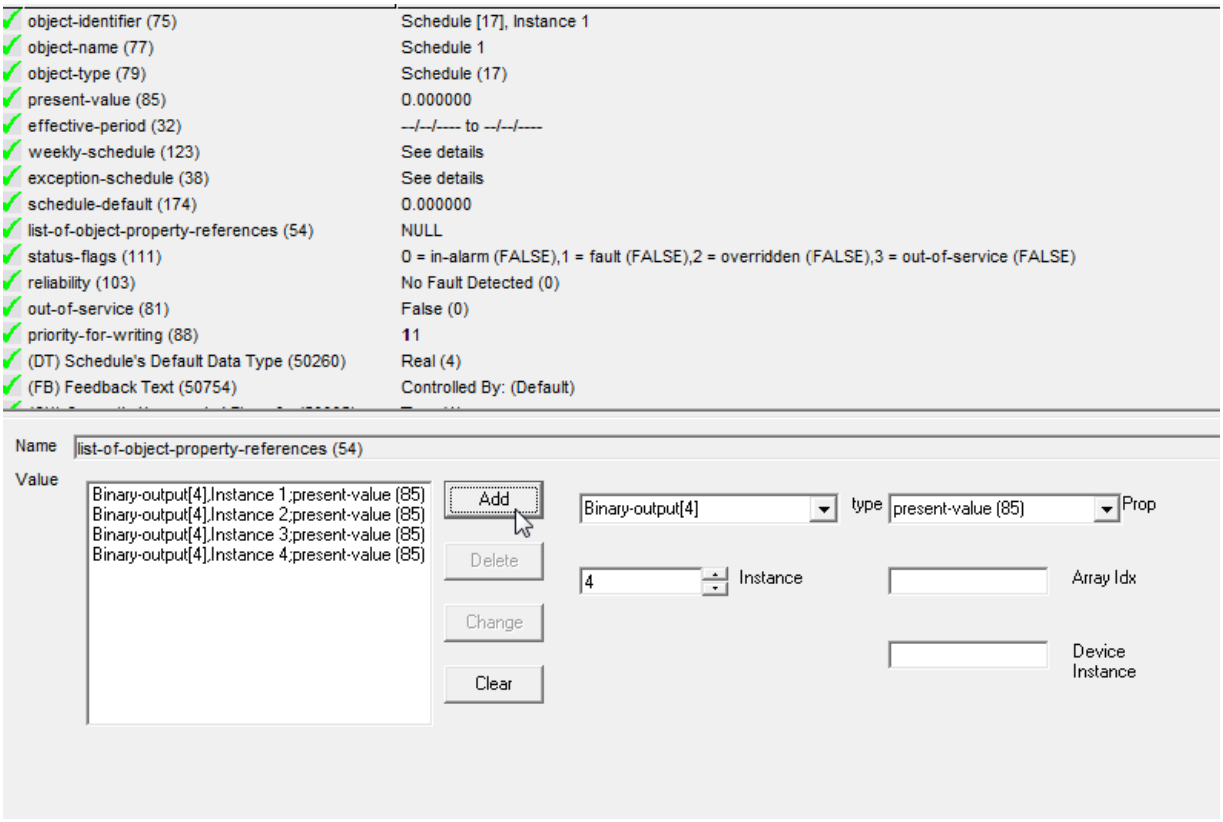

Figure 4-4 Configuring the list-of-object-property-references

Schedules can accept up to 50 object-property-references. A reference can be any object-property that exists within the controller. For objects on remote devices, you must enter the Device Instance of the BACnet device you wish to write to. Should you need to control additional object-property-references using the same Schedule, it is recommended that you create another Schedule object, configure it's list-of-object-property-references, and use a Remap object to write the **present-value** of the first Schedule to the **schedule-default** property of the new schedule.

NOTE

If you experience issues with the MatrixBBC writing to objects on other devices, check to ensure you have entered the correct Device Instance into this property. You can also check the device-address-binding property of the Device Object of the MatrixBBC to ensure that the device was located. If the device was not located, you can manually configure an entry into the binding list.

4.2.5 CONFIGURE THE PRIORITY FOR WRITING

The priority-for-writing property defines which Command Prioritization level the Schedule object will write with in the event that its list-of-object-property-references is programmed to control the present-value of

Analog Output, Analog Value, Binary Output or Binary Value objects. This value is assignable to a value between 1 and 16.

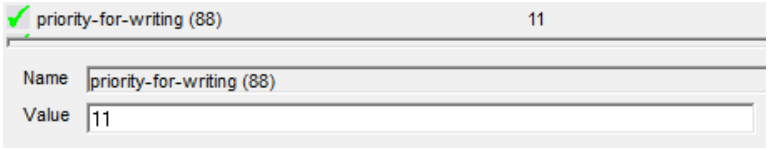

Figure 4-5 - Configuring Priority-For-Writing

NOTE

When writing to objects in third-party devices, refer to the vendor's documentation on priority levels that can be used for writing. Some third-party devices may limit or restrict the use of specific priority levels for use or application.

4.2.6 CONFIGURE THE WEEKLY-SCHEDULE

The weekly-schedule property contains time/value pairs which tell the Schedule what value to write to the object listed in **list-of-object-property-references** at the time defined in the weekly-schedule. The weekly-schedule contains a list for each day of the week (Monday through Sunday), where each day of the week supports up to 9 time/value pair entries.

To configure a weekly-schedule, perform the following steps:

1. Select the weekly-schedule property. From the editor, select the day of the week you wish to define.
2. Using the editors, enter a time and associated value. Click *Add* to add the definition.
3. If you wish for the objects to follow the value listed in schedule-default, enter a time and select the Resume Default button or type NULL. This will place a NULL next to the time in the list.
4. Once you have successfully defined the list for a day, click *Update Value*.
5. Repeat steps 2 and 4 for all other days of the week.

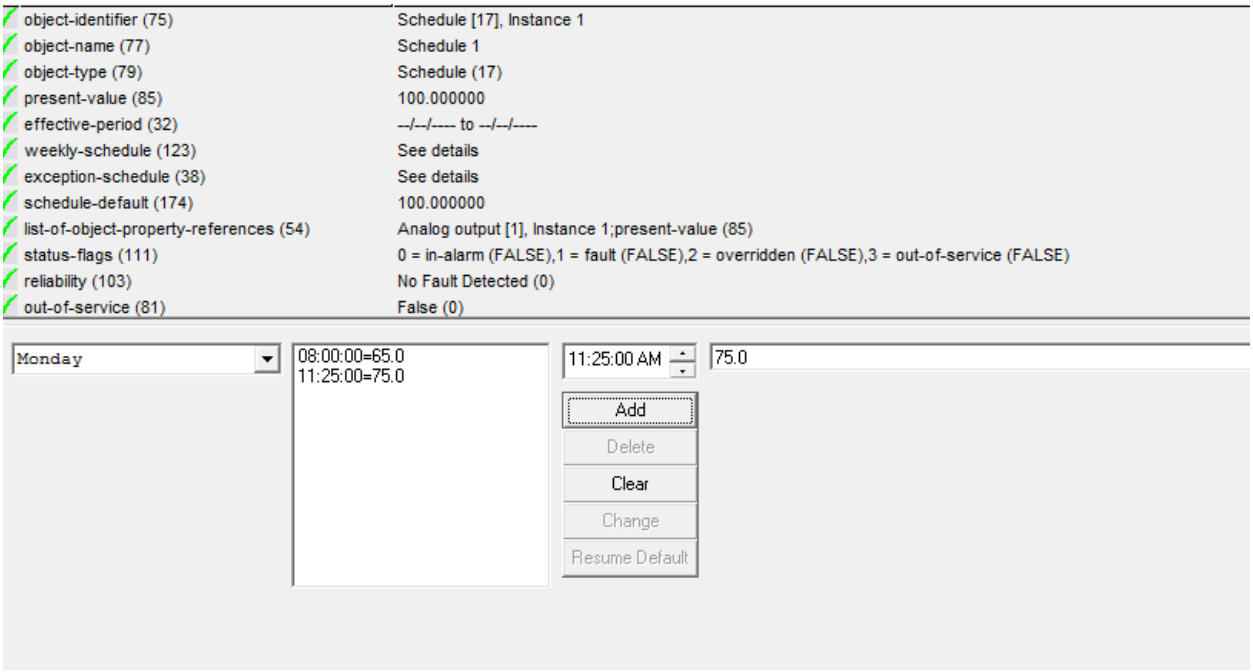

Figure 4-6 Configuring the weekly-schedule

4.2.7 CONFIGURING THE EXCEPTION SCHEDULE

The exception-schedule is used to configure a higher priority schedule, thus overriding the functionality defined in the weekly-schedule for special situations, such as one-time events (e.g. holidays, snow days, last minute meetings, etc.). These exceptions are listed Scheduled events entered into the exception-schedule are intended for singular events, or even recurring situations where the schedule must be overridden. Each Schedule object can contain up to five (5) listed exceptions.

Exceptions can be based on one of four reference methods:

. Date - a single defined date with our without wildcard.
. Date Range - a range of dates with or without wildcard.
. Week N Day - Entry based on week number, day and month - with or without wildcard
. Calendar Reference - references a configured Calendar object within the MatrixBBC.

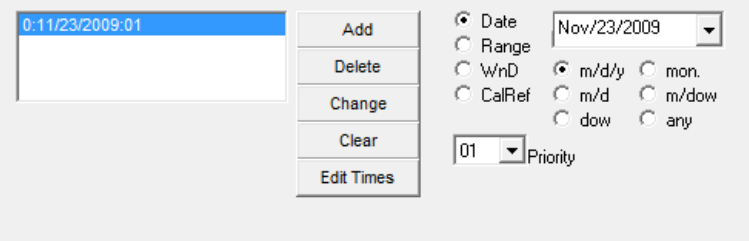

Figure 4-7 Options for the exception-schedule reference

For each reference, a priority is also available with prioritizes the importance of the exception list entries. Once a reference method has been selected, simply enter your time and associated value.

Figure 4-8 Entering Time,Value Pairs for Exception Schedules

CAUTION

In order to perform exception-scheduling with a Calendar reference, at least one (1) Calendar object must be created. If you attempt to configure an exception-schedule to reference a Calendar when one does not exist, writes will not be accepted.

4.3 CALENDAR OBJECT CONFIGURATION

Calendar objects are simple and straight-forward to setup and configure. Each Calendar object contains a **datelist** property, which defines a list of dates, date ranges, or week-n-day entries. A maximum of 25 entries (consisting of any entry type) can be programmed into the datelist of each Calendar object.

When the **local-date** of the MatrixBBC's Device object coincides with an entry programmed into the **datelist**, the Calendar's **present-value** will indicate that the Calendar is active. This permits users to program special events (such as recurring holidays, functions, one-time events, etc..)

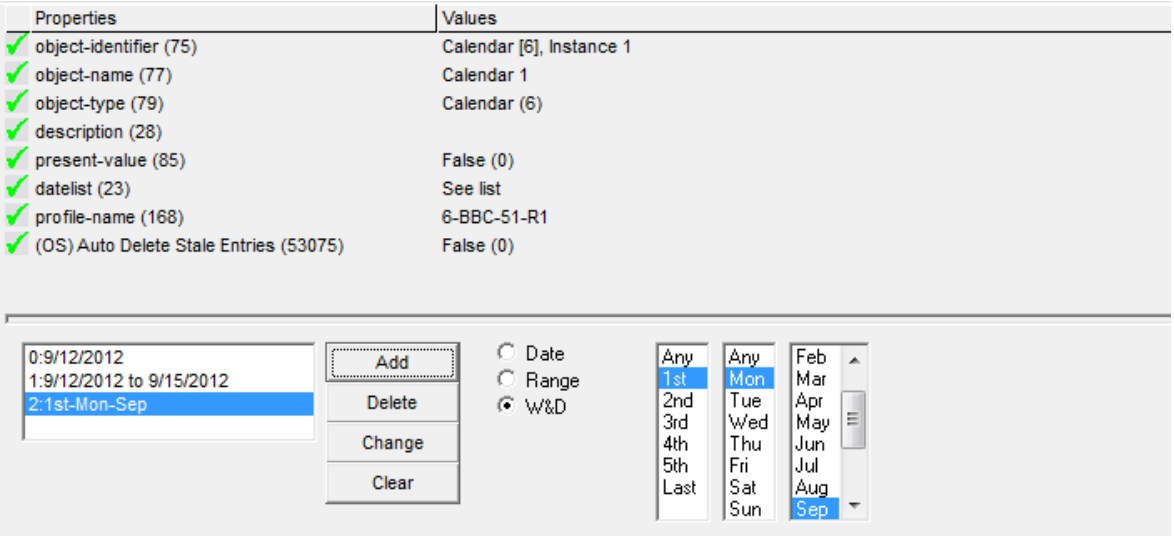

Figure 4-9 Calendar Configuration

Calendar objects are mainly used in conjunction with exception-schedule configurations of a Schedule object. However, you are free to use any additional logic available at your disposal within the MatrixBBC to perform other control sequences based on the **present-value** of the Calendar object.

4.3.1 AUTO-DELETING STALE CALENDAR ENTRIES

The **(OS) Auto Delete Stale Entries** property can be used to perform an automated "clean-up" of special events from the datelist that are Date based. For example, an entry could be made into the datelist of a Calendar object signify a special one-time event. When the event expires, traditional BACnet devices may keep the entry of this event until a user manually removes the event from the datelist. When **(OS) Auto Delete Stale Entries** is set to True (1), the MatrixBBC will examine the datelist upon every day at midnight, or upon receiving a time-synchronization from a higher level server such as AspectFT. If the local-date exceeds a Date-based entry made that does not contain wild-card placeholders, the entry will be removed.

SECTION 5: ALARM ROUTING

This section provides general information regarding the setup and configuration of alarm routing using BACnet Notification Class objects. The MatrixBBC will support a maximum of 10 Notification Class objects.

IN THIS SECTION

5.1 NOTIFICATION CLASS OVERVIEW

Notification Classes are used in BACnet to organize and define the distribution of alarm and event notifications from objects that support notifications. Within the MatrixBBC, several objects support notifications, allowing end-users to know when certain situations occur within the system (e.g. a space temperature may be too high for normal comfort). Notification Classes are useful for event-initiating objects that have identical needs of how their notifications should be handled, what the destinations are for {where notifications should be sent, and how they should be acknowledged}.

In many cases, Notification Classes are configured to allow the MatrixBBC to send alarm and event notifications to an operator workstation or centralized front-end/web-server.

The MatrixBBC can support up to 10 Notification Class objects using dynamic creation as needed. Each of these Notification Class objects support up to 5 configurable destinations - each containing a valid day/ time schedule, unique process identification number, and other features.

5.1.1 CREATING NOTIFICATION CLASSES IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Notification Class objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC supports up to a maximum of 10 Notification Class objects.

To create a Notification Class object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MO) Max Notification Class Objects** property.
3. Write the number of total Notification Class objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Notification Class objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Notification Class objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

5.1.2 CONFIGURING THE PRIORITY

The priority property defines a numeric level used to define the criticality of an event or alarm notification. The property provides criticality levels for the three types of transitions supported by BACnet, including:

. To-OffNormal - occurs when the object exits outside the configured limit thresholds.
. To-Normal - occurs when the object enters into a normal state within configured limit thresholds.
. To-Fault - occurs when the object's health enters into an unhealthy state (e.g. temperature sensor shorts and provides unreliable values).

Each priority can be assigned a value ranging from 1 (most critical) to 255 (least critical).

Figure 5-1 - Example Configuration of priority

5.1.3 CONFIGURING ACK-REQUIRED

The ack-required property defines whether or not the MatrixBBC requires acknowledgments back from an operator workstation or BACnet device once a notification has been received from the MatrixBBC. This property provides enable/disable options for each limit threshold of BACnet (To-OffNormal, To-Normal, To-Fault). Enabling a limit threshold requires acknowledgment.

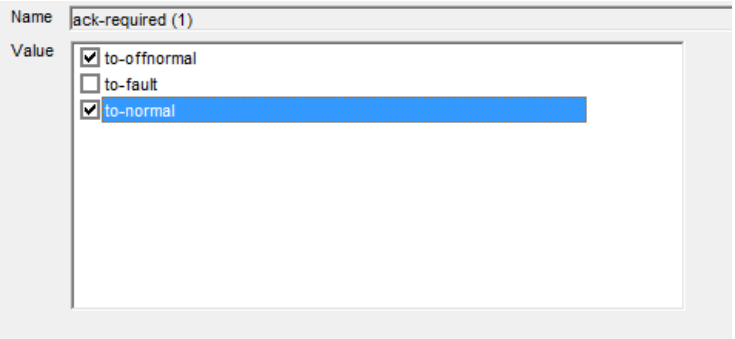

Figure 5-2 - Example Configuration of ack-required

5.1.4 CONFIGURING THE RECIPIENT LIST

The recipient-list property specifies addressed destinations where notifications are sent. Each Notification Class object can have up to five (5) recipients defined. Each recipient contains the following:

. Simple Schedule - used to define a valid period of time and day as to when notifications can be sent.
. Process ID - a unique ID allowing operator workstations or other devices to process alarms in a specific manner. Valid ranges are 0-65535.
. Address - defines where alarms are sent to. The address can be defined in the form of a device instance, or actual address with network number (0 = local, 1-65534 = specific network, 65535 = broadcast) and port.
. Notifications - defines whether or not to issue confirmed notifications, and the transitions that should be sent by the transitions.

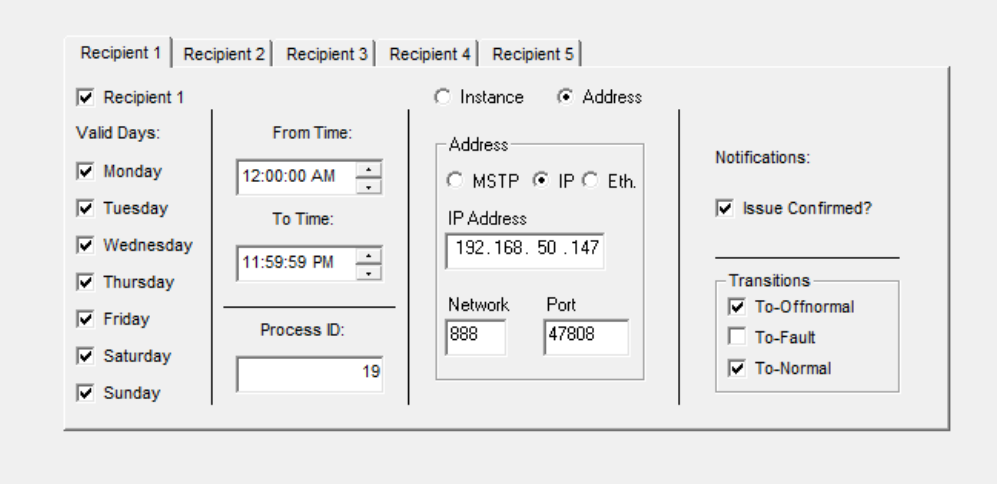

Figure 5-3 - Example Configuration of recipient-list

NOTE

If you have programmed a destination previously into the Device object's device-address-binding table, you can simply define the device instance and the MatrixBBC should automatically fill in the remaining network inform ai ton based on the table entry

SECTION 6: DATA STORAGE

This section describes the usage of Analog Value, Binary Value, and Trend Log objects within the MatrixBBC. The MatrixBBC will support up to 1000 Analog Values, 1000 Binary Values, and 256 Trend Log objects.

IN THIS SECTION

6.1 DATA STORAGE OVERVIEW

The Data Storage area of the MatrixBBC provides general purpose software value objects that can be used to store miscellaneous data such as set points and command toggles. More so, Data Storage objects can also be used to provide alarm and event notification services to software values in the controller that may not directly be addressed as a standard object. This section explains the use of Analog Value and Binary Value objects and how they can be configured to provide alarm/event notifications of software values within the controller.

Data that is stored within Analog and Binary Value objects is persistent - meaning that reboots and other power cycles will not clear this data. This is helpful for situations where you may be tracking critical data stored into Analog and Binary Values by Remaps, SPL Programming, or other sources.

6.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

Data Storage objects exist in the controller to reduce the amount of line-by-line SPL programming that one would need to write to carry out advanced control functions. To enhance your programming experience, the following are a few helpful concepts and techniques to keep in mind when using these objects.

6.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The MatrixBBC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used Data Storage object. This allows you not only to keep track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

6.1.1.2 ENABLE ALARMING WHEN NEEDED

All Data Storage objects optionally support alarming. When alarming is disabled **(EA) Enable Alarming** = Disabled (0), fewer properties will be displayed in NB-Pro, allowing the objects to be interpreted easier during programming.

6.2 ANALOG VALUE OBJECTS

Analog Values objects within the MatrixBBC are general purpose and can be used for any basic need within your application. Information stored within this object is represented in a floating point manner. Through use of Local Remap objects, other numeric data types such as Unsigned and Signed Integers can be transferred and exposed if desired. Analog Values support many useful features of the BACnet standard, including:

. Configurable object name
. Command Prioritization (Priority Array)
. Alarm and Event Notification services

As an additional degree of flexibility, any value commanded to the Analog Value will retain its last known value in the event of a restart due to power loss, cycle, etc.

6.2.1 CREATING ANALOG VALUES IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Analog Value objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC will support a maximum of 1000 Analog Value objects.

To create an Analog Value object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M1) Max Analog Value Objects** property. By default, this value is set to 0, indicating no Analog Value objects exist.
3. Write the number of total Analog Value objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Analog Value objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Analog Value objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

6.2.2 CONFIGURING ALARM/EVENT NOTIFICATIONS

Analog Values can be configured to support alarm/event notifications. To enable alarming for an Analog Value, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Analog Value objects can be configured to trigger one of the two following conditions:

. Low Limit - occurs when **present-value** is less than the value specified in **low-limit**.
. High Limit - occurs when **present-value** is greater than the value specified in **high-limit**.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.
3. Configure **limit-enable** to enable low-limit or high-limit alarming. This is accomplished by placing a check into each associated limit type.
4. Configure event-enable to have the object send alarms for how alarms transition. For example, if you wish to have the MatrixBBC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
5. Configure your **high-limit** and **low-limit** properties accordingly.

6. Configure the **time-delay** and **deadband** properties. The time-delay property defines a threshold of time (in seconds) where the **present-value** must exceed one of the limit properties in order for an alarm/event condition to be considered. The **deadband** property defines an offset from **low-limit** or **high-limit** that must be met in order for an alarm/event condition to be considered. For example, if **high-limit** = 75.0, **deadband** = 2.0, and **time-delay** = 5, the **present-value** must exceed 77.0 for at least 5 seconds before an alarm/event condition is considered.

6.2.3 ANALOG VALUE APPLICATION EXAMPLES

The following are some common examples of how an Analog Value may be used within the MatrixBBC for specific applications.

6.2.3.1 RUNTIME LIMIT ALARMING

A site may require the ability to receive runtime limit alarms for specific equipment (such as pumps, fans, generators, etc.). While the runtime hours is a property available from most inputs and outputs, they are not directly configurable for alarms directly from the input or output.

To generate runtime alarms, you must use an Analog Value object. To setup a runtime alarm for an object (in this example, we will use Binary Input 1's run hours, perform the following steps:

1. Configure a Local Remap to map the **(RH) Run Hours** property of Binary Input 1 to Analog Value 1; present-value. Your Local Remap configuration should look similar to the illustration below.

object-identifier (75)	Proprietary [304], Instance 1
object-name (77)	BI1 Run Hours to AV1
object-type (79)	304
profile-name (168)	6-NB-GPC1-7-R1
present-value (85)	19
(O1) Input Object ID (53041)	Binary input [3], Instance 1
(P1) Input Property (53297)	53832
(O2) Output Object ID (53042)	Analog value [2], Instance 1
(P2) Output Property (53298)	85
(Q2) Output Priority (53554)	11
(TO) Trigger Object (54351)	Analog input [0], Instance 0
(TP) Trigger Property (54352)	0
(TB) Trigger Biasing (54338)	Trigger Active When Non-Zero (0)
(RS) Remap Status (53843)	Bad Read (FALSE),Bad Write (FALSE),Bad Trigger Read (FALSE)
(FB) Feedback Text (50754)	Remap Object Active & Working (And Coercing Datatype)
(RM) Remap Mode (53837)	Continuous (1)

Figure 6-1 Remapping Run Hours to Analog Value 1

2. Analog Value 1 will now contain the **(RH) Run Hours** property value from Binary Input 1. Enable alarming on the Analog Value via **(EA) Enable Alarming** = 1, and configure your alarm parameters accordingly.

object-identifier (75)	Analog value [2], Instance 1
object-name (77)	Pump RunHours
object-type (79)	Analog value (2)
present-value (85)	19.000000
status-flags (111)	0 = in-alarm (FALSE),1 = fault (FALSE),2 = overridden (FALSE),3 = out-of-service (FA
event-state (36)	normal (0)
out-of-service (81)	False (0)
units (117)	no-units (95)
priority-array (87)	NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,19.000000,NULL,NULL,NU
relinquish-default (104)	0.000000
time-delay (113)	0
notification-class (17)	0
high-limit (45)	100.000000
low-limit (59)	0.000000
deadband (25)	0.000000
limit-enable (52)	0 = lowLimitEnable (FALSE),1 = highLimitEnable (TRUE)
event-enable (35)	to-offnormal (TRUE),to-fault (FALSE),to-normal (FALSE)
acked-transitions (0)	to-offnormal (TRUE),to-fault (TRUE),to-normal (TRUE)
notify-type (72)	Alarm (0)
event-time-stamps (130)	--/--/--------:--,--/--/--------:--,--/--/--------:--
profile-name (168)	6-NB-GPC1-7-R1
(EA) Enable Alarming (50497)	True (1)

Figure 6-2 Analog Value Configured for Run Hour Alarming

6.3 BINARY VALUE OBJECTS

Binary Values objects within the MatrixBBC are general purpose and can be used for any basic need within your application. Information stored within this object is represented as an Inactive/Active manner. Through use of Local Remap objects, other numeric data types such as Unsigned and Signed Integers, and Booleans can be transferred and exposed if desired. Binary Values support many useful features of the BACnet standard, including:

. Configurable object name
. Command Prioritization (Priority Array)
. Alarm and Event Notification services

As an additional degree of flexibility, any value commanded to the Binary Value will retain its last known value in the event of a restart due to power loss, cycle, etc.

6.3.1 CREATING BINARY VALUES IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Binary Value objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC will support a maximum of 1000 Binary Value objects.

To create an Binary Value object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M2) Max Binary Value Objects** property. By default, this value is set to 0, indicating no Binary Value objects exist.
3. Write the number of total Binary Value objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Binary Value objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Binary Value objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

6.3.2 CONFIGURING ALARM/EVENT NOTIFICATIONS

Binary Values can be configured to support alarm/event notifications. To enable alarming for an Binary Value, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Binary Value object alarms/events are triggered based on the setting of the **alarm-value** property.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.
3. Configure **event-enable** to have the object send alarms for how alarms transition. For example, if you wish to have the MatrixBBC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
4. Configure the **time-delay** property. The time-delay property defines a threshold of time (in seconds) where the **present-value** must exceed one of the limit properties in order for an alarm/event condition to be considered.

6.4 TREND LOG OBJECTS

Trend Log objects within the MatrixBBC are BACnet compliant objects used to trend a single object-property reference. The MatrixBBC supports a maximum of 256 Trend Log objects.

Trend Log functionality within the MatrixBBC differs greatly in comparison to standard trending capabilities that are used within the AspectFT system architecture. Trend Logs created within the MatrixBBC will retain up to 256 samples until the data needs to be collected and cleared using an automated trend retrieval process, but can be extended to support up to 1999 samples per Trend Log.

CAUTION

Currently, AspectFT v1.08.02 and earlier does not support the ability to retrieve data stored in BACnet Trend Logs. Until such time that support is implemented, you should use AspectFT trending to collect data from any BACnet architecture using the MatrixBBC.

6.4.1 CREATING TREND LOGS IN THE MATRIXBBC

To create a Trend Log object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MF) Max Trend Objects** property. By default, this value is set to 0, indicating no Trend Log objects exist.
3. Write the number of total Trend Log objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Trend Log objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Trend Log objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

6.4.2 CONFIGURING THE OBJECT-PROPERTY FOR SAMPLING

Trend Logs can reference a single point for data collection. The reference to this point is reflected in the log-device-object-property. The point being collected can be any point internal to the MatrixBBC, or a point on a remote BACnet device. To configure, perform the following steps:

1. Using NB-Pro, locate the **log-device-object-property** in the Trend Log object.
2. Using the editor shown, enter the object type, instance, property. For a point residing on a remote device, be sure to enter the remote device's Device Instance into the form.

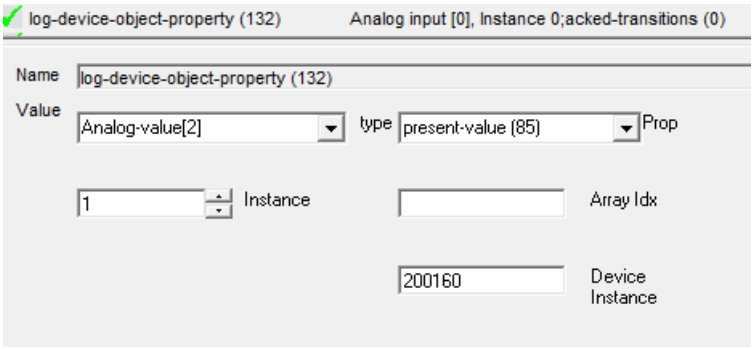

Figure 6-3 - Configuring the log-device-object-property

3. Click *Update Value*.

6.4.3 CONFIGURING THE START AND STOP TIMES

Trend Logs objects can reference a start and stop time that is used to dictate the effective period of when sampling should occur. In order for a Trend Log to initially collect data, the BACnet standard requires a valid **start-time** and **stop-time** to be programmed. These properties indicate the month, date, year, and time for effectivity.

To configure, perform the following steps:
1. Using NB-Pro, locate the **start-time** and **stop-time** properties in the Trend Log object.
2. Using the editor shown, enter a valid date and time for each.

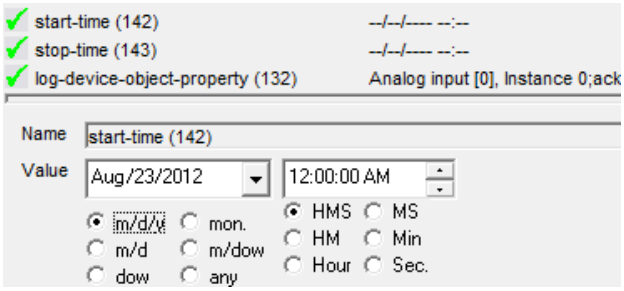

Figure 6-4 - Configuring the start-time and stop-time

3. Click *Update Value*.
4. Ensure that both the **start-time** and the **stop-time** have been configured.

CAUTION

The current revision of the BACnet standard does not permit the use of wildcard placeholders for start-time and stop-time. While the editors in NB-Pro do permit these entries, the product cannot function in this manner at this time. This is a limitation imposed by BACnet Testing Laboratories.

6.4.4 CONFIGURING THE LOGGING TYPE

Trend Logs can collect data either using polled or COV intervals. Polled manners occur on a specific duration interval which is specified in the **log-interval** property. For COV collection, the device you wish to collect data from must support the Subscribe-COV service. In addition to this, you must configure the **cov-resubscription-interval**. By default, re-subscription will occur once ever 60 minutes to ensure values are received when a change occurs.

To configure the logging type, perform the following steps:

1. Using NB-Pro, locate the **logging-type** property. By default, Trend Logs are configured to perform polled collection. If you elect to use polled collection, simply configure the **log-interval**. This property defaults to 6,000 - which is the equivalent of 1 minute.
2. If you wish to collect data via COV, set **logging-type** to a value of Polled (1). COV logging can be performed with standard objects (AI, AO, AV, BI, BO, BV) within the MatrixBBC, or with other remote devices that support COV services.

6.4.5 ENABLING THE TREND LOG

To enable the Trend Log, set **log-enable** to a value of True (1).

SECTION 7: DATA MANIPULATION

This section reviews the Data Manipulation group of objects within the MatrixBBC, which are logic-based blocks used to assist in the setup and creation of control applications.

IN THIS SECTION

7.1 DATA MANIPULATION OVERVIEW

The Data Manipulation object library provides many helpful logic blocks, which can be used to setup linking logic to perform control routines. The Data Manipulation library provides multiple quantities of each object type, including:

- **Math**: used to perform simple math functions such as Add, Subtract, etc.
- **Logic:** used to perform boolean logic functions such as AND, OR, NOT, etc.
- **Min/Max/Avg:** used to determine minimum, maximum, and average values.
- **Enthalpy:** used to calculate enthalpy by referencing temperature and humidity values.
- **Scaling:** used to create a linear, interpolation scale between defined ranges.
- **Input Selects:** used to select one of two input values based on boolean selection criteria.

Through using any of the logic blocks listed above, you can reference the present-value in any logic object within the MatrixBBC.

7.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

Data Manipulation objects exist in the MatrixBBC to reduce the amount of line-by-line SPL programming that one would need to write to carry out advanced control functions. The following are some helpful concepts and techniques to keep in mind when using these objects.

7.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The MatrixBBC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used Data Manipulation object. This allows you not only to keep track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

7.1.1.2 REFERENCING OBJECT PROPERTIES

To use these blocks, you must reference an object property. This is accomplished by identifying the object-identifier, and property using a series of available properties (e.g. ***(IO) Input Object***, (***IP) Input Property***) of each block.

7.1.1.3 THE PRESENT-VALUE PROPERTY

The present-value property of each block (with the exception of Min/Max/Average blocks) is always the result of the logic operation. The value can be referenced by other object functions of the MatrixBBC or shared amongst one another if desired.

7.2 MATH

Math objects are used to apply a chosen math operator against two specific object properties. Configuration of this object involves referencing the two object properties, as well as selecting the operators. The object's ***present-value*** will reflect the result of the math operation.

7.2.1 CREATING MATH OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Math objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC supports up to a maximum of 64 Math objects.

To create a Math object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M7) Max Math Objects** property. By default, this value is set to 0, indicating no Math objects exist.
3. Write the number of total Math objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Math objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Math objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.2.2 MATH OBJECT CONFIGURATION

The ***(OP) Operation*** property specifies the math operator applied against the first and second term. The choices for operation that can be used are displayed in Table 7-1

Table 7-1: Math Object Operator Choices

Value	Operation
0	Disabled
1	Addition
2	Subtraction
3	Multiplication
4	Division
5	Minimum
6	Maximum
7	Average

Each Math object also provides boolean logic feedback on math values. Through several additional properties available in the Math object, users can obtain feedback on Input 1 versus Input 2. Boolean logic provided includes:

. Greater Than - using the **(GT) Input One is > Input 2** property
. Greater Than Equal - using the **(GE) Input One is >= Input 2** property
. Less Than - using the **(LT) Input One is < Input 2** property
. Less Than Equal - using the **(LE) Input One is <= Input 2** property
. Equals - using the **(ET) Input One is = Input 2** property.

7.2.3 FEEDBACK TEXT

Each Math object includes feedback text, providing additional information regarding the overall health and functionality of the object. If both defined inputs are able to be read successfully, the object will report back a message such as *"Object Is Active & Working. Operation: Addition".*

However, if a problem is detected with one or all of the defined inputs, a message such as *"Object Unable to Read Input 1"* will appear. Depending on the object, the message will differ based on the erroring input.

7.3 LOGIC

Logic objects are used to perform logical operations using selectable object properties and a choice of operator. Up to eight (8) object properties can be referenced in a Logic object. The object's ***present-value*** will reflect that result of the logic operation.

7.3.1 CREATING LOGIC OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Logic objects are created by the technician when necessary, and are done in a dynamic manner. The MatrixBBC support up to a maximum of 64 Logic objects.

To create a Logic object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M6) Max Logic Objects** property. By default, this value is set to 0, indicating no Logic objects exist.
3. Write the number of total Logic objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Logic objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Logic objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.3.2 LOGIC OBJECT CONFIGURATION

The ***(OP) Operation*** property specifies the logic operator applied against the referenced object properties. The choices for operator that can be used are displayed in Table 7-2.

Table 7-2: Logic Object Operator Choices

Value	Operation	Notes
0	Disabled	Disables the object.
1	OR	Performs a logical "OR" on all of the referenced object properties. If any of the referenced object properties are true (value of 1), ***present-value*** = true. If all of the referenced object properties are false (value of 0), then ***present-value*** = false.
2	AND	Performs a logical "AND" on all of the referenced object properties. If all of the referenced object properties are true (value of 1), ***present-value*** = true. If any of the inputs are false (value of 0), then ***present-value*** = false.
3	NOT	Performs a logical "NOT" against the first referenced object property (I1 and A1).
4	XOR	Performs a local "XOR" on all of the referenced object properties. If any one of the referenced object properties is true (value of 1), then ***present-value*** = 1. Otherwise, ***present-value*** = false.

7.4 MIN/MAX/AVG

Mix/Max/Avg objects are used to calculate the minimum, maximum, and average values of reference object properties within the block. Up to four (4) object properties can be referenced by each Min/Max/Avg block. The MatrixBBC supports up to a maximum of 64 Math objects.

The ***(HV) High Value*** property will output the highest value of all referenced object property.
The ***(LV) Low Value*** property will output the lowest value of all referenced object property.
The ***(AV) Average Value*** property will reference the arithmetic mean of all referenced object properties.

7.4.1 CREATING MIN/MAX/AVG OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Min/Max/Avg objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Min/Max/Avg object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M8) Max MinMaxAvg Objects** property. By default, this value is set to 0, indicating no Min/Max/Avg objects exist.
3. Write the number of total Min/Max/Avg objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Min/Max/Avg objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Min/Max/Avg objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.5 ENTHALPY

Enthalpy objects are used to calculate enthalpy based on a referenced temperature and referenced humidity value. The referenced values can be connected to inputs, retrieved using Netmap Objects, taken from Analog Value objects, etc. The MatrixBBC support up to a maximum of 64 Enthalpy objects.

The ***present-value*** is the result of this calculation.

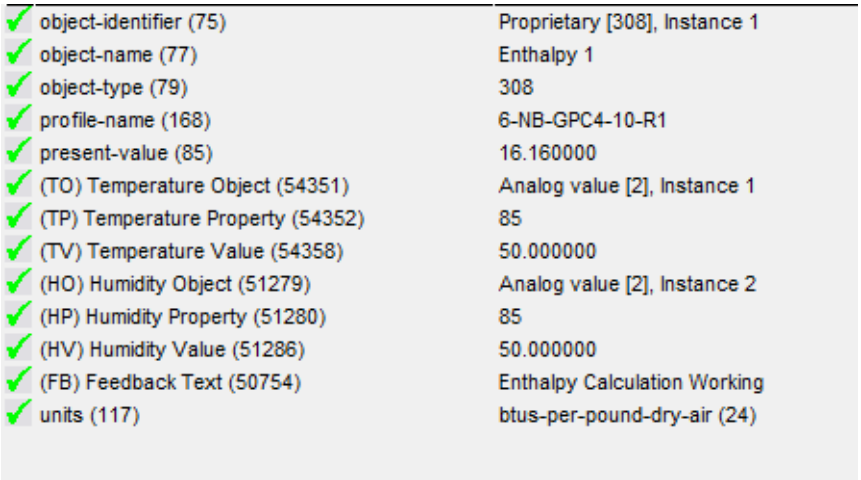

Figure 7-1 - Enthalpy Object Example

7.5.1 CREATING ENTHALPY OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Enthalpy objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Enthalpy object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(ME) Max Enthalpy Objects** property. By default, this value is set to 0, indicating no Enthalpy objects exist.
3. Write the number of total Enthalpy objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Enthalpy objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Enthalpy objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.6 SCALE

Scale objects perform a linear interpolation between two known points, which can be used to scale a single value within programming by looking up values that lie along the created linear segment. A referenced object property's value is applied against the scale. As a result, ***present-value*** will indicate the calculated scale value. The MatrixBBC will support up to a maximum of 64 Scale objects.

7.6.1 CREATING SCALE OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Scale objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Scale object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M4) Max Scale Objects** property. By default, this value is set to 0, indicating no Scale objects exist.
3. Write the number of total Scale objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Scale objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Scale objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.6.2 SCALE OBJECT CONFIGURATION

Properties **(X1) Input Range X1 Value**, **(X2) Input Range X2 Value** and **(Y1) Output Range Y1**, **(Y2) Output Range Y2** indicate the x- and y-coordinate values to be used for the starting and ending points of the line segment. Both x- and y-coordinate values are given in engineering units of your input object property.

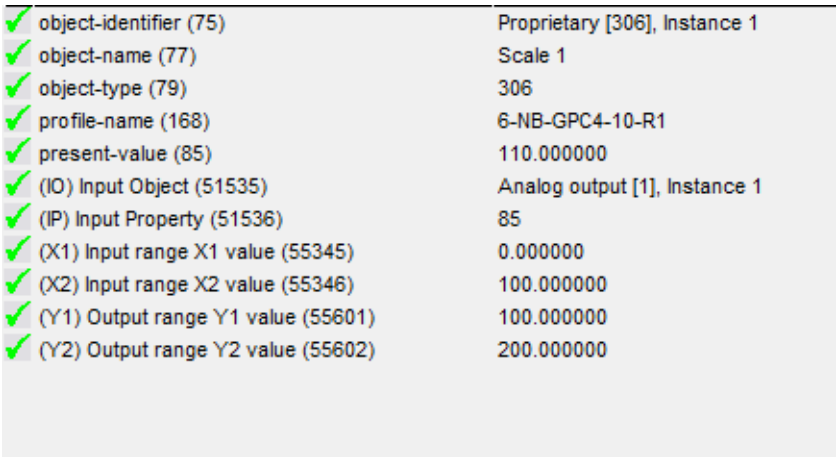

Figure 7-2 - Scaling Example

In the example shown above, we have configured an Input range for 0 and 100, and our output range for 100 to 200. In this example, when our input value (which is linked to Analog Output 1; present-value) is at a value of 10.0, the result of the scale will output a value of 110.0.

7.7 INPUT SELECT

Input Select objects allow you to choose one of two referenced object property values based on a true/ false input status. The MatrixBBC supports up to a maximum of 64 Input Select objects.

7.7.1 CREATING INPUT SELECT OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Input Select objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Input Select object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M9) Max Input Select Objects** property. By default, this value is set to 0, indicating no Input Select objects exist.
3. Write the number of total Input Select objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Input Select objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Input Select objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.7.2 INPUT SELECT OBJECT CONFIGURATION

To use an Input Select object, you must reference two object properties, along with a selection criteria reference.

If the selection criteria is false (value of 0), then **present-value** will equal the value of the first object property reference (I1, A1). If the selection criteria is true (value other than 0), then **present-value** will equal the value of the second object property reference (I2, A2).

object-identifier (75)	Proprietary [300], Instance 1
object-name (77)	Input Select 1
object-type (79)	300
profile-name (168)	6-NB-GPC4-10-R1
present-value (85)	50.000000
(I1) Input Object 1 (51505)	Analog value [2], Instance 1
(A1) Input Property 1 (49457)	85
(I2) Input Object 2 (51506)	Analog value [2], Instance 2
(A2) Input Property 2 (49458)	85
(SC) Selection Object (54083)	Binary value [5], Instance 1
(SA) Selection Property (54081)	85
(FB) Feedback Text (50754)	Object Reflects Input 1

Figure 7-3 - Input Select Example

7.8 STAGING

Staging objects are used to perform staging of binary outputs for control related purposes, but can also be used to trigger other control logic if deemed necessary. Each Staging object provides support for multiple staged outputs (as few as two, as many as eight), each with a dedicated setpoint, feedback status, and runtime timer. Stages can be configurably transitional for lead/lag, and wear leveling. The MatrixBBC supports up to a maximum of 16 Staging objects.

object-identifier (75)	Proprietary [309], Instance 1
object-name (77)	Staging 1
object-type (79)	309
profile-name (168)	6-NB-GPC1-7-R1
(SM) Staging Mode (54093)	Object Turned Off (0)
(LM) Lead/Lag/Leveling Mode (52301)	Normal <First On Last Off> (0)
(NS) Number Of Stages <Max Loading> (52819)	8 Stages (8)
(IO) Input Object (51535)	Analog input [0], Instance 0
(IP) Input Property (51536)	0
(IV) Input Value (51542)	0.000000
(II) Invert The Input? (51529)	False (0)
(IS) Invert The Setpoints? <Higher Stages=Lower Numbers> (51539)	False (0)
(OO) Interlock Override Object (53071)	Analog input [0], Instance 0
(OP) Interlock Override Property (53072)	0
(OM) Interlock Staging Map (53069)	Stage 1 (FALSE),Stage 2 (FALSE),Stage 3 (FALSE),Stage 4 (FA
(OS) Interlock Status (53075)	Object is Turned Off
(P1) Stage 1 Setpoint (53297)	0.000000
(P2) Stage 2 Setpoint (53298)	0.000000
(P3) Stage 3 Setpoint (53299)	0.000000
(P4) Stage 4 Setpoint (53300)	0.000000
(P5) Stage 5 Setpoint (53301)	0.000000
(P6) Stage 6 Setpoint (53302)	0.000000
(P7) Stage 7 Setpoint (53303)	0.000000
(P8) Stage 8 Setpoint (53304)	0.000000
(LD) Loading Interval <Seconds> (52292)	60
(UD) Unloading Interval <Seconds> (54596)	60
(LR) Seconds Until Next Loading Event Could Occur (52306)	0
(UR) Seconds Until Next Unloading Event Could Occur (54610)	0
(PR) Present Stages Of Loading (53330)	0
(S1) Stage 1 Status (54065)	NULL
(S2) Stage 2 Status (54066)	NULL
(S3) Stage 3 Status (54067)	NULL
(S4) Stage 4 Status (54068)	NULL
(S5) Stage 5 Status (54069)	NULL

Figure 7-4 Staging Object View

7.8.1 CREATING STAGING OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Staging objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Staging object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MJ) Max Staging Objects** property. By default, this value is set to 0, indicating no Staging objects exist.
3. Write the number of total Staging objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Staging objects, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Staging objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

7.8.2 BASIC CONFIGURATION

The following basic configuration items should be taken into account prior to defining the staging mode.

7.8.2.1 CONFIGURING THE INPUT OBJECT PROPERTY REFERENCE

The Staging object will energize and de-energize stages based on the value received from the configured input object-property reference. The input object-property reference is configured using properties **(IO) Input Object** and **(IP) Input Property**.

When the Staging object is activated, the current value of the input object property will be reflected in property **(IV) Input Value**.

7.8.2.2 CONFIGURING THE NUMBER OF STAGES

Determine how many output stages you wish to have. This is defined using the **(NS) Number of Stages <Max Loading>** property. Each Staging object can support as few as two outputs, or as many as eight output stages.

When you have successfully configured the entire Staging object for control, the control value for each Stage is defined in property **(S1) Stage 1 Status** through **(S8) Stage 8 Status**. In Binary Output control methods, a corresponding stage status property would be addressed by the Binary Output's AutoStuff process (reference the Outputs Setup section for additional information).

7.8.2.3 CONFIGURING THE LEAD/LAG/LEVELING MODE

Output stages can be enabled/disabled based on Normal Mode, or through Wear Leveling.

In the Normal Mode, stage outputs will energize in a classic First On / Last Off method. For example, when stages are called, they will be energized in logical order (Stage 1, Stage 2, Stage 3,...). When de-activation occurs, the last stage on will be turned off first (Stage 8, Stage 7, Stage 6,...).

In Wear Leveling Mode, stages will be turned on and off based on the runtimes for each stage. This method allows each mechanical stage to be used for an even amount of time - thereby increasing the lifespan of equipment. Runtimes are tracked for each stage through properties **(R1) Stage 1 Runtime** through **(R8) Stage 8 Runtime**. When stages are called, stages are energized based on the least amount of runtime.

7.8.2.4 CONFIGURING THE CONTROL SIGN

Staging is commonly used for heating or cooling. This is controlled by how setpoints control the stages. Property **(IS) Invert the Setpoints? <Higher Stages = Lower Numbers>** essentially commands how staging will work.

When set to *False*, the Staging object will work in a Heating-like mode, where stage outputs are enabled as the input variable exceeds defined setpoints.

When set to *True*, the Staging object will work in a Cooling-like mode, where stage outputs are enabled as the input variable falls below defined setpoints.

7.8.3 STAGING MODES

The Staging object provides two different modes of how outputs can be staged. The mode is directly controlled through (SM) Staging Mode. There are two options that can be selected for staging.

7.8.3.1 DELAY ON/DELAY OFF

The Delay On/Delay Off mode utilizes two definable setpoints to enable and disable stages based on timed intervals. When Delay On/Delay Off mode has been selected, the object will transition its properties

to provide a **(SU) Unloading Setpoint** and **(SL) Loading Setpoint**. These setpoints determine when stages will be disabled (unloaded) and enabled (loaded).

When the referenced object-property exceeds the value defined in **(SL) Loading Setpoint**, output stages will be energized (turned on) based on property **(LD) Loading Interval <Seconds>**. In this scenario, the first stage will be energized, and wait the amount of time specified in **LD**. Once the time has expired, the next stage will be energized, followed by the delay specified in **LD**. This process will repeat until all stages have been energized.

When the referenced object-property falls below the value defined in **(SU) Unloading Setpoint**, output stages will be de-energized (turned off) based on property **(UD) Unloading Interval <Seconds>**. In this scenario, the last stage energized during loading will be de-energized, and wait the amount of time specified in **UD**. On the time has expired, the next stage will be de-energized, followed by the delay specified in **UD**. This process will repeat until all stages have been deactivated.

For troubleshooting and convenience, two timers are provide to allow users to know when the next loading or unloading event will occur. These properties, **(LR) Seconds Until Next Loading Event Could Occur** and **(UL) Seconds Until Next Unloading Event Could Occur**, can be found in the Staging object and monitored using an engineering tool or front-end.

7.8.3.2 THRESHOLD BASED STAGING

The Threshold Based Staging mode utilizes a setpoint for each individual stage, rather than a single setpoint. When Threshold Based Staging mode has been selected, the object transition its properties to provide up to eight setpoints properties, defined as **(P1) Stage 1 Setpoint** through **(P8) Stage 8 Setpoint**.

Stages will be energized (turned on) when the referenced input object-property has exceeded each defined stage setpoints. In the event that a large value increase occurs, multiple stages will be energized in a time delayed manner based on the configuration of **(LD) Loading Interval <Seconds>**.

When the referenced object-property falls below each defined setpoint, output stages will be de-energized (turned off). In the event of a large decrease of the input object-property value, multiple stages will be de-energized in time delayed manner based on the configuration of **(UD) Unloading Interval <Seconds>**.

7.8.4 STAGE INTERLOCKING

Staging can also be subject to interlocking. Interlocking may be used to lock out stages in certain situations (e.g. supply air temperature or outside air temperature exceeds a specific setpoint value).

The Interlock input is defined using properties **(OO) Interlock Override Object** and **(OP) Interlock Override Property**. When the Interlock input is a non-zero value, all of the Stages will become interlocked, whereas a zero value will disable interlocking - allowing the Staging object to resume normal operations.

The state of each stage (enabled/disabled) when the Interlock input is a non-zero value is controlled through property **(OM) Interlock Staging Map**. When a specific stage has a check mark next to it, this commands the Interlock routine to energize (turn on) the corresponding stage. When a specific stage has no check mark next to it, this commands the Interlock routine to de-energize (turn off) the corresponding stage.

For troubleshooting purposes, property **(OS) Interlock Status** will provide plain-English feedback as to the current status of the Interlocking process.

SECTION 8: DATA MOVEMENT

This section reviews the Data Movement group of objects within the MatrixBBC, which are logic-based blocks which can be used to move values from one object to another within the controller, broadcast values to a group of devices, or even read and write information over the BACnet network to/from peer devices.

IN THIS SECTION

8.1 DATA MOVEMENT OVERVIEW

The Data Movement object library provides many helpful logic blocks, used to move data from one part of the controller to another.

. **Broadcasts**: used to send/receive information to multiple controllers without writing complex logic.
. **Local Remaps:** used to move data back and forth between two different points local to the MatrixBBC.
. **Netmaps:** exactly like a Local Remap, but can also move data to/from other devices on the network.

Through using any of the logic blocks listed above, you can reference the present-value in any logic object within the MatrixBBC.

8.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

Data Movement objects exist in the MatrixBBC to reduce the amount of line-by-line SPL programming that one would need to write to carry out advanced control functions. The following are some helpful concepts and techniques to keep in mind when using these objects.

8.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The MatrixBBC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used Data Movement object. This allows you not only to keep track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

8.1.1.2 REFERENCING OBJECT PROPERTIES

To use these blocks, you must reference an object property. This is accomplished by identifying the object-identifier, and property using a series of available properties (e.g. **(O1) Input Object**, (**P1) Input Property**) of each block.

8.1.1.3 THE PRESENT-VALUE PROPERTY

The present-value property of each block is always the result of the operation. The value can be referenced by other object functions of the MatrixBBC or shared amongst one another if desired.

8.2 BROADCASTS

Broadcast objects allow the MatrixBBC to send and receive values over the network at configured time intervals to AAM Native Series devices. The MatrixBBC will support up to a maximum of 8 Broadcast objects. When an object is configured to send or receive a broadcast, **present-value** will display the property value being sent or received from the network.

NOTE

Broadcast objects are intended to send “primitive” datatypes, such as REAL, UNSIGNED, BOOLEAN, etc.

8.2.1 CREATING BROADCAST OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Broadcast objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Broadcast object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MT) Max Broadcast Objects** property. By default, this value is set to 0, indicating no Broadcast objects exist.
3. Write the number of total Broadcast objects you wish to have in the MatrixBBC. For example, if you wish to have all 8 Broadcast objects available, write a value of 8.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Broadcast objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

8.2.2 BROADCASTING CONCEPTS

Before configuring broadcasts on your BACnet network, there are a few concepts that should be followed when performing Broadcasts.

8.2.2.1 OUTSIDE AIR TEMPERATURE BROADCASTS

If you intend on sending an Outside Air Temperature broadcast to other Native Series devices, such as NB-ASC(e) devices, you must send the broadcast using Broadcast, Instance 3. The value sent must be a floating point datatype.

CAUTION

Configuring the Outside Air Temperature Broadcast object to send a value that is not a floating point datatype will result in recipient NB-ASC family controllers rejecting the data.

8.2.2.2 SCHEDULE BROADCASTS

If you intend on sending a Schedule broadcast to other Native Series devices, such as NB-ASC(e) or NB-VAV devices, you must send the broadcast using Broadcast, Instance 5. More so, the datatype must be an Unsigned Integer.

CAUTION

Configuring the Schedule Broadcast object to send a value that is not a Unsigned Integer datatype will result in recipient NB-ASC family controllers rejecting the data.

8.2.3 SENDING A BROADCAST

To configure an object to send a broadcast, perform the following steps:

1. Configure ***(BM) Broadcasting Mode*** = Send (1).
1. Reference the object property you wish to broadcast by configuring ***(IO) Input Object*** and ***(IP) Input Property.***
2. Configure ***(BZ) Broadcast Zone/Global*** accordingly.
3. Configure ***(ZN) Zone Number*** for the controller zone you wish to send the broadcast to.
4. Configure ***(BT) Broadcast Time Interval***. Determine the time interval, in minutes, you wish to have the MatrixBBC send the broadcast.

8.2.4 RECEIVING A BROADCAST

To configure an object to send a broadcast, perform the following steps:

1. Configure ***(BM) Broadcasting Mode*** = Receive (2).
2. Configure ***(ZN) Zone Number*** for the controller zone you wish to send the broadcast to.

8.2.5 FEEDBACK AND STATUS INFORMATION

Broadcast objects provide a feedback property, **(FB) Feedback Text,** that provides up to date information regarding the health of the object. This property can be monitored via NB-Pro or even an operator workstation or web server. The following table provides a list of the messages and statuses.

Table 8-1: Broadcast Feedback Text Notes

Feedback	Notes
Broadcast Object Turned Off	Broadcast object is disabled
Broadcast Object Active & Transmitting	Self-explanatory
Broadcast Object Receiving	Self-explanatory
Broadcast Object Unable to Read Input	Self-explanatory

8.3 LOCAL REMAPS

Local Remap objects are used to move data from one point to another in the controller. Effectively, Local Remap objects accomplish an equate. In previous generations of controllers, line-by-line SPL was required to stuff a value from one place to another. This object eliminates the need to do so. The MatrixBBC will support up to a maximum of 64 Local Remap objects.

To use a Local Remap, you must specify an input object property and an output object property. The input object property, defined by properties **(O1) Input Object Reference** and **(P1) Input Property Reference**. This is essentially the left side of an equate statement.

The value of the referenced input object property will be transferred to the output object property, defined by properties **(O2) Output Object Reference** and **(P2) Output Property Reference**. For situations where you are remapping a value to the present-value property of a commandable object (AO, AV, BO, BV, etc.), you must specify the priority array level which the value will be assigned to. This is accomplished through **(Q2) Output Priority Level**. By default, this value is set to 255 for no priority. A priority range between 1 (highest priority) and 16 (lowest priority) is used by commandable objects.

8.3.1 CREATING LOCAL REMAP IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Local Remap objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Local Remap object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the (MA) Max Remap property. By default, this value is set to 0, indicating no Local Remap objects exist.
3. Write the number of total Local Remap objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Local Remap, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Local Remap objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

8.3.2 REMAP MODE

Local Remaps can be configured to remap data on a continuous method, or a trigger method. A method is chosen through **(RM) Remap Mode**. By default, remap objects are disabled. To enable the object, you must select a valid mode from this property.

Table 8-2: Remap Modes

Remap Mode	Notes
Disabled	Remap object is disabled. No data transfer occurs.
Continuous	Remap object is enabled. Data transfer occurs continuously.
When Triggered	Remap object is enabled. Data transfer occurs continuously when the referenced trigger object property value is active. If the trigger referenced object property is inactive, no data transfer occurs.

Table 8-2: Remap Modes

Remap Mode	Notes
When Triggered Else NULL	Remap object is enabled. Data transfer occurs continuously when the referenced trigger object property value is active. If the trigger referenced object property is inactive, a NULL value is sent. This Remap Mode is intended for use with writing to the **present-value** of a commandable object (AO, AV, BO, BV, etc.)

NOTE

When configured for continuous mode, the MatrixBBC will transfer data from input to output once per second.

If you have elected to configure your remap object to operate by a trigger, you must specify a trigger object property through properties **(TO) Trigger Object** and **(TP) Trigger Property**. Finally, configure the **(TB) Trigger Biasing** to allow the Remap object to know what type of input value should be considered active or inactive. A trigger can be considered active when a non-zero value is referenced, or when a zero-based value is referenced.

8.3.3 DATA COERCION PROTECTION

Remap objects do provide good protection relative to data coercion (mis-matching data types) within logic automatically.

Remaps will read from the specified input object property and will display that data in the present-value property of the Local Remap. The present-value uses the same datatype as the input object property. When that variable is then written to the output object property, the data is initially written as-is. If the data is rejected due to it being the wrong datatype (e.g. remapping BO1; present-value to AO1; present-value), the data will then be forced into the datatype of the value currently in the output object property. The data is then written once more, using the preferred datatype.

8.3.4 FEEDBACK AND STATUS INFORMATION

Local Remap objects provide two feedback properties that provide up to date information regarding the health of the object. These properties, **(FB) Feedback Text** and **(RS) Remap Status** will provide specific information that can be monitored via NB-Pro or even an operator workstation or web server. The following table provides a list of the messages and statuses.

Table 8-3: Local Remap Object Feedback Information

Feedback Text	Notes
Remap Object Turned Off	Remap object is disabled via **(RM) Remap Mode**.

Table 8-3: Local Remap Object Feedback Information

Feedback Text	Notes
Remap Object Active & Working	Remap object is working and actively transferring data.
Remap Object Active & Working (And Coercing Datatype)	Remap object is working, actively transferring data, and is also coercing the datatype between the input and output.
Remap Object Unable to Read Trigger	Remap object cannot read the referenced trigger input object property specified in **(TO) Trigger Object** and **(TP) Trigger Property**.
Remap Object Unable to Read Input	Remap object cannot read the referenced input object property specified in **(O1) Input Object** and **(P1) Input Property**.
Remap Object Unable to Write Output	Remap object cannot transfer data to the output object property specified in **(O2) Output Object** and **(O2) Output Property**.

8.4 NETMAP OBJECTS

Very similar to Local Remap objects, Netmap objects provide data movement both within the MatrixBBC itself, as well as mode data to and from other devices on the network. Using a Netmap object, the MatrixBBC can write data from within its control routine to another device on the BACnet network. Alternatively, the MatrixBBC can take a value from a peer device on the BACnet network and write it to another, different peer device on the BACnet network, peer to peer, or even MS/TP slaves on a local MS/TP network. The MatrixBBC will support up to a maximum of 64 Netmap objects.

To use a Netmap, you must specify an input device-object-property, and an output device-object-property. The input object property, defined by properties **(II) Input Device Instance**, **(O1) Input Object Reference** and **(P1) Input Property Reference**. This is essentially the left side of an equate statement.

The value of the referenced input object property will be transferred to the output device-object-property reference, defined by properties **(OI) Output Device Instance**, **(O2) Output Object Reference** and **(P2) Output Property Reference**. For situations where you are mapping a value to the present-value property of a commandable object (AO, AV, BO, BV, etc.), you must specify the priority array level which the value will be assigned. This is accomplished through **(Q2) Output Priority Level**. By default, this value is set to 255 for no priority. A priority range between 1 (highest priority) and 16 (lowest priority) is used by commandable objects.

Finally, configure the **(TM) Time Between Writes in Seconds** property to specify how often writes should occur. A valid write time can be no faster than 30 seconds.

8.4.1 CREATING NETMAP OBJECTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Netmap objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Netmap object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MB) Max Netmap Objects** property. By default, this value is set to 0, indicating no Netmap objects exist.
3. Write the number of total Netmap objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Netmap, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Netmap objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

8.4.2 NETMAP MODE

Netmaps are configured to remap data on a continuous method, or a trigger method. A method is chosen through **(NM) Netmap Mode**. By default, Netmap objects are disabled. To enable the object, you must select a valid mode from this property.

Table 8-4: Netmap Modes

Remap Mode	Notes
Disabled	Netmap object is disabled. No data transfer occurs.
Continuous	Netmap object is enabled. Data transfer occurs continuously.

Table 8-4: Netmap Modes

Remap Mode	Notes
When Triggered	Netmap object is enabled. Data transfer occurs continuously when the referenced trigger object property value is active. If the trigger referenced object property is inactive, no data transfer occurs.
When Triggered Else NULL	Netmap object is enabled. Data transfer occurs continuously when the referenced trigger object property value is active. If the trigger referenced object property is inactive, a NULL value is sent. This Netmap Mode is intended for use with writing to the **present-value** of a commandable object (AO, AV, BO, BV, etc.)

If you have elected to configure your Netmap object to operate by a trigger, you must specify a trigger object property through properties **(TO) Trigger Object** and **(TP) Trigger Property**. Finally, configure the **(TB) Trigger Biasing** to allow the Netmap object to know what type of input value should be considered active or inactive. A trigger can be considered active when a non-zero value is referenced, or when a zero-based value is referenced.

8.4.3 FEEDBACK AND STATUS INFORMATION

Netmap objects provide two feedback properties that provide up to date information regarding the health of the object. These properties, **(FB) Feedback Text** and **(RS) Remap Status** will provide specific information that can be monitored via NB-Pro or even an operator workstation or web server. The following table provides a list of the messages and statuses.

Table 8-5: Netmap Object Feedback Information

Feedback Text	Notes
Netmap Object Turned Off	Netmap object is disabled via **(NM) Netmap Mode**.
Netmap Object Active & Working	Netmap object is working and actively transferring data.
Netmap Object Active & Working (And Coercing Data)	Netmap object is working, actively transferring data, and is also coercing the datatype between the input and output.
Netmap Trigger is Off	Netmap object cannot read the referenced trigger input object property specified in **(TO) Trigger Object** and **(TP) Trigger Property**.
Netmap Object Unable to Read Trigger	Netmap object cannot read the referenced input object property specified in **(O1) Input Object** and **(P1) Input Property**.
Netmap Object Unable to Read Input	Netmap object cannot transfer data to the output object property specified in **(O2) Output Object** and **(O2) Output Property**.

Table 8-5: Netmap Object Feedback Information

Feedback Text	Notes
Netmap Object Unable to Write Output	Netmap object is disabled via **(NM) Netmap Mode**.
Netmap Object Misconfigured	Netmap object is working and actively transferring data.

SECTION 9: EXPANSION I/O

This section discusses STATbus Expansion IO, including wiring and programming.

IN THIS SECTION

9.1 WHAT ARE IOX MODULES?

IOX modules are specialized STATbus devices which allow you to add remote I/O points to the MatrixBBC

These units have on-board I/O and, include the ability to add additional I/O, using IOX modules to achieve the number of inputs and outputs needed. This allows you to craft a controller with a completely customized I/O profile. In this way, you can tailor the controller to suit the job rather than designing the job around the capabilities of a controller. This will allow you to make decisions based on good design principles rather than system limitations.

9.1.1 FEATURES OF IOX MODULES

. Provide remote I/O points to MatrixBBC controllers
. Provide the ability to locate I/O hardware where it is most convenient
. Communication via STATbus
. Easy 2- or 4- wire connection using twisted pair wire
. Easy configuration within MatrixBBC using NB-Pro.

9.1.2 REMOTE I/O AND MAPPING POINTS

IOX modules provide additional, remote I/O points to MatrixBBC controllers. Modules exist that can provide additional Universal Input, Pulse Input, Analog Output and Digital Output points. These points appear to the controller to be identical to an on-board input or output, therefore only minimal additional work is needed when commissioning IOX modules

Remote I/O behaves in the same way as on-board I/O, except that it is located remotely from the controller. Because they are not on-board, each remote device requires a unique address, known as a Global Identification (GID) number so that the controller may recognize and direct communications to it. When working with IOX modules, there is the additional commissioning step of associating the remote I/O point with inputs and outputs within the controller. This is accomplished by assigning the GID number of the device to the desired input or output. Once the IOX module is mapped in this way, it will function as any other input or output of the same type.

9.2 IOX MODULE SPECIFICATIONS

9.2.1 GENERAL

9.2.1.1 NETWORKING

. **communications protocol:** STATbus
. **wiring:** 2- or 4-wire (device dependent), 18-22 ga., twisted pair
. **update frequency:** nominally every 100 mS
. **network configuration:** multidrop bus

9.2.1.2 TERMINATIONS

. Pluggable terminal blocks for inputs and/or outputs, power and network connection.

9.2.1.3 OPERATING ENVIRONMENT

. **temperature range:** 32-122°F (0-50°C)
. **humidity range:** 0-80% RH, non-condensing

9.2.1.4 AGENCY APPROVALS

. UL listed 916, Management Equipment, Energy (PAZX)
. FCC rules Part 15 Class B Computing Device
. UL 873 Recognized, Component-Temperature Indicating and Regulating Equipment
. Complies with CE directives and standards (XAPX2)

9.2.2 SSB-FI1

9.2.2.1 I/O

. One (1) 12-bit Universal Input (interpolated to a 16-bit value)
. Selectable 0-5 VDC,0-10 VDC, 0-20 mA or 0-250 kΩ input range

9.2.2.2 POWER REQUIREMENTS

. None

9.2.2.3 DIMENSIONS

. **size:** 3.02 x 1.41 x. 0.95in. (7.67 x 3.58 x 2.41 cm)
. **shipping weight:** .04 lb. (.018 kg)

9.2.3 SSB-UI1

9.2.3.1 I/O

. One (1) 24-bit Universal Input
. Selectable 0-5 VDC,0-10 VDC, 0-20 mA or 0-250 kΩ input range

9.2.3.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, 1 A (max)

9.2.3.3 DIMENSIONS

. **size:** 4.2 x 4.2 x 1.0 in. (10.67 x 10.67 x 2.54 cm)
. **shipping weight:** .50 lb. (.23 kg)

9.2.4 SSB-AO1

9.2.4.1 I/O

. One (1) Analog Output
. Selectable 0-10 VDC or 0-20 mA output range

9.2.4.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, 1 A (max)

9.2.4.3 DIMENSIONS

. **size:** 4.2 x 4.2 x 1.0 in. (10.67 x10.67 x2.54 cm)
. **shipping weight:** .50 lb. (.23 kg)

9.2.5 SSB-DI1

9.2.5.1 I/O

. One (1) Optically Isolated, Pulse Counting, Digital Input

9.2.5.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, 1 A (max)

9.2.5.3 DIMENSIONS

. **size:** 4.2 x 4.2 x 1.0 in. (10.67 x 10.67 x 2.54 cm)
. **shipping weight:** .50 lb. (.23 kg)

9.2.6 SSB-DO1

9.2.6.1 I/O

. One (1) Digital Output (relay)

9.2.6.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, .25 A (max)

9.2.6.3 DIMENSIONS

. **size:** 4.75 x 3.25 x 2.0 in. (12.07 x 8.26 x 5.08 cm)
. **shipping weight:** .50 lb. (.23 kg)

9.2.7 SSB-DO1-I

9.2.7.1 I/O

. One (1) Digital Output (relay)
. One (1) Digital Input (dry contact only, no pulse counting)

9.2.7.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, .25 A (max)

9.2.7.3 DIMENSIONS

. **size:** 4.75 x 3.25 x 2.0 in. (12.07 x 8.26 x 5.08 cm)
. **shipping weight:** .50 lb. (.23 kg)

9.2.8 SSB-DO2

9.2.8.1 I/O

. Two (2) Digital Outputs (relays)

9.2.8.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, .25 A (max)

9.2.8.3 DIMENSIONS

. **size:** 4.75 x 3.25 x 2.0 in. (12.07 x 8.26 x 5.08 cm)
. **shipping weight:** .56 lb. (.25 kg)

9.2.9 SSB-DO2-I

9.2.9.1 I/O

. Two (2) Digital Outputs (relays)
. Two (2) Digital Inputs (dry contacts only, no pulse counting)

9.2.9.2 POWER REQUIREMENTS

. 24VAC, 50/60 Hz, .25 A (max)

9.2.9.3 DIMENSIONS

. **size:** 4.75 x 3.25 x 2.0 in. (12.07 x 8.26 x 5.08 cm)
. **shipping weight:** .56 lb. (.25 kg)

9.2.10 SSB-IOX1-1

9.2.10.1 I/O

- Four (4) 24-bit Universal Inputs
- Selectable 0-5 VDC,0-10 VDC, 0-20 mA or 0-250 kΩ input range
- One (1) Optically Isolated, Pulse Counting, Digital Input
- Two (2) Analog Outputs
- Selectable 0-10 VDC or 0-20 mA output range
- Two (2) Digital Outputs (triacs)

9.2.10.2 POWER REQUIREMENTS

- 24VAC, 50/60 Hz, 1.85 A (max)

9.2.10.3 DIMENSIONS

- **size:** 5.75 x 6.35 x 1.05 in. (14.60 x 16.13 x 2.67 cm)
- **shipping weight:** .95 lb. (.42 kg)

9.2.11 SSB-IOX1-2

9.2.11.1 I/O

- Eight (8) 24-bit Universal Inputs
- Selectable 0-5 VDC, 0-20mA or -250 kΩ input range.

9.2.11.2 POWER REQUIREMENTS

- 24VAC, 50/60 Hz, 1.85 A (max)

9.2.11.3 DIMENSIONS

- **size:** 5.75 x 6.35 x 1.05 in. (14.60 x 16.13 x 2.67 cm)
- **shipping weight:** .95 lb. (.42 kg)

9.2.12 SSB-IOX2-1

9.2.12.1 I/O

- Twelve (12) 24-bit Universal Inputs
- Selectable 0-5 VDC,0-10 VDC, 0-20 mA or 0-250 kΩ input range
- Six (6) Analog Outputs
- Selectable 0-10 VDC or 0-20 mA output range
- Six (6) Digital Outputs (triacs)

9.2.12.2 POWER REQUIREMENTS

- 24VAC, 50/60 Hz, 1.85 A (max)

9.2.12.3 DIMENSIONS

- **size:** 8.407 x 6.5 x 1.25 in. (20.83x16.51x3.18 cm)
- **shipping weight:** 3 lb. (1.36 kg)

9.2.13 SSB-IOX2-2

9.2.13.1 I/O

- Twelve (12) 24-bit Universal Inputs
- Selectable 0-5 VDC,0-10 VDC, 0-20 mA or 0-250 kΩ input range

9.2.13.2 POWER REQUIREMENTS

- 24VAC, 50/60 Hz, 1.85 A (max)

9.2.13.3 DIMENSIONS

- **size:** 8.407 x 6.5 x 1.25 in. (20.83x16.51x3.18 cm)
- **shipping weight:** 3 lb. (1.36 kg)

9.3 LENGTH OF THE NETWORK

The distance measured from the controller to the STATbus device on the network located furthest away from it should not exceed 1000' in length. The STATbus shown in Figure 9-1a is a valid configuration because the distance from the controller to the most distant device is less than 1000' whereas the configuration shown in Figure 9-1b is not valid because the total length to the most distant device is 1150', exceeding the 1000' maximum.

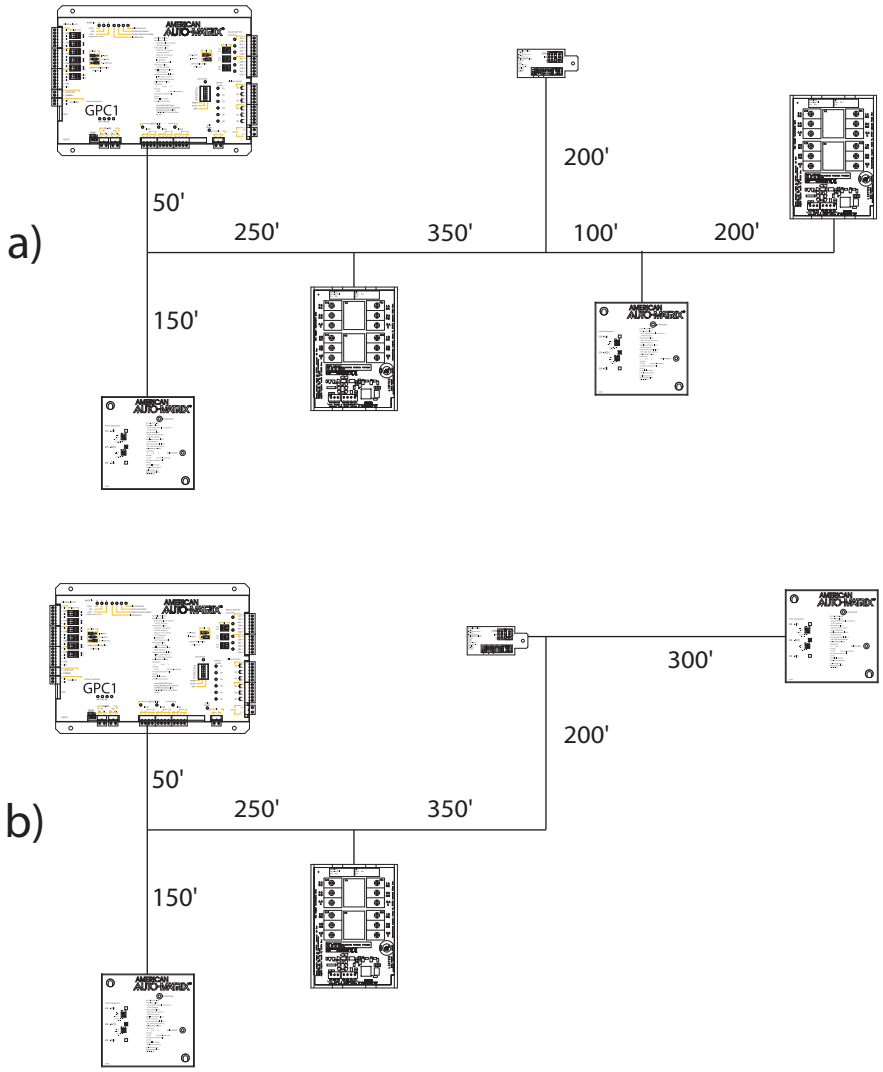

Figure 9-1 Determining Maximum STATbus Length

9.4 NUMBER OF DEVICES

Each STATbus channel on the controller will support a maximum of thirteen (13) devices.

CAUTION

A maximum of thirteen (13) devices can be connected to a single STATbus channel. If more than thirteen devices are connected, only thirteen will be enumerated by the controller. If more than thirteen devices are connected, there is no way to predict which devices will be enumerated and which will be left off.

9.4.1 COMMUNICATIONS LIMITS

While the STATbus protocol allows up to thirteen devices to be connected to a single network, certain devices reduce the maximum number of other devices that may be used on a single channel. In particular, STAT1D, STAT2D, and STAT3 have a higher power requirement than other STATbus devices and limit the total number of devices that can be put on the network and still communicate. When one or more STAT devices are included on the network, the maximum number of devices allowed on the network will be reduced.

9.4.1.1 NO STATS ON THE STATBUS

If your STATbus channel does not have any STATs (STAT1D, STAT2D, or STAT3) on it, then you may have up to thirteen devices in any combination on the network. This may include SSB-FI1s, SSB-UI1s, SSB-AO1s, SSB-DI1s, SSB-DO1s, SSB-DO1-Is, SSB-DO2s, SSB-DO2-Is, and SSB-IOX1 modules.

9.4.1.2 ONE OR MORE STATS ON THE STATBUS

If one or more STATs are being used, the total number of devices that can communicate on a single STATbus channel will be reduced. Table 9-1 lists the number of additional STATbus devices that can be connected for a given number of STATs.

Table 9-1 Number of Devices Allowed on a STATbus

Number of STATs	Number of other STATbus Devices
1	11
2	9
3	7
4	5
5	2

9.4.1.3 EXAMPLE: NO STATS ON THE STATBUS

With no STATs on the Bus, you may use any combination of SSB devices, up to a maximum of thirteen devices total. This means any of the following would be valid:

. 13 SSB-FI1s to read a number of different inputs
. 6 SSB-FI1s and 6 SSB-AO1s to provide simple damper control for six zones

. 3 SSB-UI1s, 3 SSB-AO1s, and 3 SSB-DO1-Is, giving three zones with a zone temperature input, control for a damper and a reheat coil with supervisory monitoring

9.4.1.4 EXAMPLE: STATS ON THE STATBUS

For a STATbus design that contains STATs, you must refer to the Table 9-1 above to determine the number of devices that can be used in addition to the STATs.

For a system with 4 STATs, for example, Table 9-1 indicates that up to five additional devices can be connected to the bus. You could use four SSB-AO1s to create four zones with damper control.

9.5 GID NUMBERS AND MAPPING IOX MODULES

9.5.1 WRITING GIDS TO DEVICES

Every IOX module has a unique Global Identification (GID) which is identical to the unit's serial number. The GID number is the address that the controller will use to uniquely identify and communicate with the module. The GID numbers are used during commissioning to map the remote I/O point(s) on the IOX modules to inputs and/or outputs in the controller.

NOTE

It is recommended that, at some point before or during installation, you compile a list of all the STATbus devices, their location or function, and their GID numbers. This information will be needed in the commissioning of the project and may be difficult to obtain once the units are installed.

To prepare the controller to perform this mapping you must set the **(CR) Configure Remote I/O** property in the STATBus Summary object to "Edit I/O GIDs" (**CR**=2). This allows you to manually assign the GID numbers of remote I/O devices located on the STATbus to the inputs or outputs in the controller.

You must select an input or output in the controller and choose the device you wish to assign to it. You must then enter the GID number of the chosen device in to the **(GI) GID of I/O Device** property for the input or output. If the GID entered is valid, the GID number will be displayed, along with the type of device.

Once **(GI) GID of I/O Device** has been set, you must then configure the index number properties **(I#) Input Index of I/O Device** for input objects, and **(O#) Output Index for I/O Device** for output objects. For IOX modules that only contain a single I/O point, the index number shall be se to a value of 1. For IOX modules that contain multiple points of data, you will need to set the index number according to the I/O assignment. For example, if you wish to configure a Universal Input to focus on UI2 of an SSB-IOX, set the index number to a value of 2. Each IOX module reviewed will include a table that provides a reference for the index number that corresponds with each input or output.

Repeat this process of assigning GIDs for as many devices as you wish to configure.

Once all the devices you wish to configure have had their GIDs successfully assigned to an input or output, you must set the **(CR) Configure Remote I/O** = 1. This will write the configuration information to the devices on the STATbus network. The Configure Remote I/O setting will automatically return to "Normal" (**CR**=0) once the write is complete, eliminating the possibility of accidentally overwriting STATbus configuration information.

9.5.2 REMOVING GID ASSIGNMENTS

Once the GID of an IOX module has been mapped to a particular input or output, that mapping is stored both in the controller and on the device itself. If you wish to remove a GID mapping, you can do so by entering a value of 0 into the **GI** property or attribute for the input or output. Unmapping a module in this way will only remove the assignment to that particular input or output and will not effect any other inputs or outputs to which the module is mapped. If the module has multiple inputs or outputs, this will leave all other mappings intact. This situation will cause communication problems between the controller and the module

that will result in the module behaving unpredictably. Additionally, **I#** and **O#** can be set to a value of 255 to un assigning the device.

NOTE
 When unmapping the GID of a module with multiple inputs and/or outputs, you must zero the GID in all objects or channels to which it is assigned.

9.6 SSB-FI1

9.6.1 FEATURES

The SSB-FI1, shown in Figure 9-2, is a STATbus device which provides a single remote Flexible Input, that is installed at the sensor location - in all most cases, directly coupled with the sensor. A Flexible Input is a lower resolution version of the Universal Input found on a device such as an NB-GPC controller. The signal read by the SSB-FI1 is processed using a 12-bit analog-to-digital converter and, through digital signal processing algorithms, extrapolated to a 16-bit reading.

The SSB-FI1 is an option for sensors which do not require excitation power or for inputs which do not require the additional resolution provided by the SSB-UI1.

CAUTION

The design of the SSB-FI1 is intended for installation scenarios where the FI1 module will be mounted directly against or near the sensor. This device is not intended for scenarios where it is mounted in an enclosure panel with sensor wire "home-runned" similar to a physical input. If you wish to perform this type of applicaiton, you must use an SSB-UI1. Failure to follow this guideline can result in poor STATbus network communicaitons.

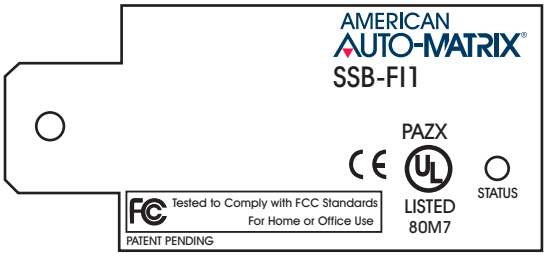

Figure 9-2: The SSB-FI1

The input on the SSB-FI1 can be configured to read a 0-10 V, 0-5 V, 0-20 mA or 0-250 kΩ signal using jumpers located on the device. Any of the jumper configurations can also be interpreted by the controller as a digital value by setting the sensor type to digital (**ST**=0). When being interpreted as a digital signal, a reading of 0-25% of full scale corresponds to a zero state and a reading of 26-100% of full scale corresponds to a one state.

9.6.2 WIRING/CONFIGURATION

9.6.2.1 IVR JUMPER

Before installing the SSB-FI1, you must configure the SSB-FI1 for the type of sensor connected to it, connect the sensor to be used, and connect the SSB-FI1 to the STATbus network. The jumpers used to determine the type of sensor connected to the SSB-FI1 are shown in Figure 9-3.

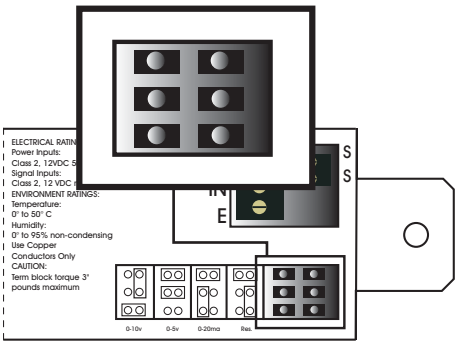

Figure 9-3: Location of the Input Select Jumpers

Once the type of sensor that will be connected to the SSB-FI1 has been determined, the jumpers should be moved to the positions appropriate for that type. The jumper settings for a 0-10 V, 0-5 V, 0-20 mA and 0-250 kΩ resistive inputs are given in Figure 9-4a-d respectively. The jumper configurations are also printed on the SSB-FI1 enclosure adjacent to the jumpers.

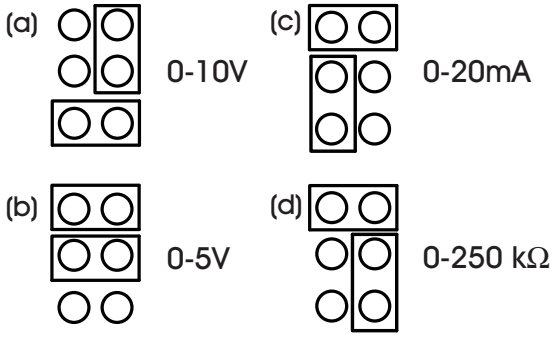

Figure 9-4: Jumper Settings for the SSB-FI1

9.6.2.2 INPUT WIRING

The SSB-FI1 is ideal for any sensor which does not require power. A voltage sensor (0-10 V or 0-5V), current sensor (0-20 mA or 4-20 mA), or resistance sensor would all be wired to the SSB-FI1 by connecting the common wire to the COM terminal and the signal wire to the IN terminal as shown in Figure 9-5.

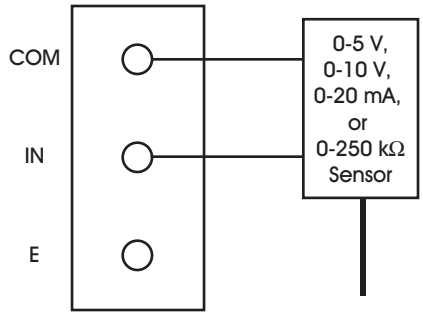

Figure 9-5: Wiring an Input to the SSB-FI1

NOTE

The COM terminal on the SSB-FI1 is not an electrical ground. The SSB-FI1 will only function properly when connected to isolated, non-grounded sensors.

9.6.2.3 POSITION POTENTIOMETER

The SSB-FI1 can also be used to read information from a position potentiometer. For this type of sensor, the jumpers must be set so that the SSB-FI1 is configured as a voltage input (either 0-5 V or 0-10 V). One side of the resistor should be connected to the COM terminal and the other side to the E terminal. The wiper, providing the position information, is connected to the IN terminal as shown in Figure 9-6.

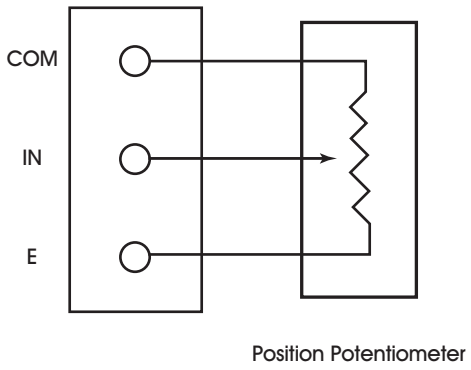

Figure 9-6:Wiring a Position Potentiometer to the SSB-FI1

Once the sensor is connected to the SSB-FI1, it must be added to the STATbus network. Connect the pair of wires coming from the last STATbus device to the two-pin terminal block labelled SS on the SSB-FI1. The STATbus network is non-polar, so you do not need to worry about maintaining polarity between devices.

9.6.3 MOUNTING THE SSB-FI1

The SSB-FI1 is designed to be mounted inside a standard 2x4 junction box as shown in Figure 9-7. The SSB-FI1 is mounted in the junction box by attaching a screw to the junction box, through the mounting hole, securing the SSB-FI1. The SSB-FI1 should be mounted such that the terminal blocks face the inside of the junction box. When the SSB-FI1 is correctly installed, the GID number printed on the label should be clearly visible.

NOTE

All connections should be made to the SSB-FI1 before it is mounted. Before final mounting, make sure that all wires are securely seated in the terminal block plugs and the that the terminal block plugs are firmly inserted in the correct socket.

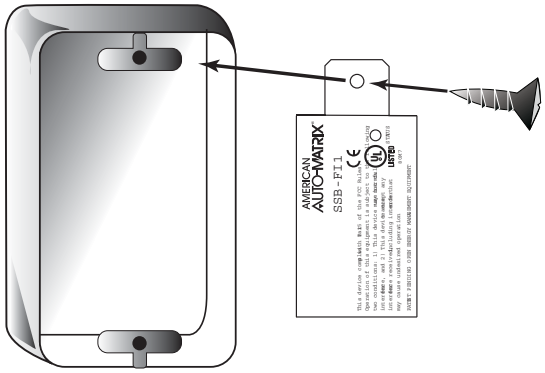

Figure 9-7: Mounting the SSB-FI1

9.6.4 STATUS INDICATOR LED

The SSB-FI1 has a status indicator LED which provides feedback as to the device's current operational status. The status indicator LED is located on the front of the SSB-FI1 (the side that faces out when installed) as shown in Figure 9-8. This allows status diagnostics to be performed without having to remove the SSB-FI1 or disconnect it from the STATbus.

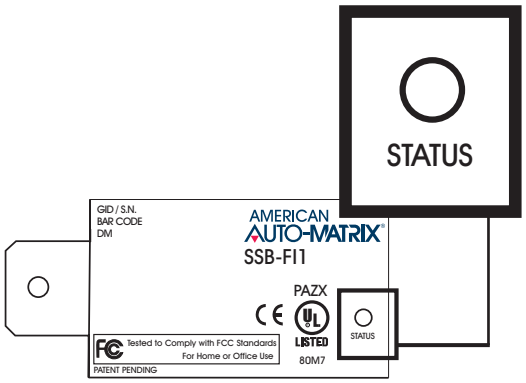

Figure 9-8: Location of the Status Indicator LED on the SSB-FI1

The status indicator LED shows one of four different states: powered but not enumerated, enumerated but not configured, configured, and “identify”. The different states are indicated by the rate at which the LED blinks. The LED blinking quickly, approximately three to four times per second, indicates that the unit is powered but has not yet been enumerated by the controller. This is useful for identifying units that are correctly wired but not configured. When the device is enumerated but not configured, the blink rate will slow to approximately twice a second.

Once the device has been configured, the blink rate will slow down to approximately one blink per second. This will be the normal state of the device when it is correctly wired, powered, enumerated, and configured in the controller.

When the controller is set to “Identify” in the Configure Function and Configure Device properties, the LED on the SSB-FI1 will be blink three times in quick succession and then pause before repeating the three blinks again. This is especially useful for quickly identifying an individual device in the field when troubleshooting the STATbus.

9.6.5 SSB-FI CONFIGURATION TABLE

Table 9-2: SSB-FI Configuration Table

I/O Quantity	**(I#)** Index Number
1 Universal Input	1

9.7 SSB-UI1

9.7.1 FEATURES

The SSB-UI1, shown in Figure 9-9, is a STATbus device which provides a single remote universal input. The input the SSB-UI1 provides is a true 24-bit universal input with more robust signal processing electronics than the SSB-FI1. The SSB-UI1 should be used when using an input which requires excitation power. The SSB-FI1, while capable of reading the signal from such devices, albeit at a lower resolution, is not capable of providing excitation power, so it is not a viable choice when dealing with sensors of this kind.

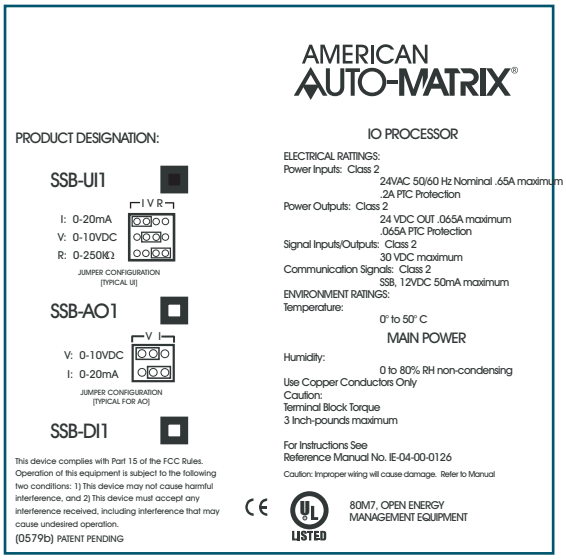

Figure 9-9: The SSB-UI1

9.7.2 WIRING/CONFIGURATION

Connecting a sensor to the SSB-UI1 requires two steps, connecting the wires from the sensor to the SSB-UI1 and configuring the IVR jumper.

Connections are made to the SSB-UI1 via the terminal blocks, located on the side of the unit which faces into the junction box, as shown in Figure 9-10. There are two sets of terminal blocks, the first is for connection to the STATbus network and power and the second is for the connection of the sensor to the SSB-UI1.

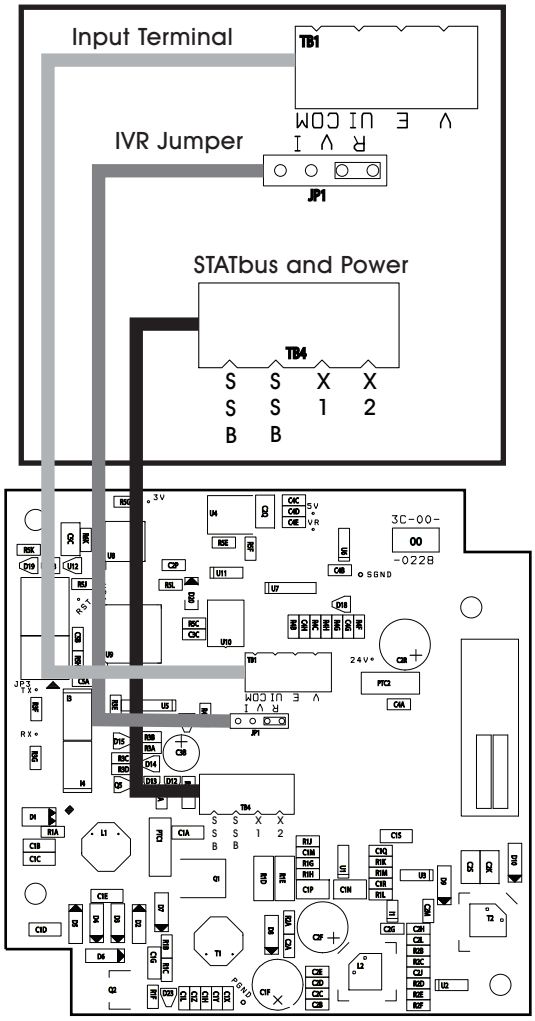

Figure 9-10: SSB-UI1 Terminal Block Locations

9.7.2.1 IVR JUMPER

The SSB-UI1 has an IVR jumper, identical in function to the ones found on the NB-GPC, which is used to select the type of input connected to the module. The SSB-UI1 can be configured to read a 0-20 mA, 0-10 V or a 0-250 kΩ sensor. The jumper settings corresponding to these options are shown in Figure 9-11a-c respectively.

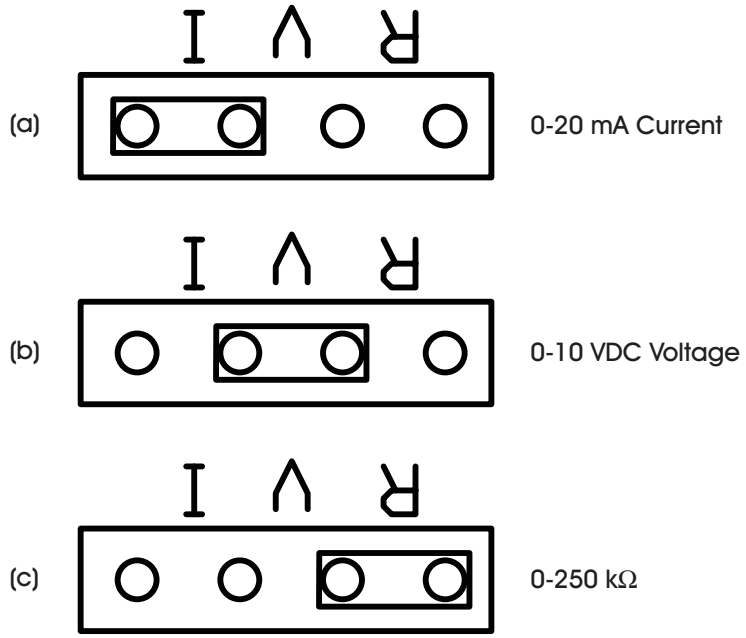

Figure 9-11: SSB-UI1 IVR Jumper Positions

9.7.2.2 RESISTIVE INPUTS

A resistive input, such as a thermistor, would be connected as shown in Figure 9-12. One side of the input should be connected to the UI terminal and the other to the COM terminal. Since a thermistor is a resistive input, the IVR jumper should be set to the “R” position.

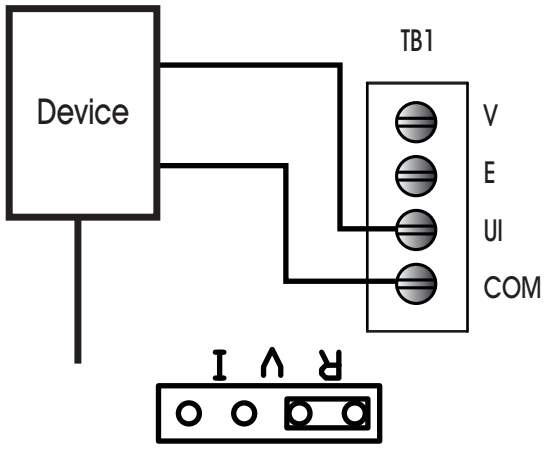

Figure 9-12: Wiring a Resistive Input to the SSB-UI1

9.7.2.3 VOLTAGE INPUTS

The connections needed to use a voltage sensor with the SSB-UI1 are shown in Figure 9-13. The signal wire from the sensor should be connected to the UI terminal, the power connection should be connected to

the V terminal and the common wire should be connected to the COM, terminal V terminal provides 24VDC output. The IVR jumper should be set to the “V” position when using this type of input.

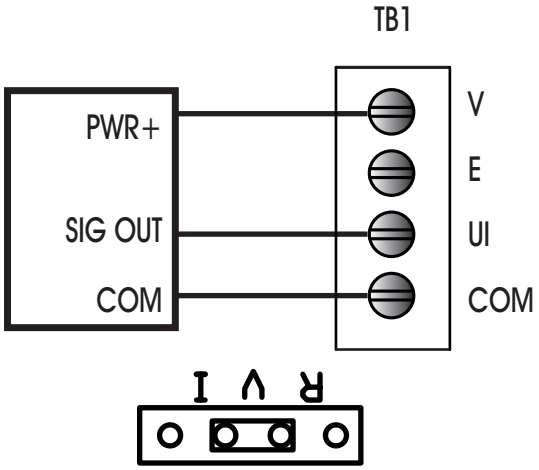

Figure 9-13: Wiring a Voltage Input to the SSB-UI1

9.7.2.4 CURRENT INPUTS

When using a current sensor with the SSB-UI1, the sensor would be connected as shown in Figure 9-14. The + and - terminals of the sensor should be connected to the V and UI terminals on the SSB-UI1 respectively. When using 3-wire current sensors, the COM terminal on the sensor should be connected to the COM terminal on the SSB-UI1. Regardless of which type of current sensor you are using, the IVR jumper should be set to the “I” position.

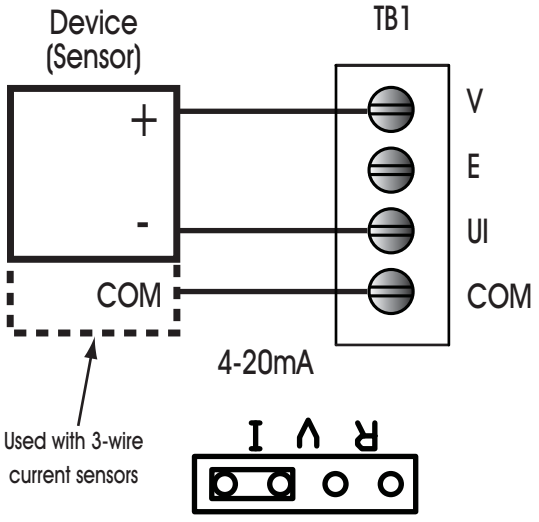

Figure 9-14: Wiring a Current Input to the SSB-UI1

9.7.2.5 POSITION POTENTIOMETER

The SSB-UI1 can be configured to read the signal from a position potentiometer as shown in Figure 9-15. For this type of sensor, the one side of the resistor should be connected to the COM terminal, the other side should be connected to the E terminal, and the wiper, providing the position information, should be connected to the UI terminal. When using this type of sensor, the IVR jumper should be set to the "V" position.

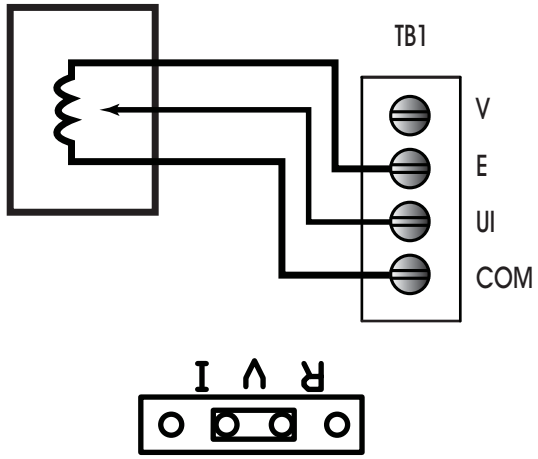

Figure 9-15: Wiring a Position Potentiometer to the SSB-UI1

9.7.3 MOUNTING THE SSB-UI1

The SSB-UI1 has the same footprint as, and is designed to be mounted on top of, a standard 4x4 junction box, replacing the junction box's cover plate.

Before mounting the SSB-UI1 to the junction box, verify all wiring is correct, making sure that all screw terminals are sufficiently tightened and all terminal blocks are securely seated.

With the wires attached to the device, loosen the screws on the 4x4 junction box slightly. The screws should be loose enough to provide room to slide the SSB-UI1 onto the screws, but not so loose that they can fall out. Align the channels on the back corners of the SSB-UI1 with the screws on the junction box and slide the unit downward onto the screws as shown in Figure 9-16. Tighten the screws through the holes in the front of the SSB-UI1 to secure the device to the junction box.

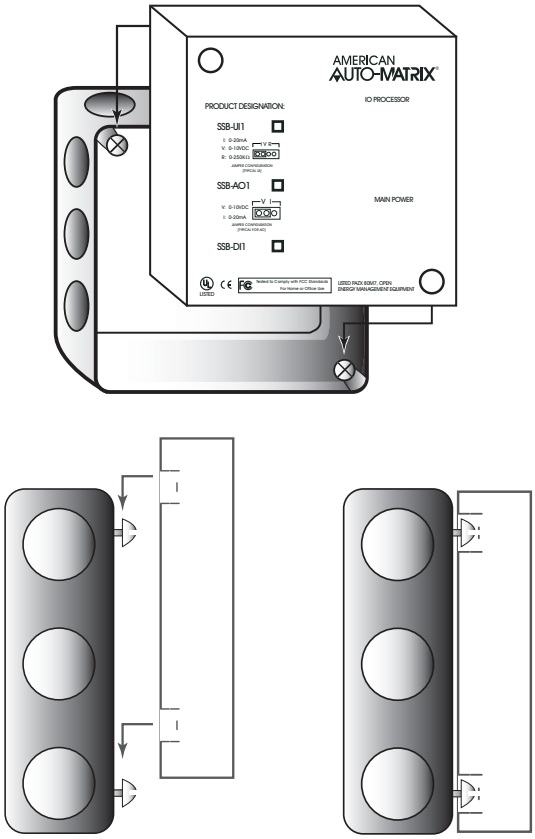

Figure 9-16: Mounting the SSB-UI1 to a 4x4 junction box

9.7.4 STATUS INDICATOR LED

The SSB-UI1 has an IO Processor indicator LED which provides feedback as to the device's current operational status. The IO Processor indicator LED is located on the front of the SSB-UI1 (the side that faces out when installed) as shown in Figure 9-17. This allows status diagnostics to be performed without having to remove the SSB-UI1.

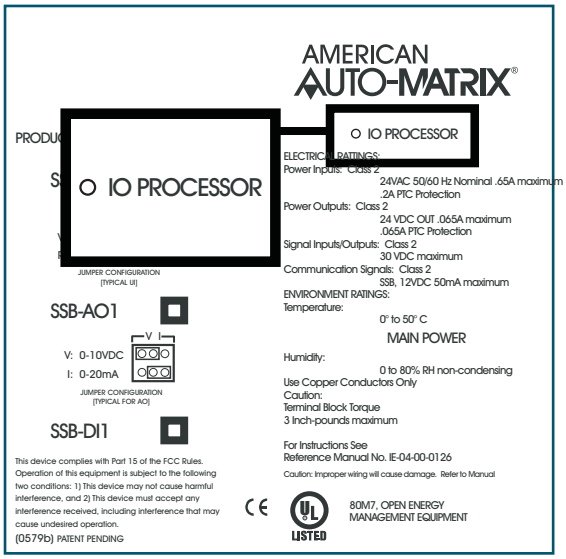

Figure 9-17: Location of the IO Processor Indicator LED on the SSB-UI1

The status indicator LED shows one of four different states: powered but not enumerated, enumerated but not configured, configured, and “identify”. The different states are indicated by the rate at which the LED blinks. The LED blinking quickly, approximately three to four times per second, indicates that the unit is powered but has not yet been enumerated by the controller. This is useful for identifying units that are correctly wired but not configured. When the device is enumerated but not configured, the blink rate will slow to approximately twice a second.

Once the device has been configured, the blink rate will slow down to approximately one blink per second. This will be the normal state of the device when it is correctly wired, powered, enumerated, and configured in the controller.

When the controller is set to “Identify” in the Configure Function and Configure Device properties, the LED on the SSB-UI1 will be blink three times in quick succession and then pause before repeating the three blinks again. This is especially useful for quickly identifying an individual device in the field when troubleshooting the STATbus.

9.7.5 SSB-UI CONFIGURATION TABLE

Table 9-3: SSB-UI Configuration Table

I/O Quantity	**(I#)** Index Number
1 Universal Input	1

9.8 SSB-AO1

9.8.1 FEATURES

The SSB-AO1, shown in Figure 9-18, is a STATbus device which provides a single remote analog output to the MatrixBBC. This output can be configured as either a 0-10 VDC or 0-20 mA output via a user-selectable jumper.

Figure 9-18: The SSB-AO1

9.8.2 WIRING/CONFIGURATION

Connections are made to the SSB-AO1 via the terminal blocks shown in Figure 9-19. The terminal blocks are located on the side of the unit which faces the junction box. There are two terminal blocks, the first is for connections to the STATbus network and power and the second is for the connection to the output device.

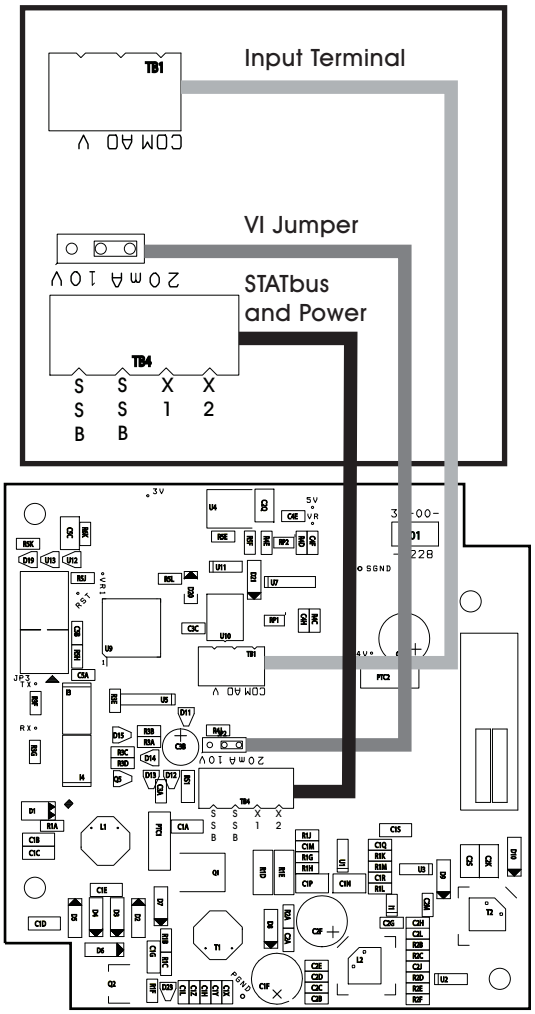

Figure 9-19:SSB-AO1 Terminal Block Locations

Each SSB-AO1 has a VI jumper identical to the ones found on the NB-GPC. This is used to select the output range of the SSB-AO1. The output can be configured for 0-10 VDC or 0-20 mA operation, using the jumper positions shown in Figure 9-20a and b respectively.

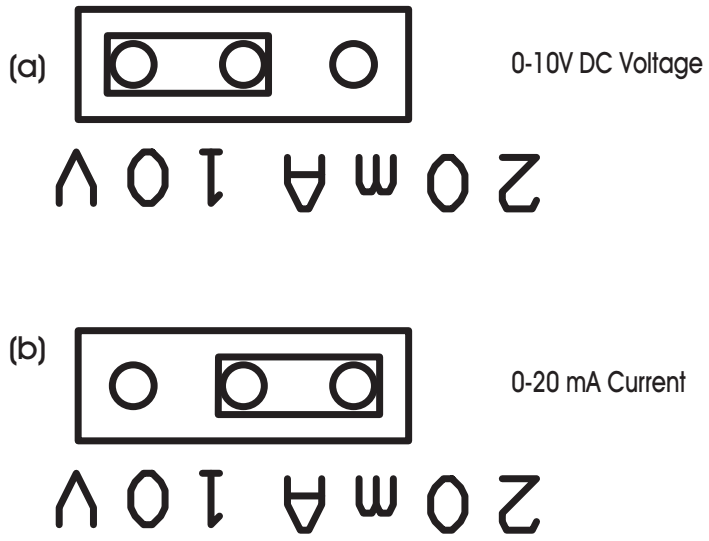

Figure 9-20: SSB-AO1 VI Jumper Positions

9.8.2.1 OUTPUT WIRING

When using devices which do not require power, either because they do not require a power supply or because they have a dedicated external power supply, the SSB-AO1 is wired as shown in Figure 9-21. The signal wire should be connected to the AO terminal on the SSB-AO1 and the common wire should be connected to the COM terminal. For a 0-10 V device, the VI jumper should be set to the “V” position. If the device operated on a 0-20 mA signal, you would instead set the VI jumper to the “I” position.

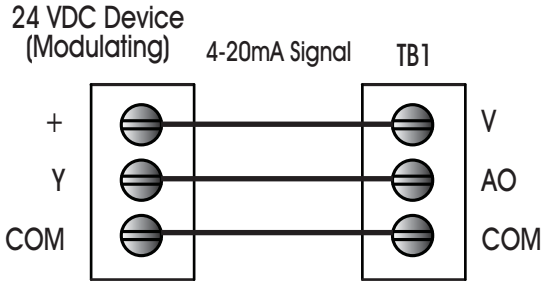

Figure 9-21: Wiring the SSB-AO1 to a Device That Does Not Require Power

9.8.2.2 POWERED DEVICES

When the output device requires power, the SSB-AO1 would be connected as shown in Figure 9-22. The + or power wire from the sensor should be connected to the V terminal on the SSB-AO1. The Y wire should be connected to the AO terminal and the common wire should be connected to the COM terminal. For a 0-10 V device, the VI jumper should be set to the “V” position. If the device operated on a 0-20 mA signal, you would instead set the VI jumper to the “I” position.

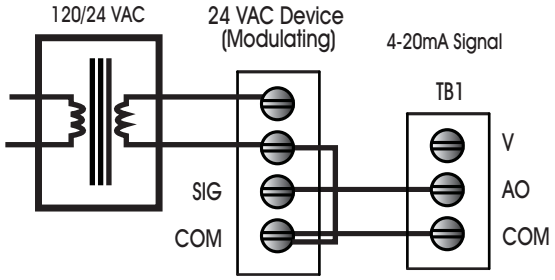

Figure 9-22: Wiring the SSB-AO1 to a Device That Requires Power

9.8.3 MOUNTING THE SSB-AO1

The SSB-AO1 is designed to be mounted on top of, a standard 4x4 junction box, replacing the junction box’s cover plate.

Before mounting the SSB-AO1 to the junction box, verify all wiring is correct, making sure that all screw terminals are sufficiently tightened and all terminal blocks are securely seated.

With the wires attached to the device, loosen the screws on the 4x4 junction box slightly. The screws should be loose enough to provide room to slide the SSB-AO1 onto the screws, but not so loose that they can fall out. Align the channels on the back corners of the SSB-AO1 with the screws on the junction box and slide the unit downward onto the screws as shown in Figure 9-23. Tighten the screws through the holes in the front of the SSB-AO1 to secure the device to the junction box.

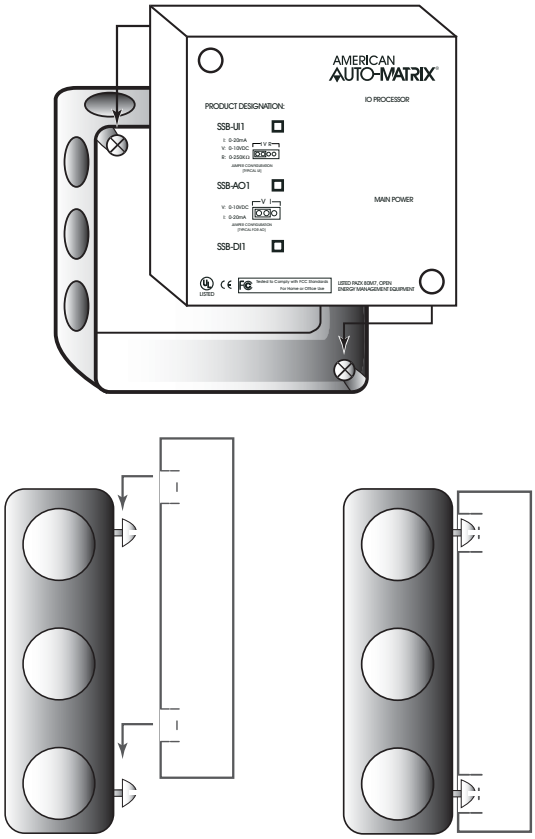

Figure 9-23: Mounting the SSB-AO1 to a 4x4 junction box

9.8.4 STATUS INDICATOR LED

The SSB-AO1 has an IO Processor indicator LED which provides feedback as to the device's current operational status. The IO Processor indicator LED is located on the front of the SSB-AO1 (the side that faces out when installed) as shown in Figure 9-24. This allows status diagnostics to be performed without having to remove the SSB-AO1.

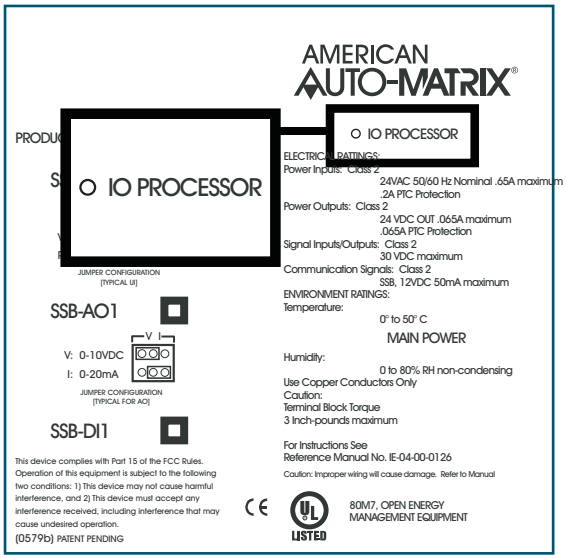

Figure 9-24: Location of the IO Processor Indicator LED on the SSB-AO1

The status indicator LED shows one of four different states: powered but not enumerated, enumerated but not configured, configured, and “identify”. The different states are indicated by the rate at which the LED blinks. The LED blinking quickly, approximately three to four times per second, indicates that the unit is powered but has not yet been enumerated by the controller.

Once the device has been enumerated, the blink rate will slow down to approximately one blink per second. This will be the normal state of the device when it is correctly wired, powered, enumerated, and configured in the controller.

When the controller is set to “Identify” in the Configure Function and Configure Device properties, the LED on the SSB-AO1 will be blink three times in quick succession and then pause before repeating the three blinks again. This is especially useful for quickly identifying an individual device in the field when troubleshooting the STATbus.

9.8.5 SSB-AO CONFIGURATION TABLE

Table 9-4: SSB-FI Configuration Table

I/O Quantity	**(O#)** Index Number
1 Analog Output	1

9.9 SSB-DI1

9.9.1 FEATURES

The SSB-DI1, shown in Figure 9-25, is a STATbus module which provides a single remote digital input to the MatrixBBC. This input is a wet contact that is capable of pulse counting.

Figure 9-25: The SSB-DI1

9.9.2 WIRING/CONFIGURATION

Connections are made to the SSB-DI1 via the terminal blocks shown in Figure 9-26. The terminal blocks are located on the side of the unit which faces the junction box. There are two blocks, the first for connections to the STATbus network and power, and the second for the connection to the input device.

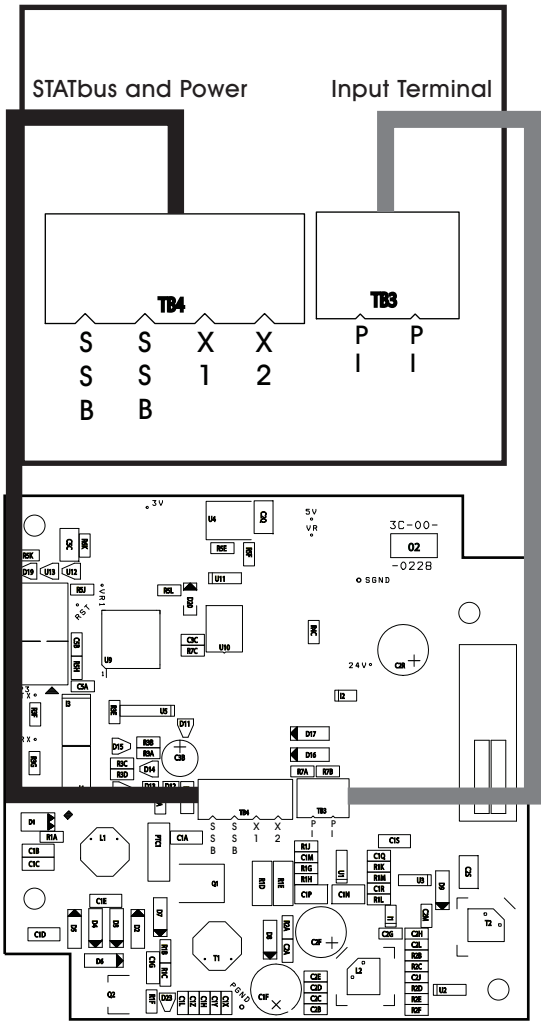

Figure 9-26:SSB-DI1 Terminal Block Locations

When the SSB-DI1 is configured to operate with an internally powered input, it would be wired as shown in Figure 9-27. The two wires from the input should be connected to the to PI terminals on the SSB-DI1.

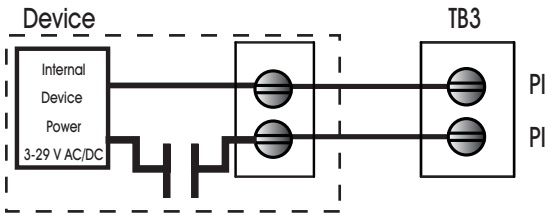

Figure 9-27: Wiring the SSB-DI1 for use with an Internally Powered Input

If the SSB-DI1 is to be used with a externally powered input, the wiring must include a source of power for the input device. As shown in Figure 9-28, the X1 and X2 terminals are connected to one of side of the sensor and one of the PI terminal. The other side of the sensor is wired to the remaining PI terminal.

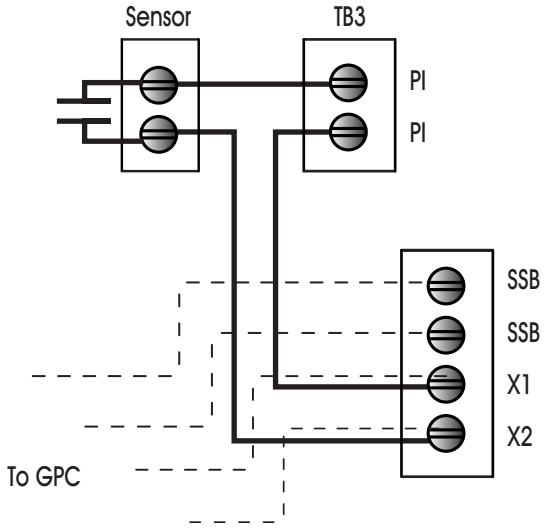

Figure 9-28: Wiring the SSB-DI1 for use with an Externally Powered Input

9.9.3 MOUNTING THE SSB-DI1

The SSB-DI1 has the same footprint and is designed to be mounted on top of a standard 4x4 junction box, replacing the junction box's cover plate.

Before mounting the SSB-DI1 to the junction box, verify all wiring is correct, making sure that all screw terminals are sufficiently tightened and all terminal blocks are securely seated.

WIth the wires attached to the device, loosen the screws on the 4x4 junction box slightly. The screws should be loose enough to provide room to slide the SSB-DI1 onto the screws, but not so loose that they can fall out. Align the channels on the back corners of the SSB-DI1 with the screws on the junction box and slide the unit downward onto the screws as shown in Figure 9-29. Tighten the screws through the holes in the front of the SSB-DI1 to secure the device to the junction box.

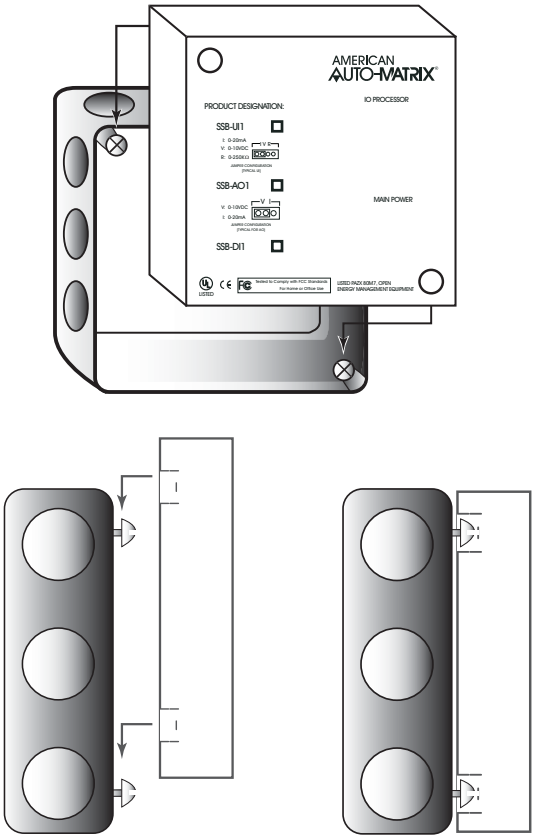

Figure 9-29: Mounting the SSB-DI1 to a 4x4 junction box

9.9.4 STATUS INDICATOR LED

The SSB-DI1 has an IO Processor indicator LED which provides feedback as to the device's current operational status. The IO Processor indicator LED is located on the front of the SSB-DI1 (the side that faces out when installed) as shown in Figure 9-30. This allows status diagnostics to be performed without having to remove the SSB-DI1.

9.9.5 SSB-DI1 CONFIGURATION TABLE

Table 9-5: SSB-FI Configuration Table

I/O Quantity	**(I#)** Index Number
1 Digital Input	1

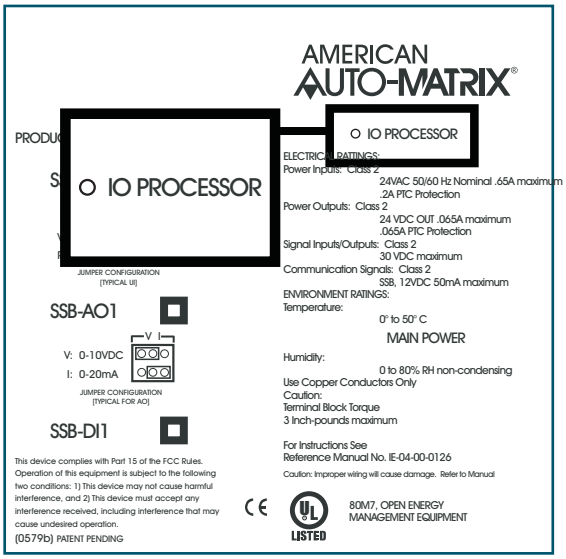

Figure 9-30: Location of the IO Processor Indicator LED on the SSB-DI1

The status indicator LED shows one of four different states: powered but not enumerated, enumerated but not configured, configured, and “identify”. The different states are indicated by the rate at which the LED blinks. The LED blinking quickly, approximately three to four times per second, indicates that the unit is powered but has not yet been enumerated by the controller. This is useful for identifying units that are correctly wired but not configured. When the device is enumerated but not configured, the blink rate will slow to approximately twice a second.

Once the device has been configured, the blink rate will slow down to approximately one blink per second. This will be the normal state of the device when it is correctly wired, powered, enumerated, and configured in the controller.

When the controller is set to “Identify” in the Configure Function and Configure Device properties, the LED on the SSB-ADI1 will be blink three times in quick succession and then pause before repeating the three blinks again. This is especially useful for quickly identifying an individual device in the field when troubleshooting the STATbus.

9.10 SSB-DO1

9.10.1 FEATURES

The SSB-DO1, shown in Figure 9-31, is a STATbus module which provides a single remote digital output to the MatrixBBC. The digital output on the SSB-DO1 is a relay capable of switching up to 250 VAC/DC at up to 10 A.

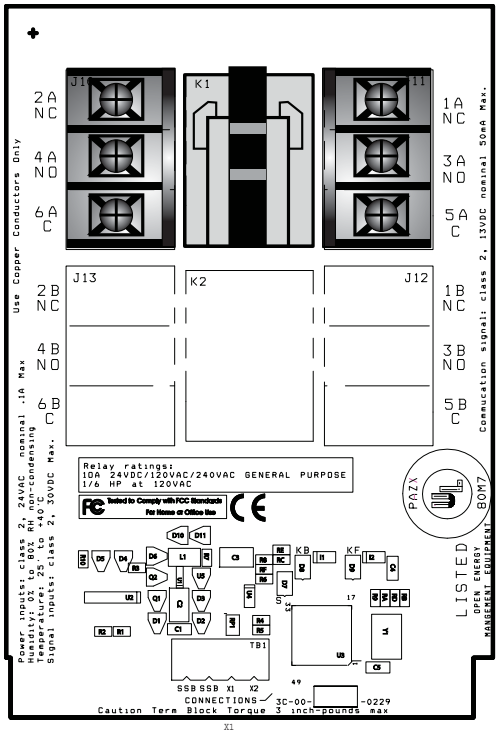

Figure 9-31: The SSB-DO1

9.10.2 MOUNTING THE SSB-DO1

The SSB-DO1 is intended to be installed near the equipment it is going to control. The module is designed to be mounted in the snap-in plastic track included with the module.

Before installing the module, the snap-in plastic track should be attached to the mounting surface using the screws provided. Once the snap-in plastic track has been firmly attached, the SSB-DO1 should be installed by first inserting one edge of the module in the channel on the inside of one side of the rail and then pressing the module so that it snaps into the other side. This is shown in Figure 9-32.

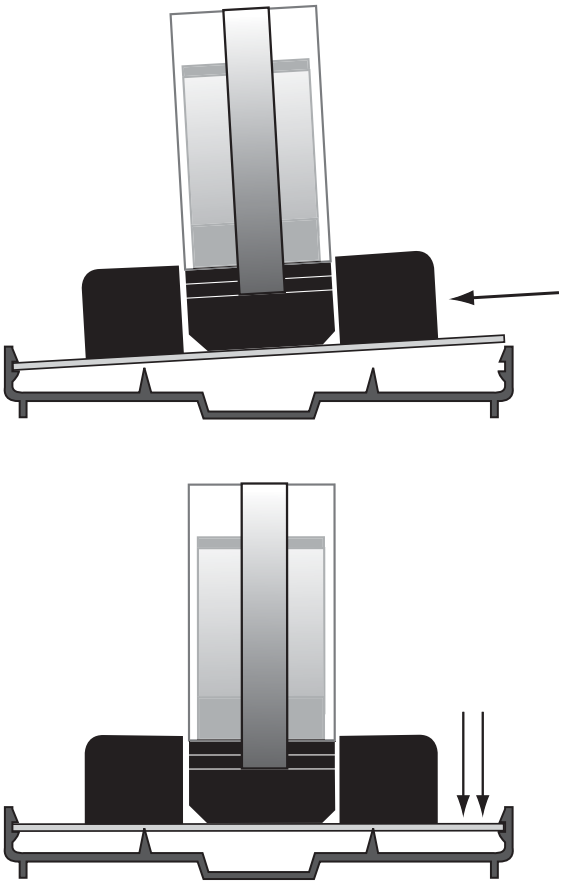

Figure 9-32: Attaching the SSB-DO1 to the Snap-in Plastic Track

WARNING

To avoid damage to the module, you should not attempt to slide the module into the snap-in plastic track.

Once the module has been properly mounted, you may make the connections for STATbus and power.

9.10.3 WIRING/CONFIGURATION

9.10.3.1 POWER AND STATBUS CONNECTIONS

Connections for power and STATbus communications are made via terminal block TB1, shown in Figure 9-33. The two wire connection for STATbus communications should be connected to the two left most terminals labeled “SSB”. A source of 24VAC should be connected to the right two terminals labeled “X1” and “X2”. The easiest way to connect these terminals is to connect them to the AC OUT terminals on the same terminal block as the SSB connection on the controller. This allows you to simply run a 4-conductor

cable to connect both power and STATbus communications to the device without any additional wiring. Alternately, a dedicated external power supply may be provided for the module.

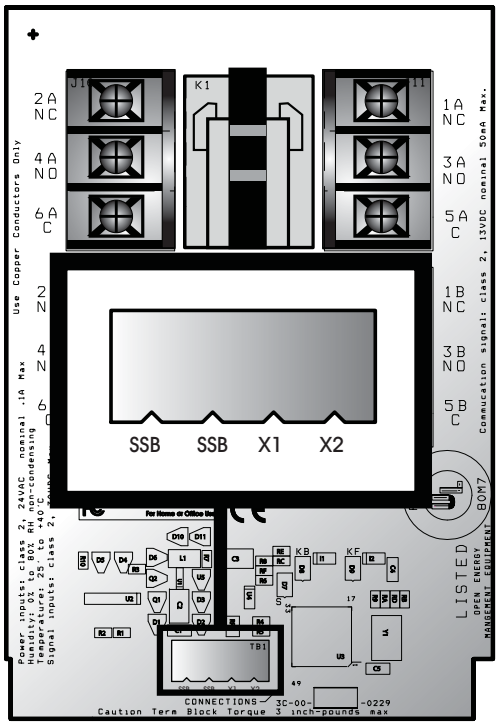

Figure 9-33: SSB and Power Connections on the SSB-DO1

9.10.3.2 RELAY CONNECTIONS

The SSB-DO1 has a single DPDT relay for its output. When the output is energized, the connection is made between the normally open terminal, labeled 3A and 4A, and the common terminals, labeled 5A and 6A. The normally open terminals are labeled “NO” below the terminal number and the common terminals are labeled “C”. When the output is not energized, connections will be made between the normally closed terminals, labeled 1A and 2A, and the common terminals. Like the normally open terminals, the normally closed terminals are labeled “NC” below the terminal number. The function of each of the relay terminals is summarized in Table 9-6:.

Table 9-6: SSB-DO1 Relay Terminals

Terminal	Connection
1A	Normally Closed (NC)
2A	Normally Closed (NC)
3A	Normally Open (NO)
4A	Normally Open (NO)
5A	Common
6A	Common

9.10.4 SSB-DO1 CONFIGURATION TABLE

Table 9-7: SSB-FI Configuration Table

I/O Quantity	**(O#)** Index Number
1 Digital Output	1

9.11 SSB-DO1-I

9.11.1 FEATURES

The SSB-DO1-I, shown in Figure 9-34, is identical to the SSB-DO1 except that it includes a single dry contact, digital input. The digital output on the SSB-DO1-I is a relay capable of switching up to 250 VAC/ DC at up to 10 A. The digital input is a dry contact which is intended for status monitoring of the output state of the relay. Connecting a wet contact to this input will result in damage to the SSB-DO1-I.

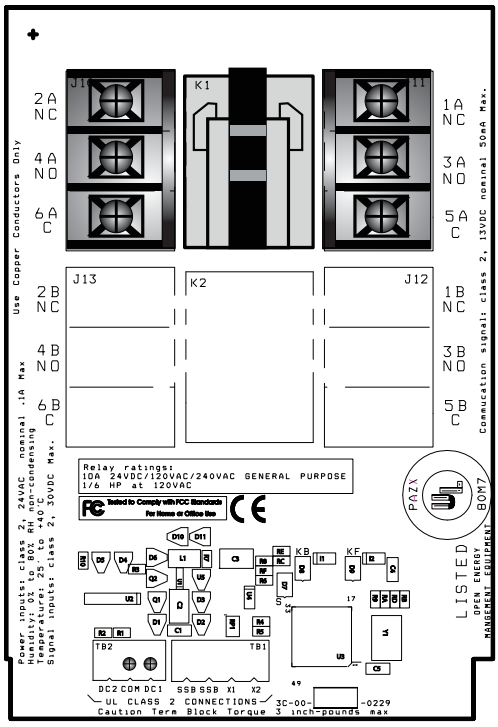

Figure 9-34: The SSB-DO1-I

9.11.2 MOUNTING THE SSB-DO1-I

The SSB-DO1-I is intended to be installed near the equipment it is going to control. The module is designed to be mounted in the snap-in plastic track included with the module.

Before installing the module, the snap-in plastic track should be attached to the mounting surface using the screws provided. Once the snap-in plastic track has been firmly attached, the SSB-DO1-I should be installed by first inserting one edge of the module in the channel on the inside of one side of the rail and then pressing the module so that it snaps into the other side. This is shown in Figure 9-35.

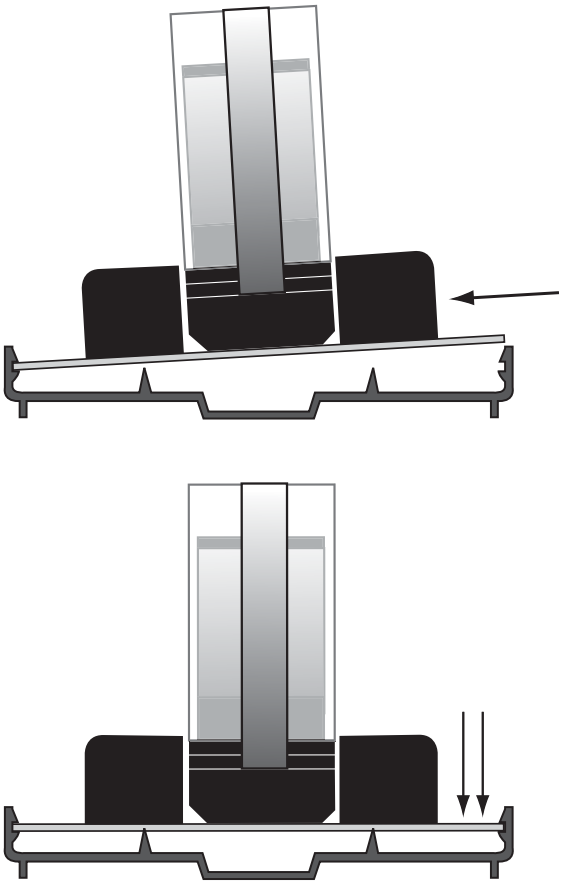

Figure 9-35: Attaching the SSB-DO1-I to the Snap-in Plastic Track

WARNING

To avoid damage to the module, you should not attempt to slide the module into the snap-in plastic track.

Once the module has been properly mounted, you may make the connections for STATbus and power.

9.11.3 WIRING/CONFIGURATION

9.11.3.1 POWER AND STATBUS CONNECTIONS

Connections for power and STATbus communications are made via terminal block TB1, shown in Figure 9-36. The two wire connection for STATbus communications should be connected to the two left most terminals labeled “SSB”. A source of 24VAC should be connected to the right two terminals labeled “X1” and “X2”. The easiest way to connect these terminals is to connect them to the AC OUT terminals on the same terminal block as the SSB connection on the controller. This allows you to simply run a 4-conductor cable to connect both power and STATbus communications to the device without any additional wiring. Alternately, a dedicated external power supply may be provided for the module.

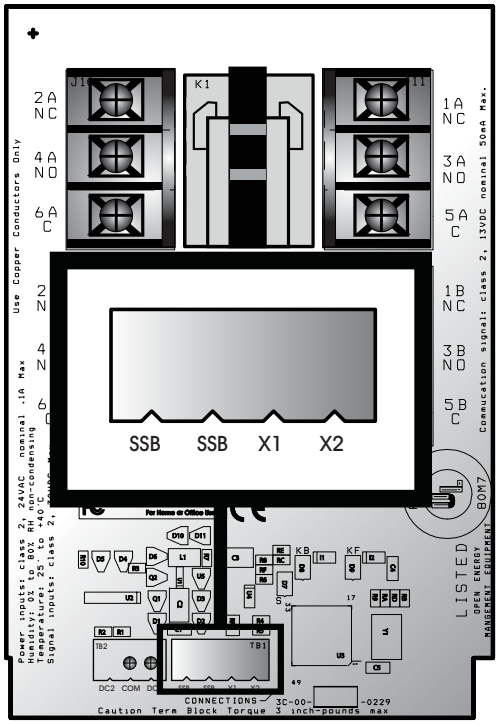

Figure 9-36: SSB and Power Connections on the SSB-DO1-I

9.11.3.2 RELAY CONNECTIONS

The SSB-DO1-I has a single DPDT relay for its output. When the output is energized, the connection is made between the normally open terminal, labeled 3A and 4A, and the common terminals, labeled 5A and 6A. The normally open terminals are labeled “NO” below the terminal number and the common terminals are labeled “C”. When the output is not energized, connections will be made between the normally closed terminals, labeled 1A and 2A, and the common terminals. Like the normally open terminals, the normally closed terminals are labeled “NC” below the terminal number. The function of each of the relay terminals is summarized in Table 9-8:.

Table 9-8: SSB-DO1-I Relay Terminals

Terminal	Connection
1A	Normally Closed (NC)
2A	Normally Closed (NC)
3A	Normally Open (NO)
4A	Normally Open (NO)
5A	Common
6A	Common

9.11.3.3 DRY CONTACT INPUT CONNECTION

The SSB-DO1-I has a single dry contact digital input intended for use as a status monitor for the relay on the module. Connecting a wet contact to this input will result in damage to the SSB-DO1-I. This input can only be assigned to a Digital Input in the controller. Unlike the on-board digital input found on an NB-GPC, the dry contact input on the SSB-DO1-I is not capable of performing pulse counting. Connections for the input are via the COM and DC1 terminals on terminal block TB2 as shown in Figure 9-37.

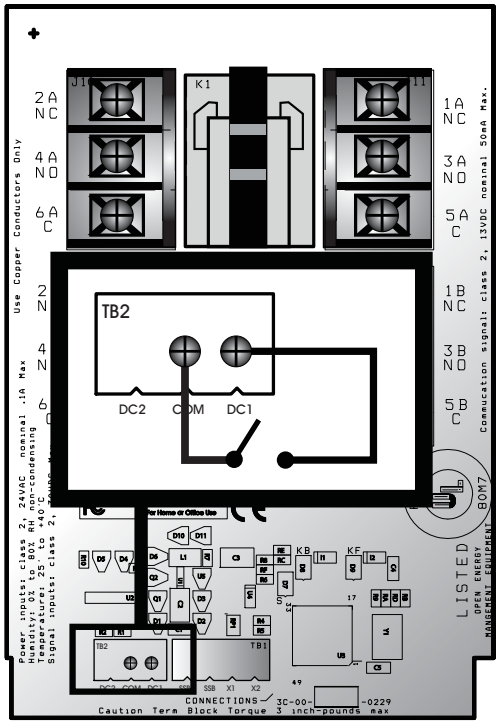

Figure 9-37: Connecting a Dry Contact Input to the SSB-DO1-I

CAUTION

Assigning the SSB-DO1-I's GID number to the **GI,** and **I#** properties of an input only assigns the input on the module. The output will not be assigned unless the GID number and **O#** property are also assigned to a Digital Output in the controller.

9.11.4 SSB-DO1-I CONFIGURATION TABLE

Table 9-9: SSB-DO1-I Configuration Table

I/O Termination	**(I# or O#)** Index Number
Digital Output 1	1
Digital Input 1	1

9.12 SSB-DO2

9.12.1 FEATURES

The SSB-DO2, shown in Figure 9-38, is a STATbus module which provides two remote digital outputs to the MatrixBBC. The digital outputs on the SSB-DO2 are relays capable of switching up to 250 VAC/DC at up to 10 A each.

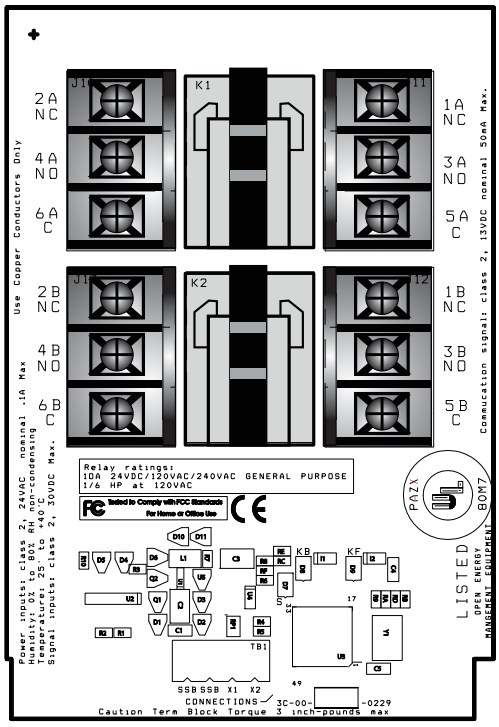

Figure 9-38: The SSB-DO2

9.12.2 MOUNTING THE SSB-DO2

The SSB-DO2 is intended to be installed near the equipment it is going to control. The module is designed to be mounted in the snap-in plastic track included with the module.

Before installing the module, the snap-in plastic track should be attached to the mounting surface using the screws provided. Once the snap-in plastic track has been firmly attached, the SSB-DO2 should be installed by first inserting one edge of the module in the channel on the inside of one side of the rail and then pressing the module so that it snaps into the other side. This is shown in Figure 9-39.

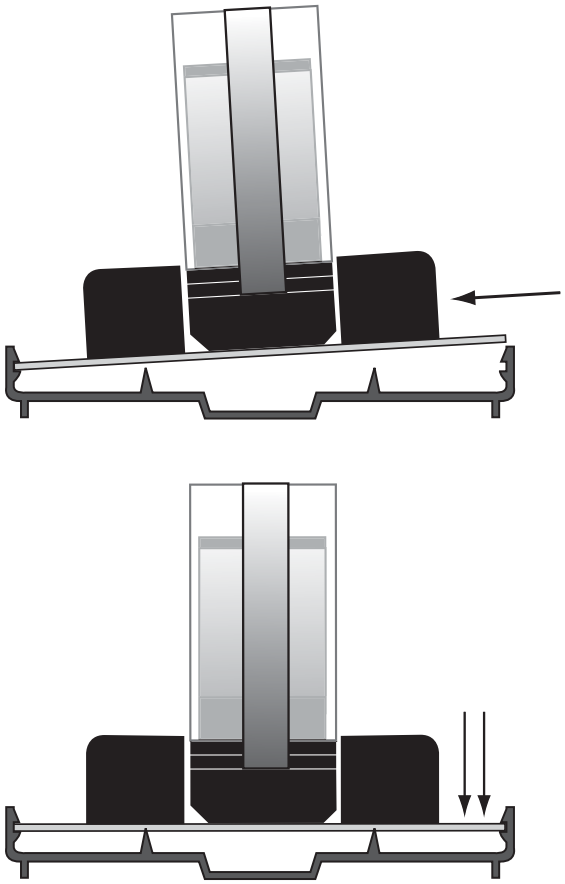

Figure 9-39: Attaching the SSB-DO2 to the Snap-in Plastic Track

WARNING

To avoid damage to the module, you should not attempt to slide the module into the snap-in plastic track.

Once the module has been properly mounted, you may make the connections for STATbus and power.

9.12.3 WIRING/CONFIGURATION

9.12.3.1 POWER AND STATBUS CONNECTIONS

Connections for power and STATbus communications are made via terminal block TB1, shown in Figure 9-40. The two wire connection for STATbus communications should be connected to the two left most terminals labeled “SSB”. A source of 24VAC should be connected to the right two terminals labeled “X1” and “X2”. The easiest way to connect these terminals is to connect them to the AC OUT terminals on the same terminal block as the SSB connection on the controller. This allows you to simply run a 4-conductor cable to connect both power and STATbus communications to the device without any additional wiring. Alternately, a dedicated external power supply may be provided for the module.

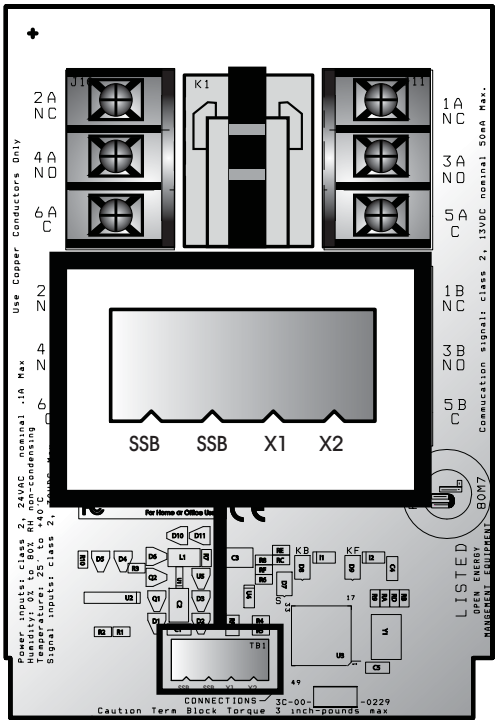

Figure 9-40: SSB and Power Connections on the SSB-DO2

9.12.3.2 RELAY CONNECTIONS

The SSB-DO2 has two (2) DPDT relays for its outputs. When output 1 is energized, the connection is made between the normally open terminal, labeled 3A and 4A, and the common terminals, labeled 5 A and 6A. When the coil for output 2 is energized, the connection is made between the normally open terminal, labeled 3B and 4B, and the common terminals, labeled 5B and 6B. The normally open terminals are labeled “NO” below the terminal number and the common terminals are labeled “C”. When the output 1 is not energized connections will be made between the normally closed terminals, labeled 1A and 2A, and the common terminals. SImilarly, when output 2 is no energized, connections will be made between the normally closed terminals, labeled 1B and 2B, and the common terminals. Like the normally open terminals, the normally closed terminals are labeled “NC” below the terminal number. The function of each of the relay terminals is summarized in Table 9-10:.

Table 9-10: SSB-DO2 Relay Terminals

Terminal	Connection
1A	Normally Closed (NC)
2A	Normally Closed (NC)
3A	Normally Open (NO)
4A	Normally Open (NO)
5A	Common

Table 9-10: SSB-DO2 Relay Terminals

Terminal	Connection
6A	Common
1B	Normally Closed (NC)
2B	Normally Closed (NC)
3B	Normally Open (NO)
4B	Normally Open (NO)
5B	Common
6B	Common

9.12.4 SSB-DO2 CONFIGURATION TABLE

Table 9-11: SSB-DO2 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Digital Output 1	1
Digital Output 2	2

9.13 SSB-DO2-I

9.13.1 FEATURES

The SSB-DO2-I, shown in Figure 9-41, is identical to the SSB-DO2 except that it includes two dry contact, digital inputs for device status monitoring. The digital inputs are dry contacts which are intended for status monitoring of the output states of the relays. Connecting wet contacts to these inputs will result in damage to the SSB-DO2-I. These inputs are mapped to Digital Inputs objects, but are not capable of performing pulse counting.

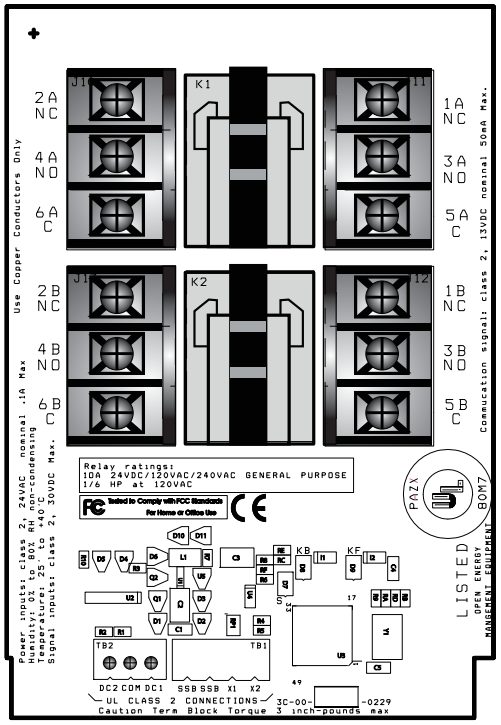

Figure 9-41: The SSB-DO2-I

9.13.2 MOUNTING THE SSB-DO2-I

The SSB-DO2-I is intended to be installed near the equipment it is going to control. The module is designed to be mounted in the snap-in plastic track included with the module.

Before installing the module, the snap-in plastic track should be attached to the mounting surface using the screws provided. Once the snap-in plastic track has been firmly attached, the SSB-DO2-I should be installed by first inserting one edge of the module in the channel on the inside of one side of the rail and then pressing the module so that it snaps into the other side. This is shown in Figure 9-42.

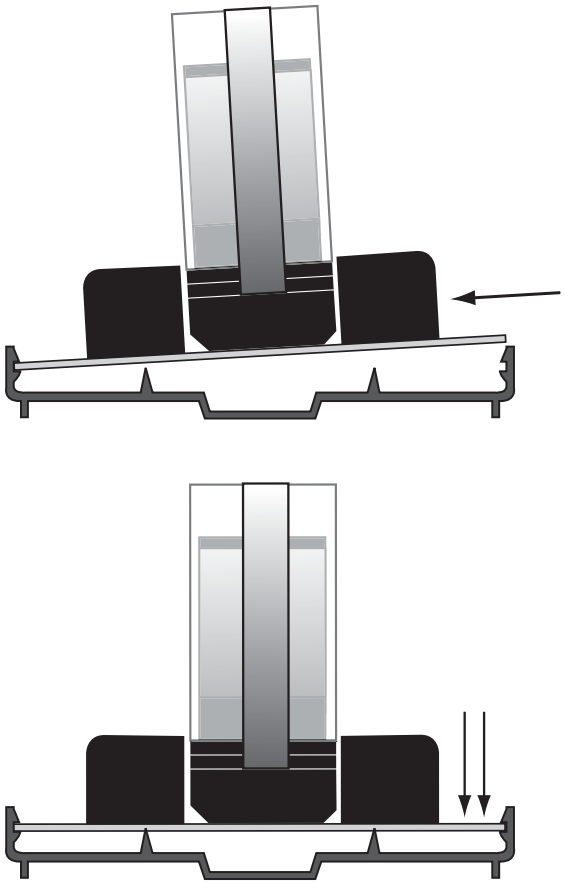

Figure 9-42: Attaching the SSB-DO2-I to the Snap-in Plastic Track

WARNING

To avoid damage to the module, you should not attempt to slide the module into the snap-in plastic track.

Once the module has been properly mounted, you may make the connections for STATbus and power.

9.13.3 WIRING/CONFIGURATION

9.13.3.1 POWER AND STATBUS CONNECTIONS

Connections for power and STATbus communications are made via terminal block TB1, shown in Figure 9-43. The two wire connection for STATbus communications should be connected to the two left most terminals labeled “SSB”. A source of 24VAC should be connected to the right two terminals labeled “X1” and “X2”. The easiest way to connect these terminals is to connect them to the AC OUT terminals on the same terminal block as the SSB connection on the controller. This allows you to simply run a 4-conductor cable to connect both power and STATbus communications to the device without any additional wiring. Alternately, a dedicated external power supply may be provided for the module.

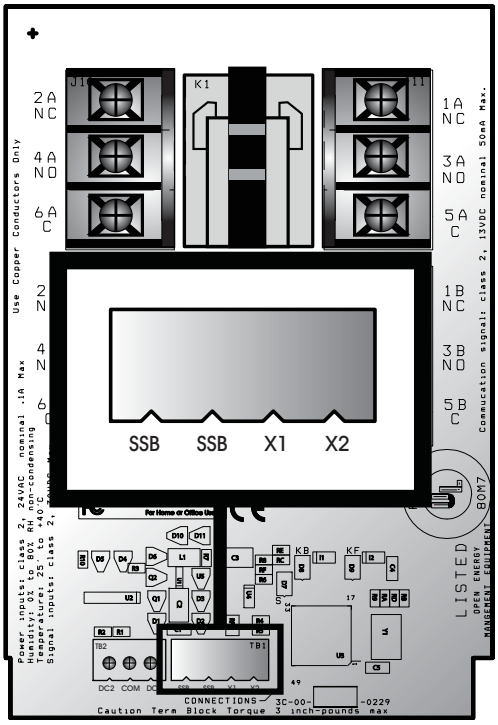

Figure 9-43: SSB and Power Connections on the SSB-DO2-I

9.13.3.2 RELAY CONNECTIONS

The SSB-DO2-I has two (2) DPDT relays for its outputs. When output 1 is energized, the connection is made between the normally open terminal, labeled 3A and 4A, and the common terminals, labeled 5 A and 6A. When the coil for output 2 is energized, the connection is made between the normally open terminal, labeled 3B and 4B, and the common terminals, labeled 5B and 6B. The normally open terminals are labeled “NO” below the terminal number and the common terminals are labeled “C”. When the output 1 is not energized connections will be made between the normally closed terminals, labeled 1A and 2A, and the common terminals. SImilarly, when output 2 is no energized, connections will be made between the normally closed terminals, labeled 1B and 2B, and the common terminals. The normally closed terminals are labeled “NC” below the terminal number. The function of each of the relay terminals is summarized in Table 9-12:.

Table 9-12: SSB-DO2-I Relay Terminals

Terminal	Connection
1A	Normally Closed (NC)
2A	Normally Closed (NC)
3A	Normally Open (NO)
4A	Normally Open (NO)
5A	Common

Table 9-12: SSB-DO2-I Relay Terminals

Terminal	Connection
6A	Common
1B	Normally Closed (NC)
2B	Normally Closed (NC)
3B	Normally Open (NO)
4B	Normally Open (NO)
5B	Common
6B	Common

9.13.3.3 DRY CONTACT INPUT CONNECTIONS

The SSB-DO2-I has two dry contact digital inputs intended for use as a status monitors for the relays on the module. Connecting a wet contact to these input will result in damage to the SSB-DO2-I. These inputs can only be assigned to Digital Inputs in the controller. Unlike the on-board digital input found on an NB-GPC, the dry contact inputs on the SSB-DO1-I are not capable of performing pulse counting. Connections for the inputs are made on terminal block TB2, with the first dry contact connected to the COM and DC1 terminals and the second dry contact connected to the COM and DC2 terminals as shown in Figure 9-44.

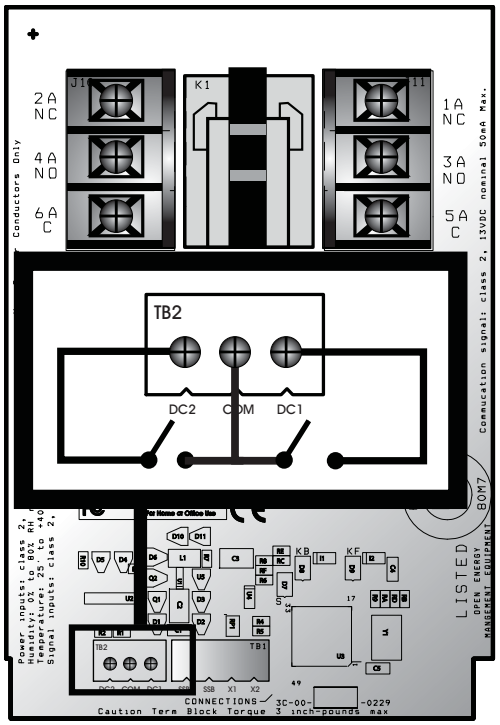

Figure 9-44: Connecting Dry Contact Inputs to the SSB-DO2-I

9.13.4 SSB-DO1-I CONFIGURATION TABLE

Table 9-13: SSB-DO1-I Configuration Table

I/O Termination	**(I# or O#)** Index Number
Digital Output 1	1
Digital Output 2	2
Digital Input 1	1
Digital Input 2	2

9.14 SSB-IOX FAMILY

The SSB-IOX Module Family are multi-I/O STATbus modules - providing multiple universal inputs, multiple analog outputs, and multiple digital outputs in contrast to singular SSB modules such as the SSB-UI1, SSB-AO1, and others.

SSB-IOX modules are available in four models, described below in Table 9-14.

Table 9-14 SSB-IOX Family Devices

Model	UIs	DIs	AOs	DOs
SSB-IOX1-1	4	1	2	2
SSB-IOX1-2	8	-	-	-
SSB-IOX2-1	12	-	6	6
SBC-IOX2-2	12	-	-	-

9.15 SSB-IOX1-x

9.15.1 SSB-IOX1-1 FEATURES

The SSB-IOX1-1, shown in Figure 9-45, is a STATbus module based on the same hardware as the GPC2 controller. It provides four (4) universal inputs, one (1) pulse input, two (2) analog outputs, and two (2) digital outputs.

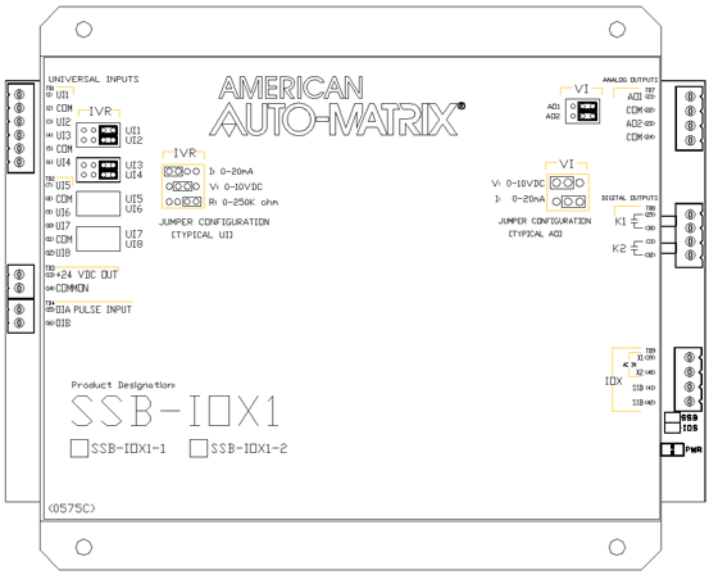

Figure 9-45 : The SSB-IOX1-1 Module

9.15.2 SSB-IOX1-2 FEATURES

The SSB-IOX1-2, shown in Table 9-46, is a STATbus module based on the same hardware as the GPC2 controller. It provides eight (8) universal inputs.

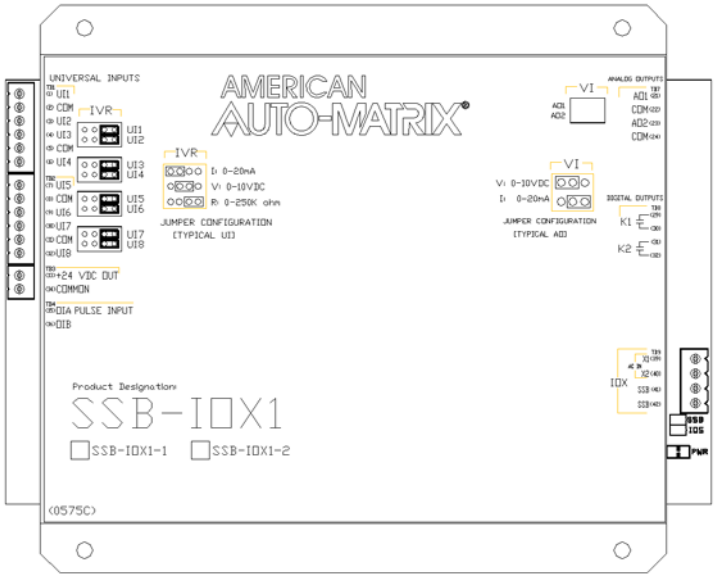

Figure 9-46 : The SSB-IOX1-2 Module

9.15.3 WIRING/CONFIGURATION

To properly configure the SSB-IOX1-X modules, you must connect the wires for network and power, connect any inputs and/or outputs, configure the IVR jumpers for Universal Inputs, and configure the VI jumpers for any Analog Outputs.

9.15.4 NETWORK & POWER

The SSB-IOX1-X must be connected to the STATbus network so that it may communicate. The network connection is made to terminal 41 and 42, labeled SSB, of terminal block TB9. The location of these terminals are shown in Figure 9-47. Power for the module is connected to terminals 39 and 40, labeled X1 and X2, on the same terminal block. This power may be provided from the AC Out terminals of the STATbus connection at the controller or a dedicated transformer may be connected.

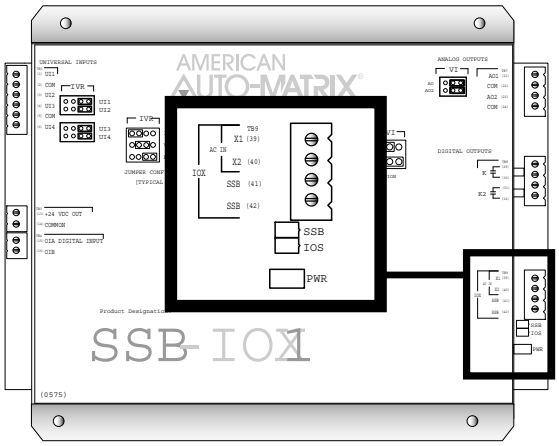

Figure 9-47: Location of the Network and Power Connections on the SSB-IOX1-1

9.15.5 UNIVERSAL INPUTS

To properly connect and configure the Universal Inputs on the SSB-IOX1-X, you must connect the sensor to the input and configure the IVR jumper to specify the type of sensor connected. The Universal Inputs are located in the upper left corner of the controller, as shown in Figure 9-48.

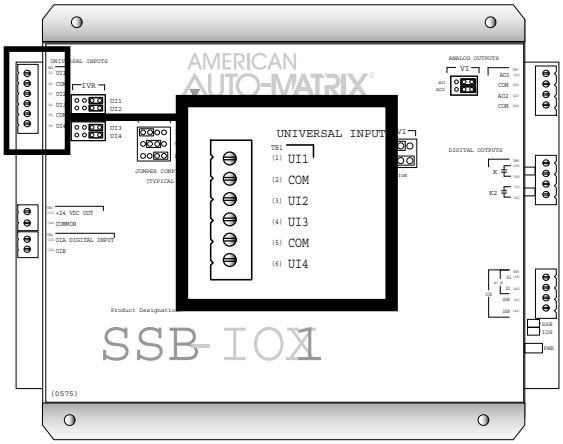

Figure 9-48: Location of the Universal Inputs on the SSB-IOX1-1 Module

To connect an input device to the SSB-IOX1-X, you must insert the leads from the sensor into the terminals for the desired input and the adjacent COM terminal. Two inputs share a single COM terminal, i.e. UIs 1 and 2 use the COM connection on terminal 2 while UIs 3 and 4 use the COM connection on terminal 5. Figure 9-49 shows how a thermistor would be connected to UI3.

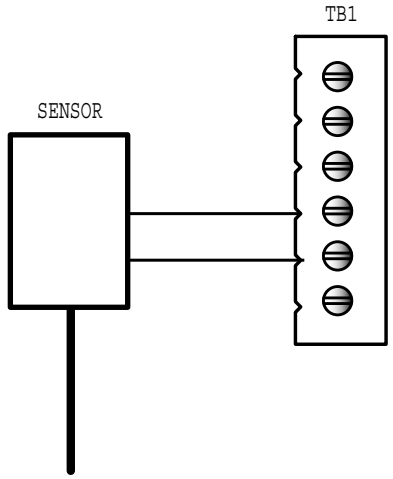

Figure 9-49: Connecting a Sensor to an Input on the SSB-IOX1-1

Each Universal Input on the SSB-IOX1-X has an IVR jumper, shown in Figure 9-50, associated with it which is used to select the type of input connected to the corresponding input.

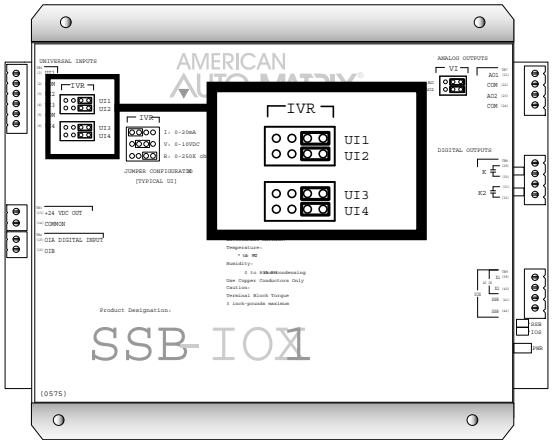

Figure 9-50: Location of the IVR Jumpers on the SSB-IOX1-1

Each input can be configured to read a 0-20 mA, 0-10 V or a 0-250 kΩ. The jumper settings corresponding to these options are shown in Figure 9-51a-c respectively.

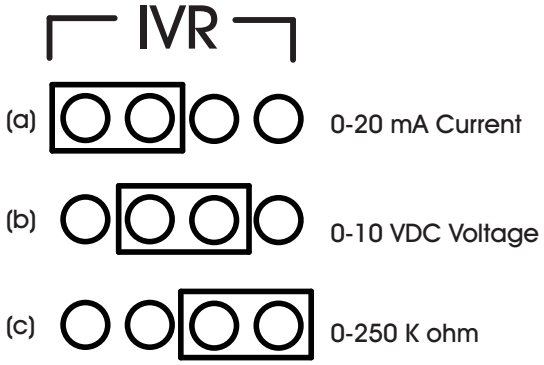

Figure 9-51: SSB-IOX1 IVR Jumper Positions

9.15.6 DIGITAL INPUTS

The SSB-IOX1-1 has a single Digital Input which is capable of performing high-speed pulse counting. The digital input is a wet contact input located on the left side of the module, as shown in Figure 9-52. There is a 24VDC power output connected to TB3 which is provided as a convenient way to power the wet contact connected to the input.

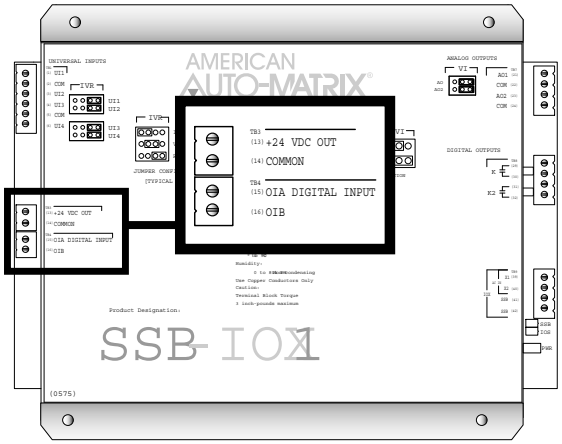

Figure 9-52: Location of the Digital Input and 24VDC Output on the SSB-IOX1-1

To connect a sensor to the pulse input on the SSB-IOX1-1, you must attach the leads from the sensor to the OIA and OIB terminals on terminal block TB4 as shown in Figure 9-53.

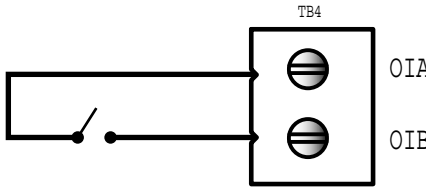

Figure 9-53: Connecting a Digital Input to the SSB-IOX1-1

9.15.7 ANALOG OUTPUTS

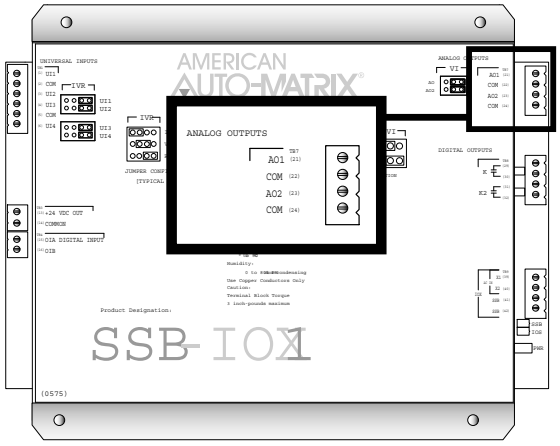

Figure 9-54: Location of the Analog Outputs on the SSB-IOX1-X

Analog outputs are connected to either terminals 21 (AO1) and 22 (COM) or 23 (AO2) and 24 (COM). A device connected to Analog Output 2 is shown in Figure 9-55.

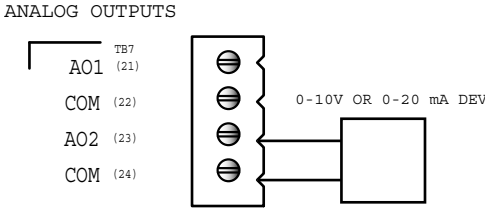

Figure 9-55: Connecting an Analog Output to the SSB-IOX1

For each Analog Output on the SSB-IOX1-1 there is a corresponding VI jumper used to select the output range for that output. These jumpers are located to the left of TB7. Each output can be configured for 0-10 VDC or 0-20 mA operation, using the jumper positions shown in Figure 9-56a and b respectively.

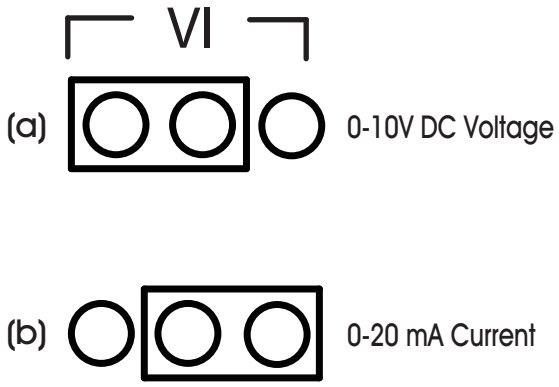

Figure 9-56: SSB-IOX1-1 VI Jumper Positions

9.15.8 DIGITAL OUTPUTS

The SSB-IOX1-1's two Digital Outputs are located on the right side of the controller as shown in Figure 9-57.

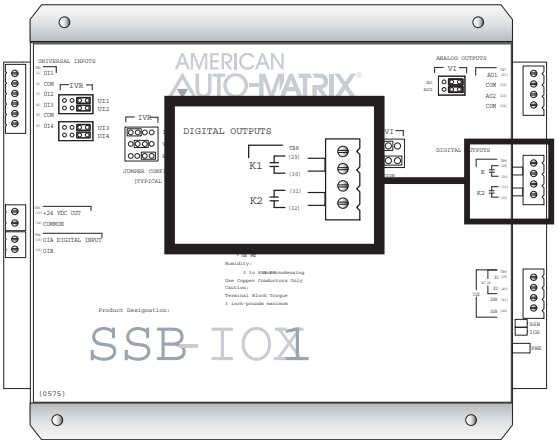

Figure 9-57: Location of the Digital Outputs on the SSB-IOX1-1

Output devices are connected using terminals 29 and 30 (labeled K1), for Digital Output 1, and terminals 31 and 32 (labeled K2), for Digital Output 2. Figure 9-58 shown a device connected to Digital Output 1.

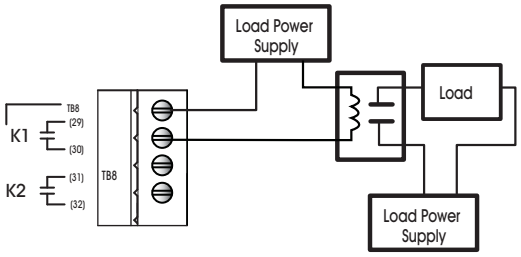

Figure 9-58: Connecting an Digital Output to the SSB-IOX1-1

9.15.9 MOUNTING THE SSB-IOX1-X

NOTE

The SSB-IOX1-1 and SSB-IOX1-2 is an open type device which, to meet UL specifications, is intended to be mounted on a panel that completes the ultimate enclosure.

The SSB-IOX1-X should be mounted to a site where the temperature is between 32º F and 122º F (0º C to 50º C) with a relative humidity of 0-80% non-condensing.

The mounting area should be flat and unobstructed by other equipment or machinery, free of moisture, and located away from potential leakage.

NOTE

When installing the SSB-IOX1-1 and SSB-IOX1-2 make sure that there is sufficient room to allow insertion and removal of the terminal block plugs..

9.15.10 STATUS INDICATOR LED

The SSB-IOX1-X has an IOS indicator LED which provides feedback on the current operational status of the module's IO processor. The IOS LED is located on lower right side of the module as shown in Figure 9-59.

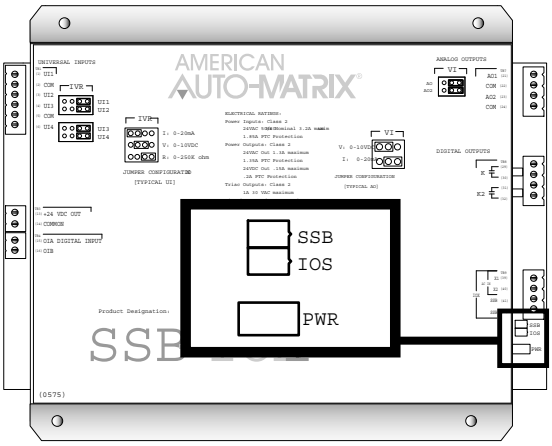

Figure 9-59: Location of the IOS Indicator LED on the SSB-IOX1-1

The IOS LED shows one of four different states: powered but not enumerated, enumerated but not configured, configured, and "identify". The different states are indicated by the rate at which the LED blinks. The LED blinking quickly, approximately three to four times per second, indicates that the unit is powered but has not yet been enumerated by the controller. This is useful for identifying units that are correctly wired but not configured. When the device is enumerated but not configured, the blink rate will slow to approximately twice a second.

Once the device has been configured, the blink rate will slow down to approximately one blink per second. This will be the normal state of the device when it is correctly wired, powered, enumerated, and configured in the controller.

When the controller is set to "Identify" in the Configure Function and Configure Device properties, the IOS LED on the SSB-IOX1 will be blink three times in quick succession and then pause before repeating the three blinks again. This is especially useful for quickly identifying an individual device in the field when troubleshooting the STATbus.

9.15.11SSB-IOX1-1 CONFIGURATION TABLE

Table 9-15: SSB-IOX1-1 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 1	1
Universal Input 2	2
Universal Input 3	3
Universal Input 4	4
Digital Input 1	1
Analog Output 1	1
Analog Output 2	2
Digital Output 1	1
Digital Output 2	2

9.15.11.1 SSB-IOX1-2 CONFIGURATION TABLE

Table 9-16 SSB-IOX1-2 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 1	1
Universal Input 2	2
Universal Input 3	3
Universal Input 4	4
Universal Input 5	5
Universal Input 6	6
Universal Input 7	7
Universal Input 8	8

9.16 SSB-IOX2-X

9.16.1 SSB-IOX2-1 FEATURES

The SSB-IOX2-1, shown in Figure 9-60, is a STATbus module based on the same hardware as the GPC1 controller. It provides four (12) universal inputs, six (6) analog outputs, and six (6) digital outputs.

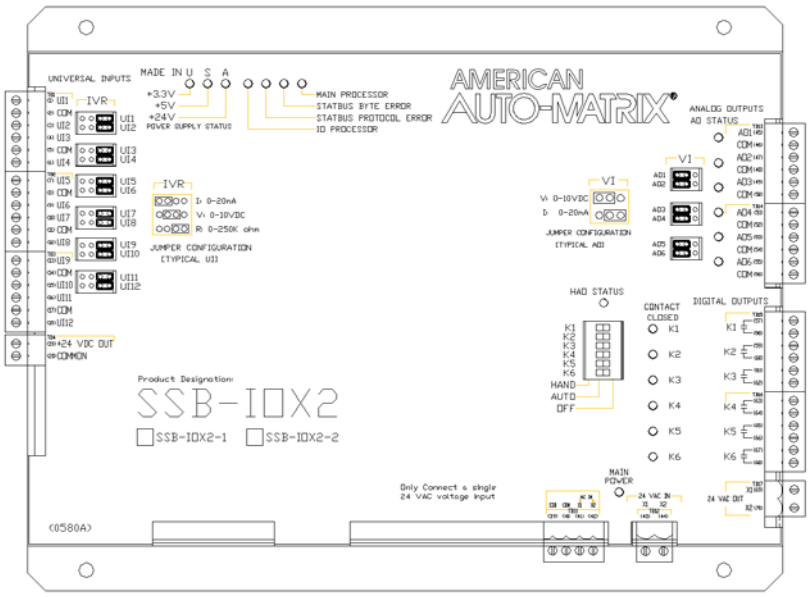

Figure 9-60 : The SSB-IOX2-1 Module

9.16.2 SSB-IOX2-2 MODULE

The SSB-IOX2-2, shown in Figure 9-61, is a STATbus module based on the same hardware as the GPC1 controller. It provides twelve(12) universal inputs.

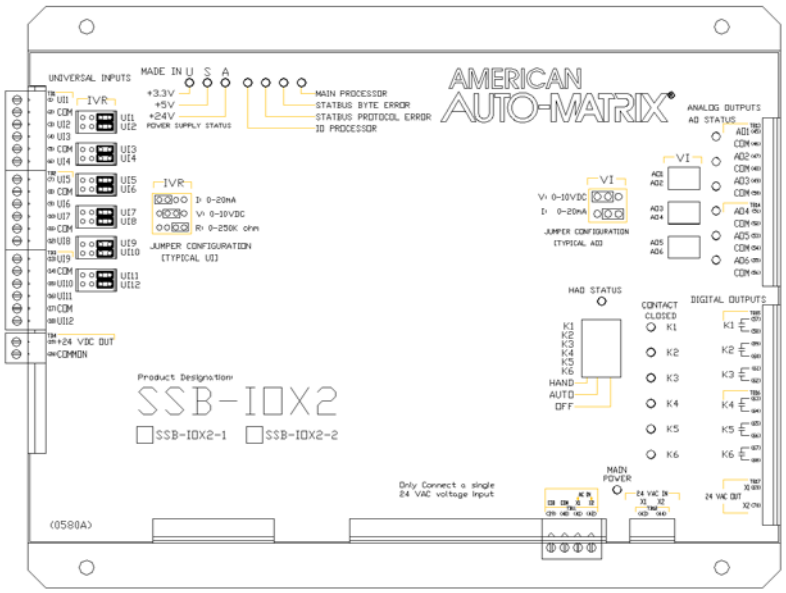

Figure 9-61 : The SSB-IOX2-2 Module

9.16.3 WIRING/CONFIGURATION

To properly configure the SSB-IOX2-X modules, you must connect the wires for network and power, connect any inputs and/or outputs, configure the IVR jumpers for Universal Inputs, and configure the VI jumpers for any Analog Outputs.

9.16.4 NETWORK & POWER

The SSB-IOX2-X must be connected to the STATbus network so that it may communicate. The network connection is made to terminal 41 and 42, labeled SSB, of terminal block TB9. The location of these terminals are shown in Figure 9-47. Power for the module is connected to terminals 39 and 40, labeled X1 and X2, on the same terminal block. This power may be provided from the AC Out terminals of the STATbus connection at the controller or a dedicated transformer may be connected.

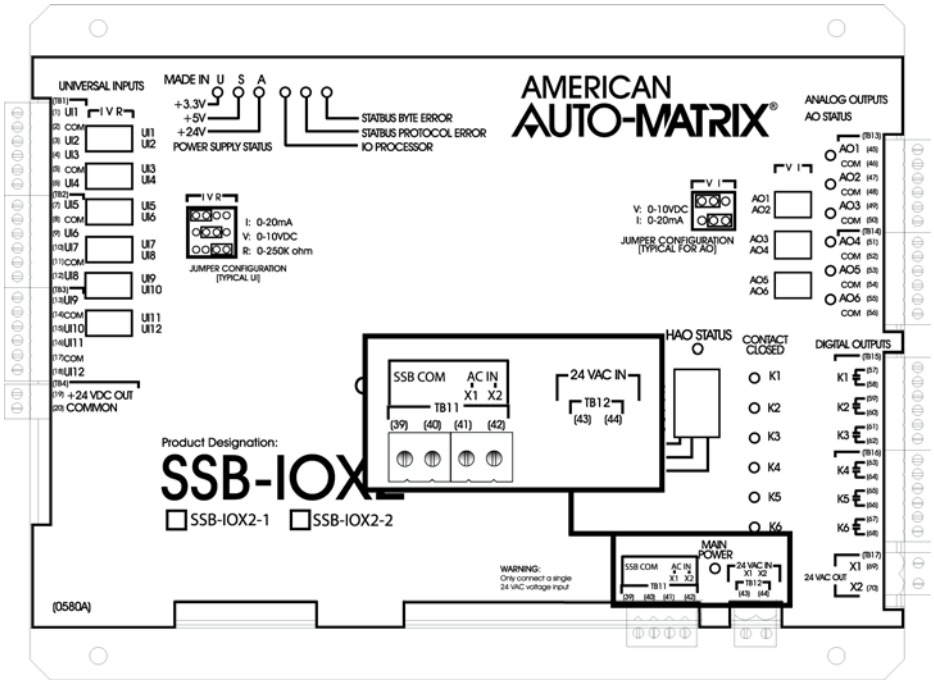

Figure 9-62: Location of the Network and Power Connections on the SSB-IOX2-1

9.16.5 UNIVERSAL INPUTS

To properly connect and configure the Universal Inputs on the SSB-IOX2-X, you must connect the sensor to the input and configure the IVR jumper to specify the type of sensor connected. The Universal Inputs are located in the upper left corner of the controller, as shown in Figure 9-48.

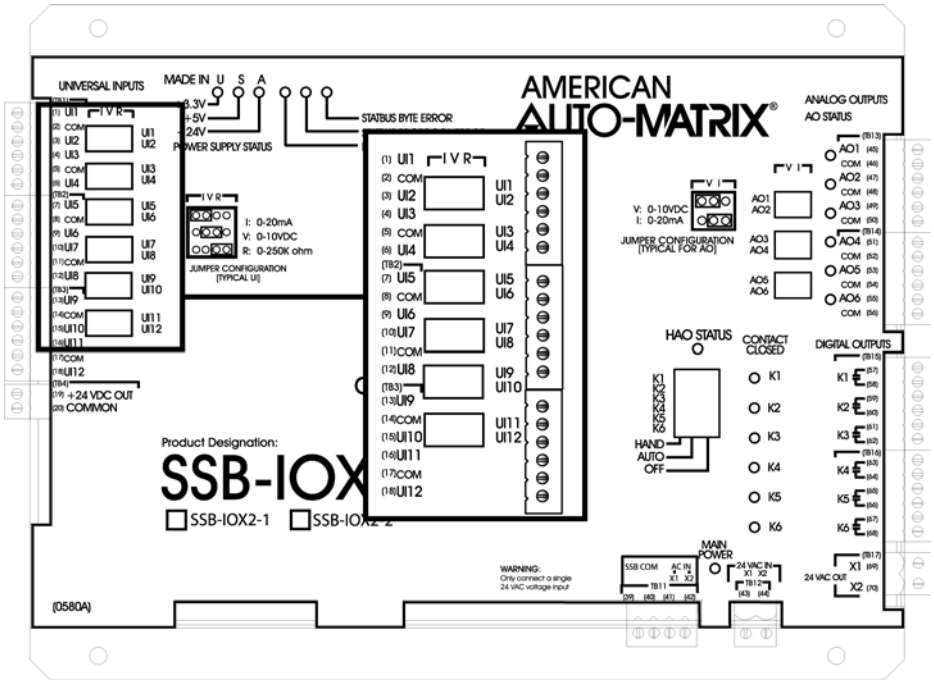

Figure 9-63: Location of the Universal Inputs on the SSB-IOX2-1 Module

To connect an input device to the SSB-IOX2-X, you must insert the leads from the sensor into the terminals for the desired input and the adjacent COM terminal. Two inputs share a single COM terminal, i.e. UIs 1 and 2 use the COM connection on terminal 2 while UIs 3 and 4 use the COM connection on terminal 5. shows how a thermistor would be connected to UI3. Figure 5-64 shows how a thermistor would be connected to UI3.

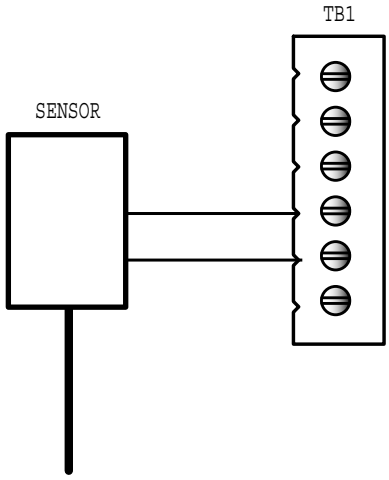

Figure 9-64: Connecting a Sensor to an Input on the SSB-IOX2-1

Each Universal Input on the SSB-IOX2-X has an IVR jumper, shown in Figure 5-65, associated with it which is used to select the type of input connected to the corresponding input.

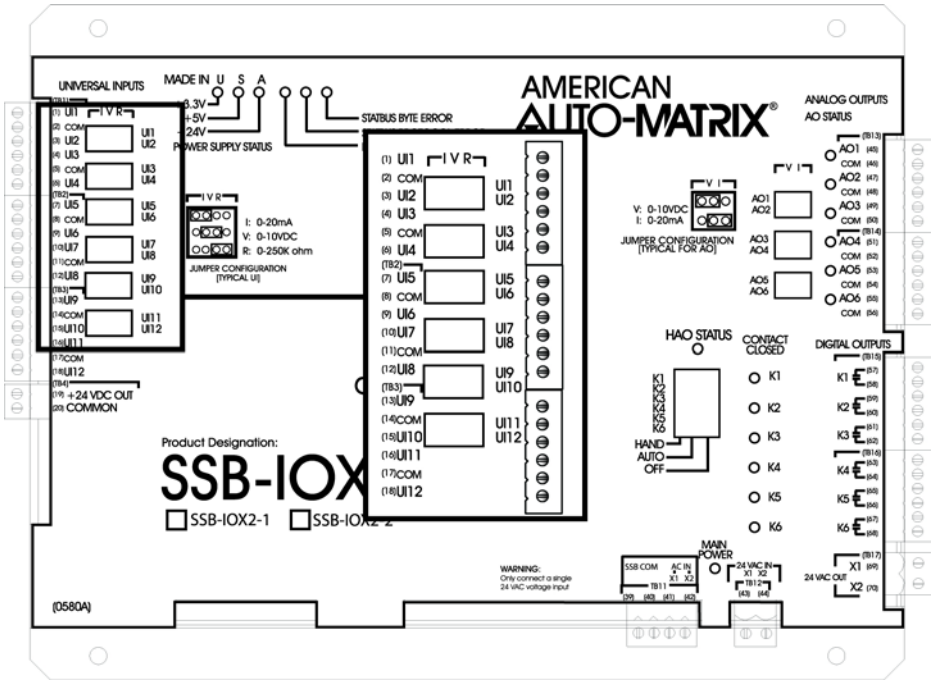

Figure 9-65: Location of the IVR Jumpers on the SSB-IOX2-1

Each input can be configured to read a 0-20 mA, 0-10 V or a 0-250 kΩ. The jumper settings corresponding to these options are shown in Figure 9-51a-c respectively.

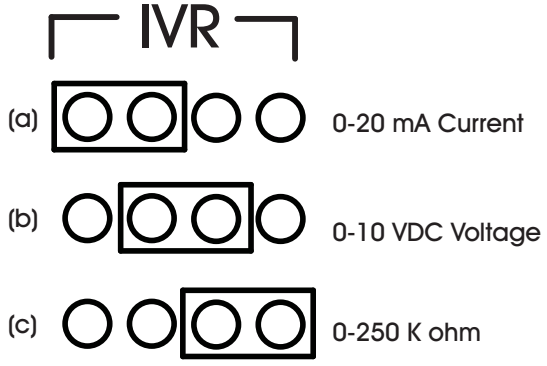

Figure 9-66: SSB-IOX1 IVR Jumper Positions

9.16.6 ANALOG OUTPUTS

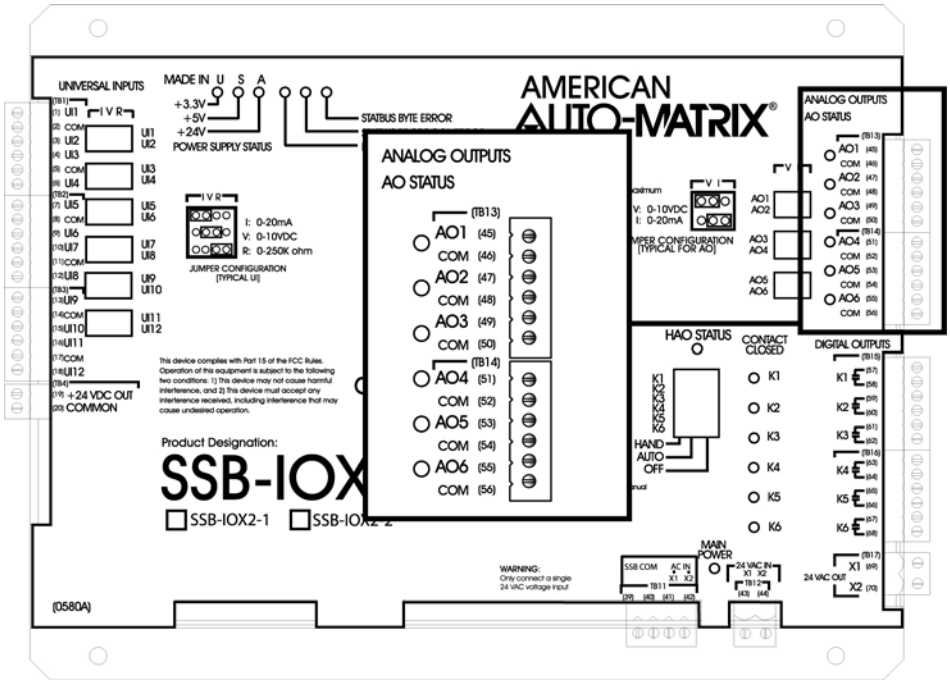

Figure 9-67: Location of the Analog Outputs on the SSB-IOX2-1

Analog outputs are connected to either terminals 45(AO1) and 46(COM) or 47(AO2) and 48(COM). A device connected to Analog Output 2 is shown in Figure 5-68.

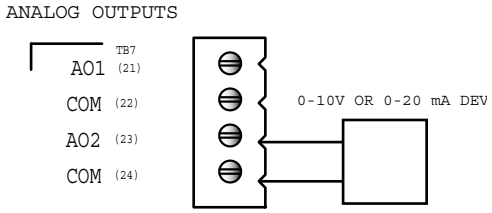

Figure 9-68: Connecting an Analog Output to the SSB-IOX2-1

For each Analog Output on the SSB-IOX2-1 there is a corresponding VI jumper used to select the output range for that output. These jumpers are located to the left of TB13. Each output can be configured for 0-10 VDC or 0-20 mA operation, using the jumper positions shown in Figure 5-69a and b respectively.

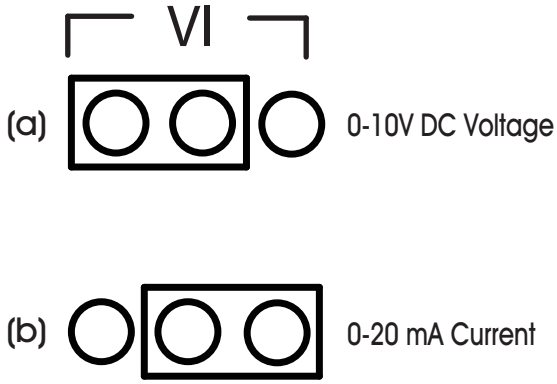

Figure 9-69: SSB-IOX1-1 VI Jumper Positions

9.16.7 DIGITAL OUTPUTS

The SSB-IOX2-1's six Digital Outputs are located on the right side of the controller as shown in Figure 5-70.

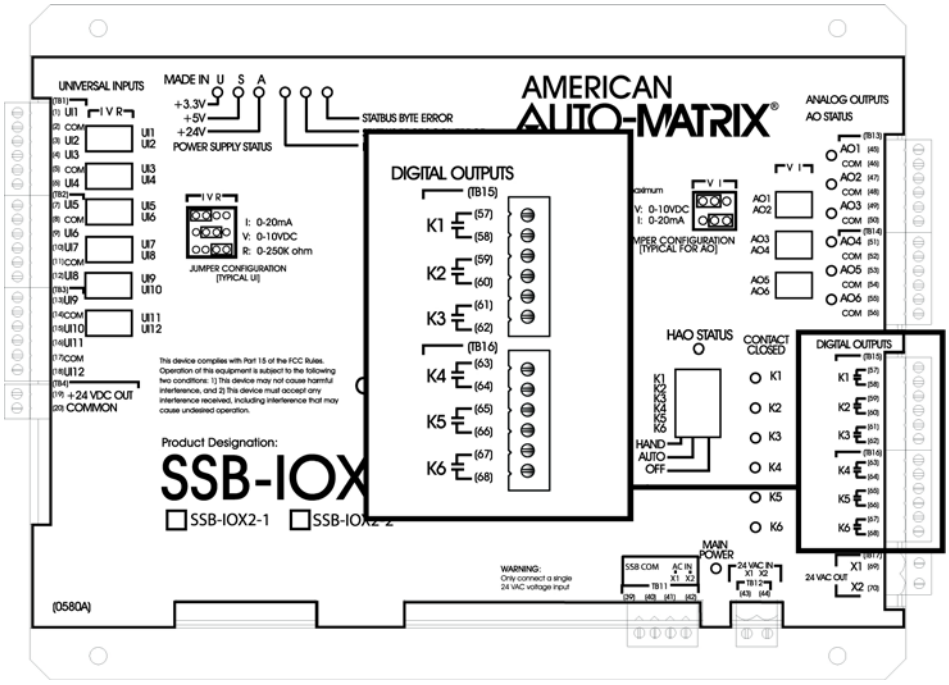

Figure 9-70: Location of the Digital Outputs on the SSB-IOX2-1

Output devices are connected using terminals 57 and 58 (labeled K1), for Digital Output 1, and terminals 59and 60(labeled K2), for Digital Output 2. Figure 5-71 shows a device connected to Digital Output 1.

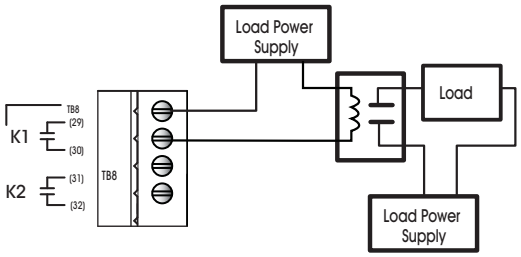

Figure 9-71: Connecting an Digital Output to the SSB-IOX2-1

9.16.8 MOUNTING THE SSB-IOX2-X

NOTE

The SSB-IOX2-x is an open type device which, to meet UL specifications, is intended to be mounted on a panel that completes the ultimate enclosure.

The SSB-IOX2-X should be mounted to a site where the temperature is between 32º F and 122º F (0º C to 50º C) with a relative humidity of 0-80% non-condensing.

The mounting area should be flat and unobstructed by other equipment or machinery, free of moisture, and located away from potential leakage.

NOTE

When installing the SSB-IOX2-X make sure that there is sufficient room to allow insertion and removal of the terminal block plugs..

9.16.9 SSB-IOX2-1 CONFIGURATION TABLE

Table 9-17: SSB-IOX2-1 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 1	1
Universal Input 2	2
Universal Input 3	3

Table 9-17: SSB-IOX2-1 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 4	4
Universal Input 5	5
Universal Input 6	6
Universal Input 7	7
Universal Input 8	8
Universal Input 9	9
Universal Input 10	10
Universal Input 11	11
Universal Input 12	12
Analog Output 1	1
Analog Output 2	2
Analog Output 3	3
Analog Output 4	4
Analog Output 5	5
Analog Output 6	6
Digital Output 1	1
Digital Output 2	2
Digital Output 3	3
Digital Output 4	4
Digital Output 5	5
Digital Output 6	6

9.16.9.1 SSB-IOX2-2 CONFIGURATION TABLE

Table 9-18 SSB-IOX2-2 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 1	1
Universal Input 2	2

Table 9-18 SSB-IOX2-2 Configuration Table

I/O Termination	**(I# or O#)** Index Number
Universal Input 3	3
Universal Input 4	4
Universal Input 5	5
Universal Input 6	6
Universal Input 7	7
Universal Input 8	8
Universal Input 9	9
Universal Input 10	10
Universal Input 11	11
Universal Input 12	12

SECTION 10: INPUTS SETUP

This section describes the process of setup and configuration of both Universal Inputs (which can be Analog or Binary) and Digital Inputs (optically isolated, high-speed pulse-counting inputs), as well as the setup of Piecewise Curves for custom Analog Input sensor configurations. The MatrixBBC supports up to a maximum of 144 Inputs (Analog or Binary).

IN THIS SECTION

10.1 INPUTS OVERVIEW

MatrixBBC product models support Universal Inputs, capable of being configured to monitor analog and binary signals. Models of GPC products support either on-board I/O or expansion modules via STATbus. While the total amount of inputs per GPC will depend on the model number, setup and configuration of outputs is exactly the same across the product family. The table below provides information on-board I/O support, as well as expansion support.

10.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

To enhance your programming experience, the following are a few helpful concepts and techniques to keep in mind when using these objects.

10.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The GPC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used Input objects. This allows you not only to keep better track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

10.1.1.2 ENABLE ALARMING WHEN NEEDED

All Input objects optionally support alarming. When alarming is disabled (EA) Enable Alarming = Disabled (0), fewer properties will be displayed in NB-Pro, allowing the objects to be interpreted easier during programming.

10.2 UNIVERSAL INPUTS

Each universal input may operate as either a digital or analog input. Each input may be configured individually to read any of the following:

. digital (on/off)
. linear inputs (0-5 V, 0-10 V, 0-20 mA, 4-20 mA, etc.) scaled between a programmable minimum and maximum value
. non-linear inputs with response provided via programmable Piecewise Curve objects
. thermistor (Precon type III 77º @ 10kΩ)
. low-speed pulsing
. SMARTStat device (using available instances higher than the pre-assigned on-board)

10.2.1 CREATING ANALOG INPUTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Analog Input objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Analog Input object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MK) Max Analog Inputs** property. By default, this value is set to 0, indicating no Analog Input objects exist.
3. Write the number of total Analog Input objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Analog Inputs, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Analog Input objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

10.2.2 ANALOG INPUT CONFIGURATION

The following sub-sections discuss the configuration of analog inputs.

10.2.2.1 CURRENT INPUTS

Any sensor which generates a signal in the form of a current is classified as a current sensor. Ranges of 0-20 mA and 4-20 mA are common in sensors. The current produced by these sensors is often proportional to the value being measured. For example, if a pressure sensor measured 0 to 5 inches of water gauge and had an output range of 0 to 20 mA, then a reading of 10 mA would correspond to a pressure of 2.5 inches of water gauge.

The first thing that needs to be done so that a universal input can be used as a current input is to make sure that the STATbus expansion module has been configured property (relative to jumpers).

Once the type of sensor has been set, you need to use *NB*-Pro. Configuration entails telling the MatrixBBC what type of sensor is connected to an input and then specifying the range of values over which that sensors operates.

To specify the sensor type, you must open the Analog Input object corresponding to the input you are configuring. Next, you want to set the **(ST) Sensor Type** property to a value of 3, for 4-20 mA inputs, or 8, for 0-20 mA inputs.

NOTE

When switching from a digital sensor type to an analog sensor type, the object type will automatically change from binary to analog. When this occurs, you must re-discover the object using the appropriate selection from the Discovery menu of NB-Pro. Refer to NB-Pro User Manual for more information.

You also need to specify the range over which the sensor operates. This is necessary so that the object can calculate the measured value from the input signal. The **min-pres-value** property should be set to the lowest value that your sensor can measure. The **max-pres-value** property should be set to the maximum scaled value for your input.

As an example, sensors which measure relative humidity are often current sensors that operate in the 4-20 mA range. For a sensor of this type you would set **ST**=3 because the sensor measures 4-20 mA. Relative humidity ranges from 0 to 100% so you would set **min-pres-value**=0 and **max-pres-value**=100 to represent the limits of the sensor's output. In this case, a raw value of 4 mA would be scaled to a value of 0% in engineering units. The relative humidity sensor would read 100% if the input were reading a signal of 20mA.

10.2.3 VOLTAGE INPUTS

Any sensor which puts out a voltage in response to a measured value is classified as a voltage sensor. Voltage sensors behave in very much the same way as current sensors.

To specify the sensor type, you must to go to Universal Input object corresponding to the input you are configuring. Next, you want to set the **(ST) Sensor Type** property to a value of 2, for 0-5 V inputs, or 6, for 0-10 V inputs.

NOTE

When switching from a binary sensor type to an analog sensor type, you must reset the MatrixBBC after changing the sensor type. Once done, you then have to rediscover the associated object. This process allows you to view other properties associated to the object, such as minimum and maximum scaled values. Refer to NB-Pro User Manual for more information.

You also need to specify the range over which the sensor operates. This is necessary so that the object can calculate the measured value from the input signal. The **min-pres-value** property should be set to the lowest value that your sensor can measure. This would correspond to the reading at zero volts. The **max-pres-value** property should be set to the maximum scaled value for your input. For example, if a 0-10 V

carbon dioxide sensor measuring from 0-5000 ppm would have **min-pres-value** would be set to 0 and **max-pres-value** would be set to 5000.

10.2.3.1 THERMISTOR INPUTS

The thermistor is one of the most common types of resistive sensors for temperature measurement. The thermistor's combination of high accuracy over a wide range coupled with its low cost makes it one of the most popular temperature sensors used. Because of the thermistor's popularity, the MatrixBBC has the response curve for a **Precon type III** thermistor built in.

Set the **(ST) Sensor Type** property equal to 7, the value which corresponds to a thermistor. The object will then automatically set the **min-pres-value** to -35.0 and the **max-pres-value** to 240.0, the minimum and maximum values that can be read by this type of sensor. The temperature will now be displayed in the **present-value** property of this object. The value of **present-value** will also be displayed in the Universal Input Summary object.

NOTE

When switching from a binary sensor type to an analog sensor type, you must rediscover the associated object. This process allows you to view other properties associated to the object, such as minimum and maximum scaled values. Refer to NB-Pro User Manual for more information.

At this point, it is helpful to give the object a name so it can be easily identify in the future. You can name the object by setting the value of the **object-name** property. You should then check the reading and adjust any disagreement between the sensor and a known reading using the **(OF) Input Offset** property, This specifies a fixed offset for the sensor and would be used if the sensor reading was off by a fixed amount. For example, if a sensor was reading three degrees below the actual room temperature. In that case, you would set **OF**=3.0 to correct the reading.

10.2.3.2 NON-LINEAR INPUTS

The thermistors discussed previously are just one example of a sensor who's output characteristics are well defined. Because Precon type III thermistors are so prevalent, the output response curve is included with the controller. However, there are a number of other common sensors who's response is non-linear. For these types of inputs, the MatrixBBC provides the option of using one of the available Piecewise Curve objects to specify the response of the input device.

The Piecewise Curves can be used with current, voltage, or resistive inputs. Before configuring the Piecewise Curve, make sure that the IVR jumper is in the correct position for the type of sensor being used on the STATbus Expansion Module.

To use an input with a non-linear response, you must define the Piecewise Curve and then set the input to use the curve you have defined to scale its readings. The process for defining the Piecewise Curve is given later in this manual. The response data necessary to construct the curve will usually be available in the catalog from which the sensor was ordered or on the data sheet accompanying the sensor.

To set the input to use the Piecewise Curve, set the **(ST) Sensor Type** property for the input equal to one of the available Piecewise Curve objects. This tells the MatrixBBC to use a Piecewise Curve object.

NOTE

When switching from an binary sensor type to an analog sensor type, you must rediscover the associated object. This process allows you to view other properties associated to the object, such as minimum and maximum scaled values. Refer to NB-Pro User Manual for more information.

At this point, you should give the object a name so that you can easily identify it in the future. You can name the object by writing to the **object-name** property.

10.2.4 CONFIGURING ANALOG INPUT ALARM/EVENT NOTIFICATIONS

Analog Inputs can be configured to support alarm/event notifications. To enable alarming for an Analog Inputs, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Analog Inputs objects can be configured to trigger one of the two following conditions:

. Low Limit - occurs when **present-value** is less than the value specified in **low-limit**.
. High Limit - occurs when **present-value** is greater than the value specified in **high-limit**.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.
3. Configure **limit-enable** to enable low-limit or high-limit alarming. This is accomplished by placing a check into each associated limit type.
4. Configure event-enable to have the object send alarms for how alarms transition. For example, if you wish to have the GPC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
5. Configure your **high-limit** and **low-limit** properties accordingly.
6. Configure the **time-delay** and **deadband** properties. The time-delay property defines a threshold of time (in seconds) where the **present-value** must exceed one of the limit properties in order for an alarm/event condition to be considered. The **deadband** property defines an offset from **low-limit** or **high-limit** that must be met in order for an alarm/event condition to be considered. For example, if **high-limit** = 75.0, **deadband** = 2.0, and **time-delay** = 5, the **present-value** must exceed 77.0 for at least 5 seconds before an alarm/event condition is considered.

10.2.5 CREATING BINARY INPUTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Binary Input objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Binary Input object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(ML) Max Binary Inputs** property. By default, this value is set to 0, indicating no Binary Input objects exist.
3. Write the number of total Binary Input objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Binary Inputs, write a value of 10.

4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Binary Input objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

10.2.6 BINARY INPUT CONFIGURATION

An input that only has two signal states is considered a binary input. The most basic binary inputs are switches or contacts. The switch is either on or off, the contact is closed or open. Inputs of this type have many uses in a building automation system.

Despite only having two possible states, binary inputs require a bit more configuration than their analog counterparts. Like an analog input, you should first check to make sure that the IVR jumper is in the correct position on the STATbus Expansion Module. For a binary input, you want to set the jumper to the "R" position. This is used because an open contact would have a very high resistance while a closed contact would have a very low resistance, making it easier to detect the states.

For a binary sensor, you will set the **(ST) Sensor Type** property to Digital (0).

NOTE

When switching from a analog sensor type to a binary sensor type, you must rediscover the associated object. This process allows you to view other properties associated to the object, such as minimum and maximum scaled values. Refer to NB-Pro User Manual for more information.

Binary inputs can take one of two states, but you can tell the controller how you want it to treat those states. By setting the **polarity** property you can specify whether the object should display **present-value**=1 for a high signal (normal operation, **polarity**=0) or a low signal (reverse operation, **polarity**=1).

Binary inputs also have a **(RH) Run Hours** property. This property tracks the amount of time (in hours) that the sensed signal has been active.

CAUTION

INB-GPC products manufactured after April 2010 can only have their Universal Inputs configured to perform pulse counting. Devices manufactured before this date cannot perform pulse counting via Universal Inputs.

10.2.7 CONFIGURING BINARY INPUT ALARM/EVENT NOTIFICATIONS

Binary Inputs can be configured to support alarm/event notifications. To enable alarming for an Binary Inputs, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Binary Inputs object alarm/events are triggered through defining the state in which the output is considered to be in alarm.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.
3. Configure **event-enable** to have the object send alarms for how alarms transition. For example, if you wish to have the GPC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
4. Configure the **time-delay** property. The time-delay property defines a threshold of time (in seconds) where the **present-value** must maintain the value in order for an alarm/event condition to be considered.

10.3 DIGITAL INPUTS

Some STATbus Expansion Modules include an optically isolated digital input with dedicated pulse counting features. These digital inputs are capable of detecting signals in the range 3-40 VDC peak to peak or 2-29 VAC at 50/60 Hz. The digital inputs operate at a much higher frequency than a Universal Input. This makes it possible for the input to not only detect whether a signal is on or off, but to detect rapid pulses from the input. These pulses would be used primarily for demand metering applications as part of an energy management system. Digital, pulse counting inputs can also be uses in flow metering applications.

Regardless of the specific application, all pulse counting, digital devices operate on the same principle. The device will generate a pulse for a given quantity of the value that is being measured for a demand metering application. One pulse might correspond to one kilowatt-hour of power whereas, for a flow metering setup, a single pulse might correspond to one gallon of liquid. The important piece of information is the correlation between pulses and measured values.

10.3.1 CONFIGURING THE DIGITAL INPUTS

To configure a digital pulse counting input, you must set the **(MD) Pulse Counter Mode** property to “Rising Edge”, “Falling Edge”, or “Both”. This tells the MatrixBBC that you want the input to operate as pulse counter and not a simple digital input.

To correlate pulses with the value being measured, you must enter a value for the **(SF) Scaled Factor** property. This multiplier specifies the amount of the measured quantity that is accumulated for each pulse. For example, if a demand meter sends out a pulse for each kilowatt-hour of power used, then you would set **SF**=1.0 since one pulse corresponds to one kilowatt-hour. If your input device were a flow meter that sent a pulse for every ten gallons of liquid that passed through a pipe, then you would set **SF**=10.0.

The total number of pulses accumulated will be displayed in the **(NP) Number of Pulses Accumulated** property. While this can be useful, you will be more concerned with the scaled pulse count stored in the **(SV) Scaled Value** property. This is the value of the total number of pulses, **NP,** multiplied by the scaling factor, **SF**. **SV** gives you the total amount of the value that you are measuring. For example, if the object has accumulated 1250 (**NP**=1250) pulses and each pulse corresponds to 2.5 gallons of liquid that has been pumped through a pipe (**SF**=2.5), then the total amount of liquid pumped (**Neptis**) would be displayed in **SV**. In this case, **SV** would have a value of 3125, meaning that 3125 gallons of liquid had been measured.

10.4 PIECEWISE CURVES

MatrixBBC products can accommodate non-linear sensors by using built-in tables to define the response characteristics of an Analog Input sensor. Each table requires eleven points to define ten linear segments which approximate the response of the sensor. The controller will perform a linear interpolation to 'look up' values that lie along an individual segment much like the calculations performed by the Scale objects. Each of the Piecewise Curve objects contain the following properties: **object_identifier, object_name**, **object_type, X1**, **X2**, **X3**, **X4**, **X5**, **X6**, **X7**, **X8**, **X9**, **XA**, **XB**, **Y1**, **Y2**, **Y3**, **Y4**, **Y5**, **Y6**, **Y7**, **Y8**, **Y9**, **YA** and **YB.**

10.4.1 CREATING PIECEWISE CURVES IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Piecewise Curve objects are created by the technician when necessary, and are done in a dynamic manner.

To create a Piecewise Curve object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(M5) Max PWC** property. By default, this value is set to 0, indicating no Piecewise Curve objects exist.
3. Write the number of total Piecewise Curve objects you wish to have in the MatrixBBC. For example, if you wish to have 5 Piecewise Curves, write a value of 5.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Piecewise Curve objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

10.4.2 PIECEWISE CURVE CONFIGURATION

The **object_name** property stores the name of the object. This is a user definable string that can be used to help identify the object or the sensor type it is approximating.

Properties **(X1) Point 1's value in % Full Scale** through **(XB) Point 11's value in % Full Scale** represent the sensor readings for eleven chosen points on a sensor curve. The acceptable range for **X1** through **XB** depend on the chosen sensor type. For a voltage input, the 0-10 V input range is mapped to **X1** through **XB** values from 0 through 100. A current input, with a range of 0-20 mA, can have **X1** through **XB** values from 0 through 50. The 0-250 kΩ range for a resistive input can have **X1** through **XB** values from 0 through 25. The values of **X1** through **XB** must be entered in increasing order (i.e. **X1** < **X2** < **X3** etc.).

NOTE

The Piecewise Curve will only interpolate values for input values between **X1** and **XB**. If the input is below **X1**, the Piecewise Curve will be pegged at the value associated with **X1**. If the input is above **XB**, the Piecewise Curve will be pegged at the value associated with **X1**.

Properties **(Y1) Point 1's value in engineering units** through **(YB) Point 11's value in engineering units** are the Engineering Unit values (e.g., 70 degrees, 72 degrees, etc.), corresponding to the sensor readings entered into **X1** through **XB**. These values, coupled with the corresponding sensor readings, define the line segments which make up the piecewise curve

10.4.3 PIECEWISE CURVES FOR VOLTAGE INPUTS

To program a piecewise curve for a nonlinear sensor, you need to know the response characteristics of the sensor. These response characteristics are usually supplied by the manufacturer and may be in the form of a graph or table. Figure 10-1 shows an example of what a curve for a temperature sensor may look like. Though the sensor response could extend beyond this range, you will get more accurate results if you limit the range of your Piecewise Curve to the range of values you expect to see from the sensor. Figure 10-1 only contains the portion of the sensor's response that would be needed for zone temperature monitoring.

Once you have the response data, in either graph or table form, you must choose the points which define the line segments that approximate the response curve in the expected response region. When choosing the points to use, you can use fewer points in areas of the curve that are mostly linear and concentrate the points to better approximate the more non-linear portions of the response. In Figure 10-1, you can see that more points are chosen near the 'bends' in the characteristic response curve.

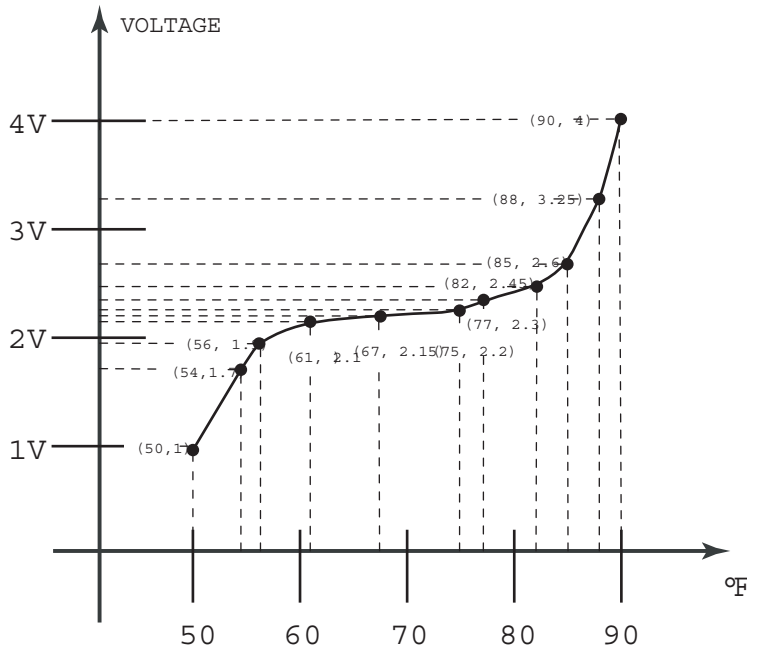

Figure 10-1: An Example of a Sensor Response Curve

From the graph, the following values are selected to represent the curve:

Table 10-1 : Sensor Response Points

Voltage	Temperature (°F)
1.00	50
1.70	54
1.90	56
2.10	61
2.15	67
2.20	75

Table 10-1 : Sensor Response Points

Voltage	Temperature (°F)
2.30	77
2.45	82
2.60	85
3.25	88
4.00	90

Next you must convert the voltage values into a percentage of full scale to be used to define the x-coordinates of your piecewise curve. Since the voltage input has a range of 0-10 V, each volt measured corresponds to ten percent of the full scale. The formula for the percentage of full scale output for a voltage sensor is simply:

$$\% \text{ Full Scale} = \text{Voltage} \times 10$$

The calculated percentages correspond to **X1** through **XB** and the temperature reading correspond to **Y1** through **YB**. For the sensor described above, this gives the following assignments:

Table 10-2 : Assigning Sensor Response Points to the Piecewise Curve

	% Full Scale	Temperature (°F)	
X1 ⇦	10.0	50	⇨ **Y1**
X2 ⇦	17.0	54	⇨ **Y2**
X3 ⇦	19.0	56	⇨ **Y3**
X4 ⇦	21.0	61	⇨ **Y4**
X5 ⇦	21.5	67	⇨ **Y5**
X6 ⇦	22.0	75	⇨ **Y6**
X7 ⇦	23.0	77	⇨ **Y7**
X8 ⇦	24.5	82	⇨ **Y8**
X9 ⇦	26.0	85	⇨ **Y9**
XA ⇦	32.5	88	⇨ **YA**
XB ⇦	40.0	90	⇨ **YB**

10.4.4 PIECEWISE CURVES FOR CURRENT INPUTS

Where the full scale of the voltage sensor is represented internally as a full 0 to 100%, a current sensor is represented in slightly less than 50% of the full scale readable by the controller. This means that the 0 to 20 mA full scale sensor reading range is mapped to the range of 0 to slightly less than 50% of the full scale readable by the controller. Calculating the Piecewise Curve for a current input is the same as for the

resistive input, except that you would instead apply a different formula to calculate the percentage of full scale. Here, 1 mA read in from the sensor corresponds to 2.49% of the full scale. You can simply multiply the current value from the sensor's characteristic response, or you can use the following formula to calculate the percentage of full scale:

$$\%\ \text{Full Scale} = \frac{\text{Current(mA)} \times 249}{100}$$

The values for **Y1** through **YB** are entered in Engineering Units in exactly the same way as for the voltage sensor.

10.4.5 PIECEWISE CURVES FOR RESISTANCE INPUTS

Like the current sensor, the resistance sensor is represented inside the controller a fraction of the full scale range possible in the controller. The 0 to 250 kΩ resistance range is represented internally as 0 to slightly less than 25% of the full scale readable by the controller. Calculating the a Piecewise Curve for a resistive input is also slightly different than for the voltage or current sensors because the controller measures the voltage drop across the input and that response is inherently non-linear. Because of this, there is no simple multiplier that can be used to convert resistance to full scale percentage as for the voltage or current sensors. Instead, you will have to use the following equation:

$$\%\ \text{Full Scale} = 25 \times \frac{\text{Resistance}(\Omega)}{\text{Resistance}(\Omega) + 20000}$$

The calculated full scale values are then entered into **X1** through **XB**. The values for **Y1** through **YB** are entered in Engineering Units in exactly the same way as for the voltage and current sensors.

SECTION 11: OUTPUTS SETUP

This section provides general information regarding setup of Analog Output and Binary Output objects. The MatrixBBC supports up to a maximum of 72 Analog Output and 72 Digital Output objects.

IN THIS SECTION

11.1 OUTPUTS OVERVIEW

NB-GPC product models support both Analog and Binary Outputs for direct equipment control using either on-board I/O or expansion modules via STATbus. While the total amount of outputs per GPC will depend on the model number, setup and configuration of outputs is exactly the same across the product family. The table below provides information on-board I/O support, as well as expansion support.

11.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

11.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The GPC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used object. This allows you not only to keep track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

11.1.1.2 ENABLE ALARMING WHEN NEEDED

All Output objects optionally support alarming. When alarming is disabled (EA) Enable Alarming = Disabled (0), less properties will be displayed in NB-Pro, allowing the objects to be interpreted easier during programming.

11.2 ANALOG OUTPUTS

Analog Outputs are used to provide proportional control signals in either 0-10vdc or 0-20mA output form.

11.2.1 CREATING ANALOG OUTPUTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Analog Output objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Analog Output object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MM) Max Analog Outputs** property. By default, this value is set to 0, indicating no Analog Output objects exist.
3. Write the number of total Analog Output objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Analog Outputs, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Analog Output objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

11.2.2 CONFIGURING MINIMUM AND MAXIMUM THRESHOLDS

Analog Outputs can be software configured to accept a minimum or maximum value. Additionally, the output itself can be mathematically scaled to establish a percentage of the maximum voltage or current.

The **(MN) Minimum Scaled Value** and **(MX) Maximum Scaled Value** properties specify the minimum and maximum scaled values for the outputs, expressed as a percentage of the full scale. **min-pres-value** and **max-pres-value** are used to specify the display range for the **present-value**.

For example, if the **present-value** is to be displayed as a percentage (0-100%) of a 10 VDC output range, set **min-pres-value** to 0 and **max-pres-value** to 100 (a display range of 0%-100% of full scale). Then set **(MN) Minimum Scaled Value** to 0.0 and **(MX) Maximum Scaled Value** to 100.0, so that when **present-value** = 0 represents 0.0% of the output range and **present-value** = 100 represents 100.0% of the output range.

11.2.2.1 2-10VDC SCALING

If the output device in the previous example only operated from 2-10 V instead of 0-10 V, you would simply change the value of **(MN) Minimum Scaled Value** to be 20.0 because 2 V is 20% of the 10 V maximum. Everything else from the previous example would remain the same.

11.2.3 CONFIGURING ALARM/EVENT NOTIFICATIONS

Analog Outputs can be configured to support alarm/event notifications. To enable alarming for an Analog Output, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Analog Output objects can be configured to trigger one of the two following conditions:

. Low Limit - occurs when **present-value** is less than the value specified in **low-limit**.
. High Limit - occurs when **present-value** is greater than the value specified in **high-limit**.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.

3. Configure **limit-enable** to enable low-limit or high-limit alarming. This is accomplished by placing a check into each associated limit type.
4. Configure event-enable to have the object send alarms for how alarms transition. For example, if you wish to have the GPC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
5. Configure your **high-limit** and **low-limit** properties accordingly.
6. Configure the **time-delay** and **deadband** properties. The time-delay property defines a threshold of time (in seconds) where the **present-value** must exceed one of the limit properties in order for an alarm/event condition to be considered. The **deadband** property defines an offset from **low-limit** or **high-limit** that must be met in order for an alarm/event condition to be considered. For example, if **high-limit** = 75.0, **deadband** = 2.0, and **time-delay** = 5, the **present-value** must exceed 77.0 for at least 5 seconds before an alarm/event condition is considered.

11.2.4 AUTOSTUFF CONFIGURATION

AutoStuff is used to allow the Analog Output object to control based on another value within the NB-GPC. In previous generations of the GPC, Analog Outputs were directly controlled by PID Control Loops or custom SPL. Using AutoStuff, you can define what source will control the Analog Output by defining an object-property reference. This results in the Analog Output “pulling in” the value of the referenced object-property and “stuffing it” into priority array.

AutoStuff consists of configuring the following three properties:

. **(O1) AutoStuff Input Object** - specifies the object ID portion of an object-property reference where the GPC will receive a control value from.
. **(P1) AutoStuff Input Property** - specifies the property of an object-property reference where the GPC will receive a control value from.
. **(Q1) AutoStuff Mode/Priority** - specifies the level of priority-array that the AutoStuff function will use within the Analog Output. A value of 255 disables AutoStuff.

In the example shown below, Analog Output 1 is configured to pull and stuff the present-value of a PID Loop.

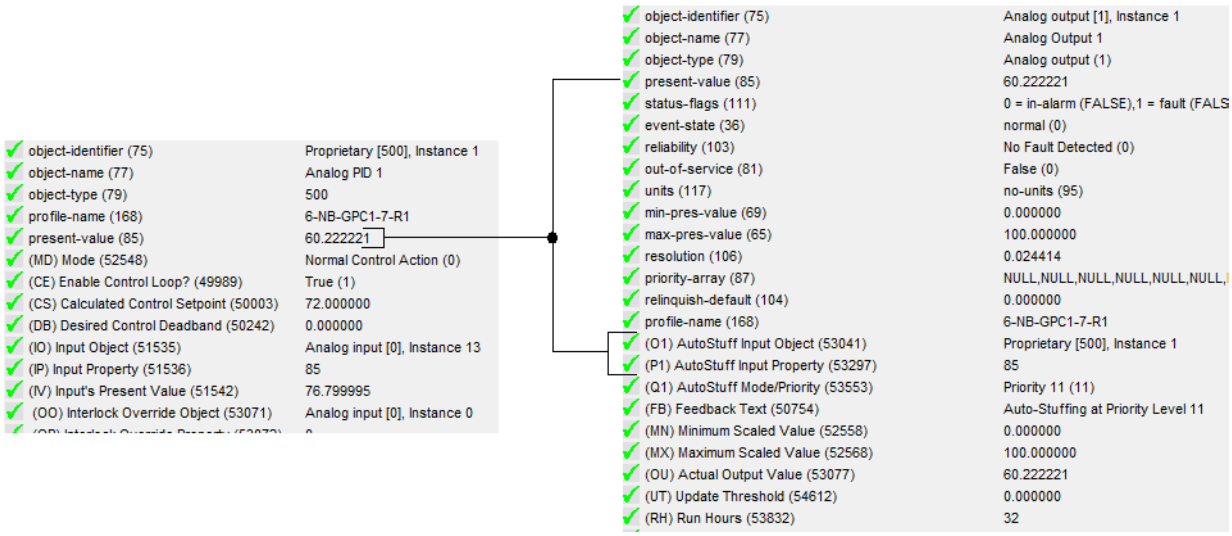

Figure 11-1 Auto-Stuff Example - Analog PID 1 to Analog Output 1

11.2.5 OTHER LOGIC PROPERTIES

Analog Output objects include some additional logic properties which may be useful when programming applications. These properties are explained below

11.2.5.1 ACTUAL OUTPUT VOLTAGE AND ACTUAL OUTPUT CURRENT

(OV) Actual Output Voltage and **(OC) Actual Output Current** defines the actual output value being sent by the GPC controller. This value can be cross-referenced to the present-value of the Analog Output object for I/O troubleshooting.

11.2.5.2 UPDATE THRESHOLD

(UT) Update Threshold defines how often (in seconds) the GPC updates the actual output. By default, this value is set to 0.0, which commands the GPC to update the output immediately.

11.2.5.3 RUN HOURS

(RH) Run Hours specifies how many hours the Analog Output has been outputting an actual signal.

11.3 BINARY OUTPUTS

Binary Outputs are used to provide on/off type control signals through use of on-board triacs or expansion outputs which can be either triac or relay based.

11.3.1 CREATING BINARY OUTPUTS IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. Binary Output objects are created by the technician when necessary, and are done in a dynamic manner.

To create an Analog Output object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the **(MN) Max Binary Outputs** property. By default, this value is set to 0, indicating no Binary Output objects exist.
3. Write the number of total Binary Output objects you wish to have in the MatrixBBC. For example, if you wish to have 10 Binary Outputs, write a value of 10.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Binary Output objects, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

11.3.2 CONFIGURING MINIMUM OFF/ON TIMES

Binary Outputs can be programmed to maintain on/off signals through use of **minimum-off-time** and **minimum-on-time**. These properties define how long, in seconds, a binary output will maintain either remain off or on after it has been commanded to do so.

These minimum timers utilizes BACnet's Command Prioritization mechanism to control the outputs in this manner. By virtue of the standard, this process uses priority level six (6).

For example, when the output is commanded off by writing at a priority level of 7 through16, it will stay off for the minimum period of time specified in minimum-off-time. If an attempt is made to command the output on, the output will remain off until the time period has expired.

If the output has been commanded at a priority higher than level six (6), the output commanded at a lower priority will automatically turn on/off.

11.3.3 CONFIGURING POLARITY

The **polarity** property defines the relationship between the physical state of the output and the software state reflected by **present-value**. By default, **polarity** is set to Normal (0). In this state, when the physical state of the output is inactive, the **present-value** will indicate a value of inactive; and when the physical state of the output is active, the **present-value** will indicate a value of active. If the **polarity** were to be switched to Reverse (1), **present-value** will indicate an Inactive status when the physical output is active, whereas **present-value** would indicate an Active status when the physical output is inactive.

11.3.4 CONFIGURING STATE TEXTS

For ease of configuring advanced operator workstations, Binary Outputs contain state texts which can be used to define the state of a Binary Output when it is inactive or active. These state text properties, **inactive-text** and **active-text**, define the state text for the Binary Output. Each state text can be up to 32 characters in length.

11.3.5 CONFIGURING ALARM/EVENT NOTIFICATIONS

Binary Outputs can be configured to support alarm/event notifications. To enable alarming for an Binary Output, set **(EA) Enable Alarming** = True. Once configured, additional properties will become available that control the setup and configuration of how alarms/events are handled by the object.

Binary Output object alarm/events are triggered through defining the state in which the output is considered to be in alarm.

To enable one of the two alarm conditions mentioned above, perform the following:

1. Configure **notification-class** to determine which Notification Class object will route objects for the alarm. If you have configured Notification Class, Instance 0, then a value of 0 must be referenced.
2. Configure **notify-type** to determine whether the notification will be of an *Alarm* type or *Event* type.
3. Configure **event-enable** to have the object send alarms for how alarms transition. For example, if you wish to have the GPC send a notification when the object enters and exits the alarm thresholds, place a check into the “To-Normal” and “To-OffNormal” boxes.
4. Configure the **time-delay** property. The time-delay property defines a threshold of time (in seconds) where the **present-value** must maintain the value in order for an alarm/event condition to be considered.

11.3.6 AUTOSTUFF CONFIGURATION

AutoStuff is used to allow the Binary Output object to be controlled based on another value within the NB-GPC. In previous generations of the GPC, Binary Outputs were directly controlled by Thermostatic Control Loops or custom SPL. Using AutoStuff, you can define what process controls the Binary Output by defining an object-property reference. This results in the Binary Output “pulling in” the value of the referenced object-property and “stuffing it” into priority array.

AutoStuff consists of configuring the following three properties:

. **(O1) AutoStuff Input Object** - specifies the object ID portion of an object-property reference where the GPC will receive a control value from.
. **(P1) AutoStuff Input Property** - specifies the property of an object-property reference where the GPC will receive a control value from.
. **(Q1) AutoStuff Mode/Priority** - specifies the level of priority-array that the AutoStuff function will use within the Binary Output. A value of 255 disables AutoStuff.

The AutoStuff function also includes helpful status feedback for troubleshooting purposes. This status is provided through the **(AF) AutoStuff Feedback** property.

11.3.7 OTHER LOGIC PROPERTIES

Analog Output objects include some additional logic properties which may be useful when programming applications. These properties are explained below.

11.3.7.1 ACTUAL OUTPUT STATE

(OU) Actual Output State defines the physical output value being sent by the GPC controller. This value can be cross-referenced to the present-value of the Binary Output object for I/O troubleshooting.

11.3.7.2 PULSE WIDTH

(PW) Pulse Width when Output Is On defines how often (in seconds) the Binary Output will pulse between on and on states to perform pulse width control when the output is driven to be active.

11.3.7.3 RUN HOURS

(RH) Run Hours specifies how many hours the Analog Output has been outputting an actual signal.

SECTION 12: CONTROL LOOPS

This section provides information regarding control loop objects that reside in the MatrixBBC, including Analog PID Control Loops, Pulse Pair PID Control Loops, and Thermostatic Control Loops. The MatrixBBC supports a maximum of 64 objects per control loop type.

IN THIS SECTION

12.1 CONTROL LOOPS OVERVIEW

The MatrixBBC provides the ability to create loop objects which can be setup to provide control to outputs that may be connected to the on-board STATbus port. Control Loops can be programmed to directly affect the status of analog or binary output or even a software parameter. By nature, all control loops are software based and are not internally linked to any specific hardware output (linking is achieved by using Netmaps, Remaps, or the AutoStuff feature of Output objects).

Three types of control loop objects are provided within the MatrixBBC, including:
1. Analog PID - uses standard PID control to provide an analog signal value.
2. Pulse-Pair - uses standard PID control to provide on/off floating point control signals.
3. Thermostatic - uses enhanced boolean logic (TSTAT logic) to provide an on/off control signal.

An explanation of each control loop object type, along with notes on setup and configuration are provided in the sections below.

12.1.1 PROGRAMMING CONCEPTS AND TECHNIQUES

To enhance your programming experience, the following are a few helpful concepts and techniques to keep in mind when using these objects.

12.1.1.1 MAKE THE OBJECT-NAME UNIQUE

The MatrixBBC supports the ability to allow each object's name to be assigned a custom value. By default, the software uses generic names for objects. For ease of programming and flow, it is strongly recommended that you change the object-name of any used control loop objects. This allows you not only to keep better track of which objects have been used, but also allows you to easily troubleshoot your linked logic.

12.2 ANALOG OUTPUT CONTROL LOOPS

Proportional + Integral + Derivative (PID) represents a method of control that controls equipment according to a set point in proportion to the value of a measured variable. It accounts for the amount of error (difference between the measured variable and the set point) and the continued presence of error.

12.2.1 BASIC SETUP

To initially use a control loop, you must first setup and configure basic properties of the control loop.

12.2.1.1 CONTROL SIGN AND OUTPUT LIMITS

The **(SG) Control Sign** property specifies the control action for the control loop. When **SG** = 0 (normal), a positive error causes an increase in output. When **SG** = 1 (reverse), a positive error causes a decrease in output. This point determines the response of the loop output to the kind of error. If the output action is to be increased (toward max) when the error is positive, set **SG** to normal (0). If the output action is to be decreased (toward min) for positive error, set **SG** to reverse (1). (Property **SG** is also used during schedule control)

The minimum and maximum limits of the control loop may also be configured if desired. The **(OL) Minimum Output Limit** and **(OH) Maximum Output Limit** properties define the minimum and maximum limits of the output range for the control loop. The **present-value** will be scaled between the limits you have defined.

12.2.1.2 MEASURED VARIABLE CONFIGURATION

The **(IO) Input Object** and **(IP) Input Property** properties specify the object and property to be used as the loop measured variable. It specifies the input to be used for the control loop's measured variable. Any object property within the MatrixBBC can be used as the measured variable by configuring these properties appropriately.

When the control loop is enabled, **(IV) Input's Present Value** will reflect the current value of the loop measured variable.

12.2.1.3 SETPOINT CONFIGURATION

Each control loop contains four setpoint properties, defining the control setpoint that is used for a specific schedule mode. The setpoint is expressed in the same kind of measurement units (engineering units) that the measured variable uses (e.g., degrees, cfm, inches of WC, etc.). This value is used with the setup/setback value and any reset to calculate the actual setpoint used to control the loop. If you intend to configure your MatrixBBC to perform four-mode scheduling (see Section 8 - Scheduling for more details), you may define a setpoint for each occupancy mode (Warmup, Occupied, Unoccupied, Night Setback).

- **(US) Unoccupied Setpoint** defines the control setpoint for Unoccupied periods.
- **(OS) Occupied Setpoint** defines the control setpoint for Occupied periods.
- **(WS) Warmup Setpoint** defines the control setpoint for Warmup periods.
- **(NS) Night Setback Setpoint** defines the control setpoint for Night Setback periods.

When the control loop is later enabled (using **(CE) Enable Control Loop**), the **(CS) Calculated Control Setpoint** will reflect which setpoint is currently active.

The **(SM) Schedules to Follow** property allows control loop to reference a schedule and transition through programmed set points appropriately. Each bit in **SM** corresponds to one of up to 10 available schedules. These bits are summarized in Table 12-1.

Table 12-1 : Schedules to Follow

SM bit	Schedule
0	Schedule 1
1	Schedule 2
2	Schedule 3
3	Schedule 4
4	Schedule 5
5	Schedule 6
6	Schedule 7
7	Schedule 8
8	Schedule 9
9	Schedule 10

If you do not want to use schedule control, set all of the bits in **SM** to 0, and configure only the **(OS) Occupied Setpoint** for control.

12.2.2 PROPORTIONAL CONTROL SETUP

The **(PB) Proportional Control Band** specifies the input variable range over which the output value is proportional to the error value (i.e., changes in the measured variable result in proportional changes in the output signal). The proportional band is centered around setpoint for the loop. This point is expressed in the same kind of measurement units (engineering units) that the measured variable uses. For example: degrees, cfm, inches of WC.

To determine **PB**, first decide how closely the loop must control to the setpoint. For instance, if the setpoint is 72°F, then an acceptable control range might be within two degrees of the setpoint. This control range can be expressed as a band centered on the setpoint: from 70° to 74°, or 4 degrees *proportional band* (**PB**). Refer to Figure 12-1 and Figure 12-2.

For normal acting control loops (see Figure 12-1), the **(PO) Percent Output** property is set to maximum output when the input variable equals the setpoint plus half of the proportional band (**CS** + **PB**/2). The percent output is set to minimum output when the input variable equals the setpoint minus half of the proportional band (**CS** - **PB**/2). These associations are reversed for reverse acting control loops. **PO** will be midway between minimum and maximum output when the measured variable is equal to the control setpoint **CS**. The opposite would be true for reverse acting control loop as shown in Figure 12-2.

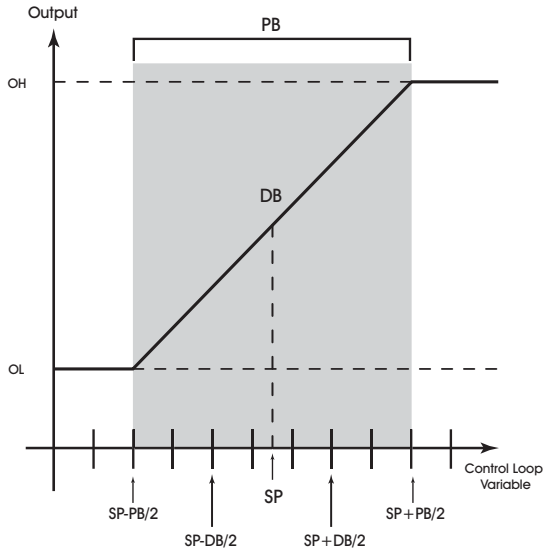

Figure 12-1: Proportional Band for Normal Acting Control (SG = 0)

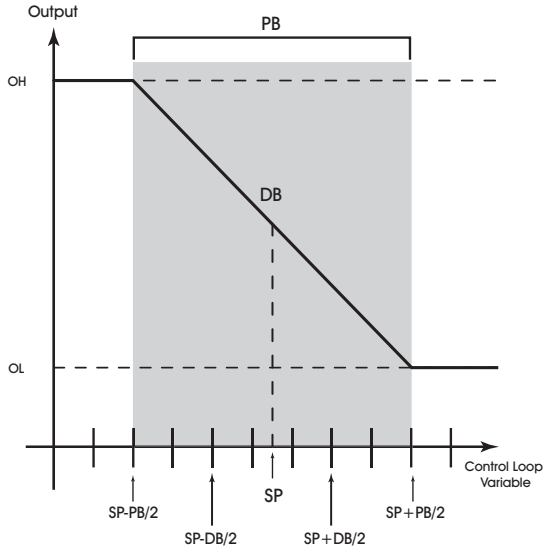

*Figure 12-2: Proportional Band for Reverse Acting Control (**SG** = 1)*

Proportional only control produces cycling, and its performance changes when the measured environment changes. The way to eliminate cycling and to compensate for load changes is to use *integral* action, the "I" part for PID control.

12.2.3 DEADBAND CONFIGURATION

The **(DB) Desired Control Deadband** property specifies the deadband within the proportional control band in which the output remains constant at a point midway between maximum output and minimum output. By specifying a deadband that is greater than or equal to the resolution of the loop measured variable, you eliminate the possibility of cycling around the setpoint. The value of the deadband should

never exceed the proportional band. If the deadband is greater than the proportional band, then the control loop will not have proportional control.

The deadband is used to specify an input variable range within the proportional band. The size of the deadband should be based on the loop measured variable. When the value of the measured variable is within this dead band, the output signal remains constant at the midpoint of the minimum/maximum range.

The point that deadband is centered on tone of the four defined set points to create the actual control dead band. When the value of the loop measured variable is within ±**DB**/2 of the setpoint, the loop assumes that it has reached the setpoint. Refer to Figure 12-3.

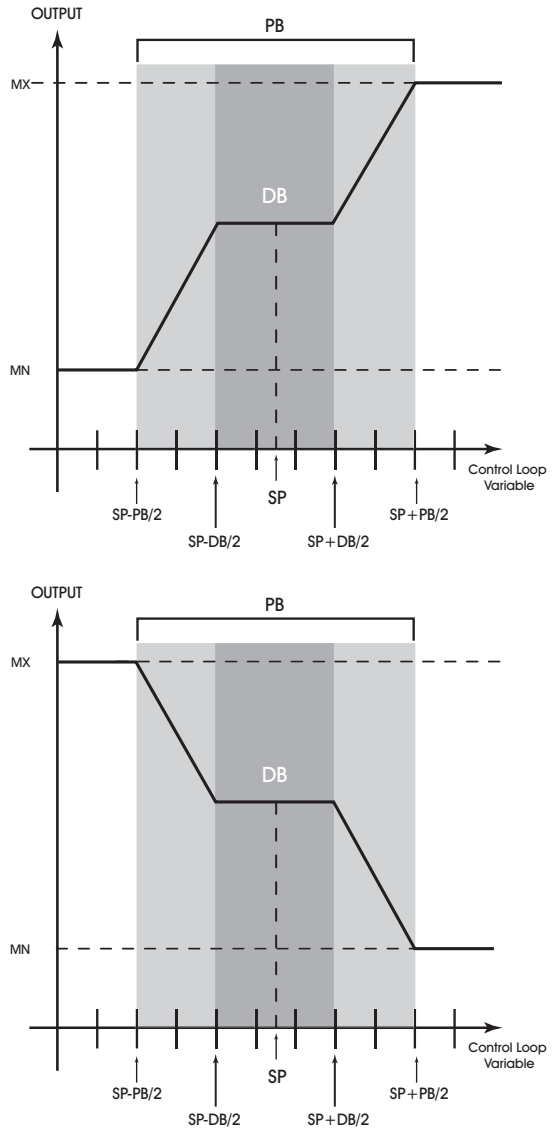

Figure 12-3: Normal Acting (above) and Reverse Acting (below), Proportional Control Output Response Showing a Dead Band Centered Around the Setpoint (SP)

By entering a value in deadband that is greater than the resolution of the measured variable sensor, you create a deadband that allows the loop to effectively reach setpoint. Be sure that the deadband selected does not exceed the size of the proportional band.

CAUTION

*Never change **DB** to a value greater than half of the proportional band **PB**. Doing so will eliminate the effects of PID control, resulting in on/off control.*

12.2.4 RESET CONTROL SETUP

The **(MR) Maximum Amount to Reset Setpoint** property specifies the maximum amount to reset the loop setpoint (**SP**) based on when reset is being used. Property **CS** takes into account the use of the maximum reset specified in **MR**.

The **(RC) Reset Variable** and **(RA) Reset Attribute** specify the object and property to be used as the Reset Variable.

The **(RS) Reset Setpoint** property specifies the value at which the reset action begins. When the value of the reset variable exceeds **RS**, reset action will be used in determining the calculated setpoint. Just as **SP** is the proportional control setpoint for the measured variable specified in **IC** and **IA**, **RS** is the reset control setpoint for the value of the reset variable selected by **RC** and **RA**.

The **(RL) Reset Limit** property specifies the value at which maximum reset is used. When the value of the reset variable is equal to **RL**, the maximum reset (**MR**) is used in determining the calculated setpoint (**CS**).

The relationship between **RL** and **RS**, as well as the sign (+ or –) of **MR**, determines how changes in the reset variable specified by **RC** and **RA** affect the calculated control setpoint **CS**. Refer to Figure 12-4.

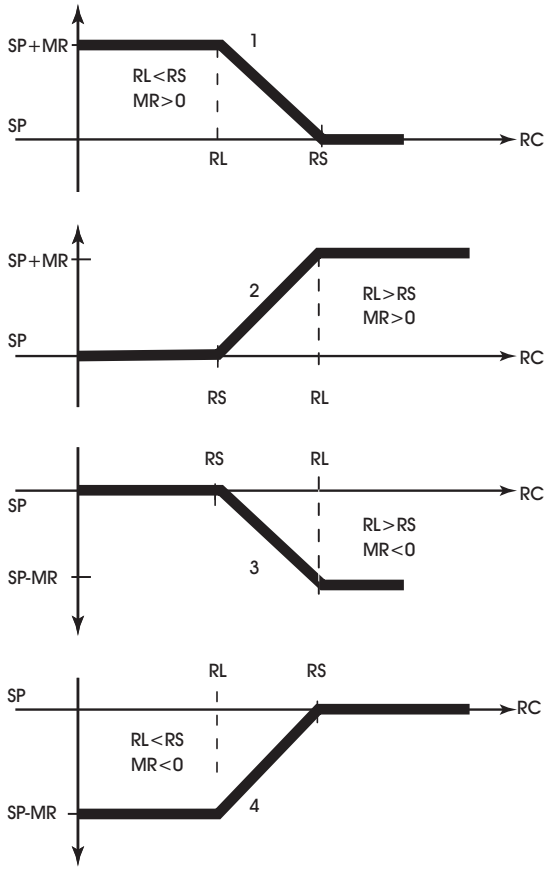

Figure 12-4: Four Forms of Reset Action

With appropriate values entered for these properties, the loop will provide simple closed loop feedback proportional control. This means that the actual measured performance of the control (from the measured variable input) is fed back to the controller and is compared with the effective setpoint for the loop. Any difference between the actual value of the measured variable and effective setpoint values is called error.

One problem with proportional only control is the changes in loop performance that occur when the condition being measured by the input sensor changes (e.g., the measured temperature changes when a door is opened and the room or space is flooded with cold air). As the loop environment changes, the proportional only control loop begins to cycle around an offset from the setpoint. Figure 12-5 illustrates the performance of a typical loop under proportional only control.

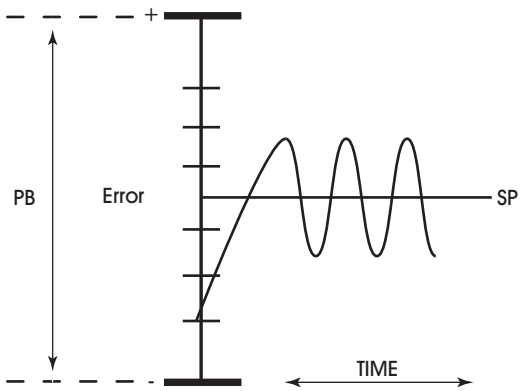

Figure 12-5: Proportional Only Control

Rather than responding exclusively to the loop error from moment to moment as is the case with proportional action, integral action is based on a summation of the error that has occurred over some period. This error sum is used to reset, or modify, the response of the control loop (output) based on a running average of the error. The amount of time over which the error averaging is accumulated is called the *reset period*.

The **(RP) Reset Period** property specifies the reset period (in seconds) over which the error history is accumulated. If **RP** = 10 seconds with a constant error of 2.0, then the error history would increase by 0.2 every second. In five seconds, the error history would be 1.0. At the end of ten seconds, the error history would be 2.0. Setting **RP** to 0 disables integral action making the loop proportional only. The longer **RP** is, the less effect it has on the control response. Figure 12-6 shows the response of a typical control loop when integral action is used in addition to proportional action (PI control). A value of 0 disables the reset period.

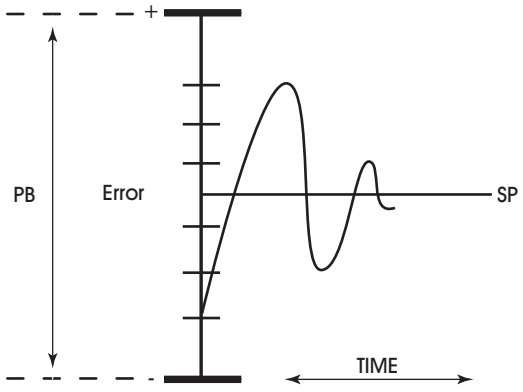

Figure 12-6: Proportional + Integral (PI) Control

At the start-up of the loop or following a change in setpoint (see Figure 12-6), the error is fairly large. Proportional action causes the loop output to accelerate toward the setpoint. However by the time the loop response reaches the setpoint value, it has gained inertia from the preceding proportional action. This causes the loop to overshoot the setpoint. As the loop exceeds the setpoint moving toward its first peak,

the error sum is accumulating. This slows down the acceleration, eventually causing the downturn in response.

As the error falls and then drops below the setpoint, the error sum will be reduced because now the error is in the opposite direction. The cycle continues in diminishing peaks until it finally converges at the setpoint as shown in Figure 12-6.

The proportional control action of the loop has a major effect on integral action. Increasing **PB** results in a smaller integral effect for a given value of **RP**. In general, decreasing the proportional band, **PB**, will increase the magnitude of the changes in **PO**.

Several important factors may not be obvious to inexperienced users of these DDC techniques.

First, whenever the error falls outside of the proportional band - that is, ±**PB**/2 from the setpoint, two important things happen: the controller's output is fully pegged in the appropriate direction, and the error sum stops accumulating. The control produces its maximum output because it must bring the error within the proportional band again. The error sum stops accumulating so that it does not "wind up" a massive error sum that would take many control cycles to dissipate. This feature is called anti-reset windup.

Anti reset windup also makes the loop recover quickly when it reenters the proportional band. Another feature of anti reset windup is that the error history is limited to **PB**/2 because that is all that required to produce maximum output. Additional error accumulation would only slow down loop recovery.

To quicken loop response while eliminating overshoot, derivative action must be taken. Derivative action takes into account the rate of change in error and allows the loop to counter the effects of the error's rate of change on the control output. To find the change in error, subtract the current error (read every second by the PID loop) from the previous second's error. A percentage of this change (specified by **RT**) becomes the derivative contribution to the PID output.

The **(RT) Derivative Rate** property specifies a percentage of change in error that is to be used in calculating **PO**. The value is specified in percent per second. The point **RT** can have any value from 0.0 to 25.5%/second.

12.2.5 INTERLOCK SETUP

Interlocking allows the control loop to be enabled or disabled based on a status input. The status input is configured using properties **(OO) Interlock Override Object** and **(OP) Interlock Override Property**. When the defined input is a true signal (value of 1), the PID Control Loop will be disabled, else, remains active. The current value of the referenced interlock object property can be monitored from the PID Control Loop through property **(OV) Interlock Override Objects Present State**.

12.2.6 SOFT START SETUP

The **(SR) Soft Start Ramp** property specifies the maximum percentage change per minute for the associated output under he following conditions: when the controller is initially powered up or reset; upon transitions from unoccupied to occupied mode, upon cancellation of an interlock failure or fire condition, or when a control loop is initially enabled. These situations can cause the control loop to peg to 100% which can cause the output to spike and, in turn, could lead to equipment damage. To prevent this, the output will be limited to changing **SR** percent per minute.

12.2.7 STAT OVERRIDE OFFSET AND ADJUSTMENT

In order for a setpoint adjustment to be made from a STAT, the Universal Input that referenced a connected SmartSTAT must be linked to a control loop that adjusts the loop. Once linked, three properties are used to

establish adjustment parameters between the STAT and the control loop. Three properties control the setpoint offset, display the current position of the override offset, and displays the remaining time duration for adjustment. These properties **(MO) STAT Maximum Override Offset** specifies the degree offset adjusted each time a user presses the up or down arrow from a specific thermostat. **(CO) Stat Current Override Offset** displays the current adjust setpoint position of the linked STAT. Finally, **(OR) STAT Override Time Remaining** displays the amount of time left for the effective setpoint adjustment made from the linked STAT.

12.2.8 ENABLING THE CONTROL LOOP

The **(CE) Enable Control Loop?** property enables and disables the PID loop. When **CE** = 0, the loop output is not updated but may be set manually. When **CE** = 1, the loop output is updated by the PID control loop and the corresponding analog output is controlled.

The **(DL) Demand Load** property indicates the amount by which **CS** differs from the loop measured variable.

12.3 PULSE-PAIR PID CONTROL

Similar to an Analog PID Control Loop, Pulse Pair PID Control loops are used to control such devices as fans, pumps, and blowers. Each loop performs either PI or PID control while providing calibration and alarming functions.

In floating point control applications, each floating point control object controls the position of a motor actuator using two digital outputs (an increase output and a decrease output).

12.3.1 BASIC SETUP

To initially use a control loop, you must first setup and configure basic properties of the control loop.

12.3.1.1 CONTROL SIGN AND OUTPUT LIMITS

The **(SG) Control Sign** property specifies the control action for the control loop. When **SG** = 0 (normal), a positive error causes an increase in output. When **SG** = 1 (reverse), a positive error causes a decrease in output. This point determines the response of the loop output to the kind of error. If the output action is to be increased (toward max) when the error is positive, set **SG** to normal (0). If the output action is to be decreased (toward min) for positive error, set **SG** to reverse (1). (Property **SG** is also used during schedule control)

12.3.1.2 MEASURED VARIABLE CONFIGURATION

The **(IO) Input Object** and **(IP) Input Property** properties specify the object and property to be used as the loop measured variable. It specifies the input to be used for the control loop's measured variable. Any object property within the MatrixBBC can be used as the measured variable by configuring these properties appropriately.

When the control loop is enabled, **(IV) Input's Present Value** will reflect the current value of the loop measured variable.

12.3.1.3 SETPOINT CONFIGURATION

Each control loop contains four setpoint properties, defining the control setpoint that is used for a specific schedule mode. The setpoint is expressed in the same kind of measurement units (engineering units) that the measured variable uses (e.g., degrees, cfm, inches of WC, etc.). This value is used with the setup/ setback value and any reset to calculate the actual setpoint used to control the loop. If you intend to configure a MatrixBBC Schedule to perform four-mode scheduling (see Section 4 - Scheduling for more details), you may define a setpoint for each occupancy mode (Warmup, Occupied, Unoccupied, Night Setback).

- **(US) Unoccupied Setpoint** defines the control setpoint for Unoccupied periods.
- **(OS) Occupied Setpoint** defines the control setpoint for Occupied periods.
- **(WS) Warmup Setpoint** defines the control setpoint for Warmup periods.
- **(NS) Night Setback Setpoint** defines the control setpoint for Night Setback periods.

When the control loop is later enabled (using **(CE) Enable Control Loop**), the **(CS) Calculated Control Setpoint** will reflect which setpoint is currently active.

The **(SM) Schedules to Follow** property allows control loop to reference a schedule and transition through programmed set points appropriately. Each bit in **SM** corresponds to one of up to 10 available schedules. These bits are summarized in Table 12-1.

Table 12-2 : Schedules to Follow

SM bit	Schedule
0	Schedule 1
1	Schedule 2
2	Schedule 3
3	Schedule 4
4	Schedule 5
5	Schedule 6
6	Schedule 7
7	Schedule 8
8	Schedule 9
9	Schedule 10

If you do not want to use schedule control, set all of the bits in **SM** to 0, and configure only the **(OS) Occupied Setpoint** for control.

12.3.2 PROPORTIONAL CONTROL SETUP

The **(PB) Proportional Control Band** specifies the input variable range over which the output value is proportional to the error value (i.e., changes in the measured variable result in proportional changes in the output signal). The proportional band is centered around setpoint for the loop. This point is expressed in the same kind of measurement units (engineering units) that the measured variable uses—for example: degrees, cfm, inches of WC.

To determine **PB**, first decide how closely the loop must control to the setpoint. For instance, if the setpoint is 72°F, then an acceptable control range might be within two degrees of the setpoint. This control range can be expressed as a band centered on the setpoint: from 70° to 74°, or 4 degrees—the *proportional band* (**PB**). Refer to Figure 12-1 and Figure 12-2.

For normal acting control loops (see Figure 12-1), the **(PO) Percent Output** property is set to maximum output when the input variable equals the setpoint plus half of the proportional band (**CS** + **PB**/2). The percent output is set to minimum output when the input variable equals the setpoint minus half of the proportional band (**CS** - **PB**/2). These associations are reversed for reverse acting control loops. **PO** will be midway between minimum and maximum output when the measured variable is equal to the control setpoint **CS**. The opposite would be true for reverse acting control loop as shown in Figure 12-2.

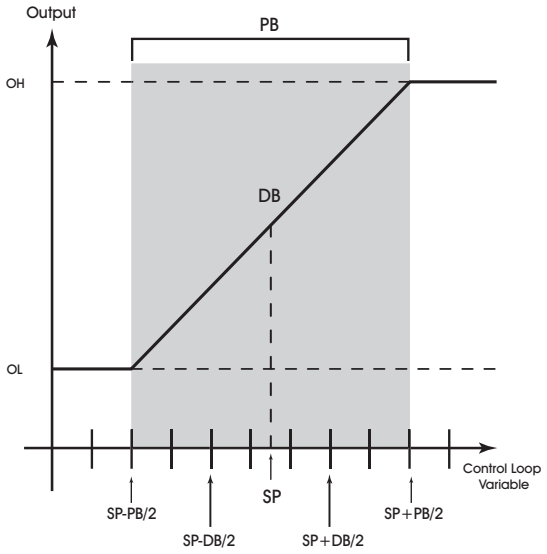

Figure 12-7: Proportional Band for Normal Acting Control (SG = 0)

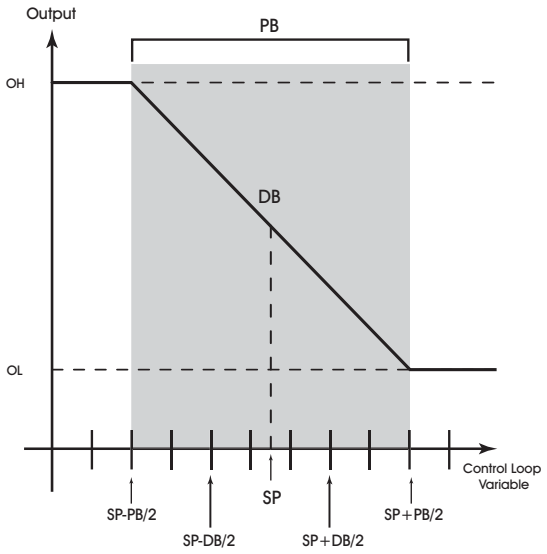

*Figure 12-8: Proportional Band for Reverse Acting Control (**SG** = 1)*

Proportional only control produces cycling, and its performance changes when the measured environment changes. The way to eliminate cycling and to compensate for load changes is to use *integral* action, the "I" part for PID control.

12.3.3 DEADBAND CONFIGURATION

The **(DB) Desired Control Deadband** property specifies the deadband within the proportional control band in which the output remains constant at a point midway between maximum output and minimum output. By specifying a deadband that is greater than or equal to the resolution of the loop measured variable, you eliminate the possibility of cycling around the setpoint. The value of the deadband should

never exceed the proportional band. If the deadband is greater than the proportional band, then the control loop will not have proportional control.

The deadband is used to specify an input variable range within the proportional band. The size of the deadband should be based on the loop measured variable. When the value of the measured variable is within this dead band, the output signal remains constant at the midpoint of the minimum/maximum range.

The point that deadband is centered on tone of the four defined set points to create the actual control dead band. When the value of the loop measured variable is within ±**DB**/2 of the setpoint, the loop assumes that it has reached the setpoint. Refer to Figure 12-3.

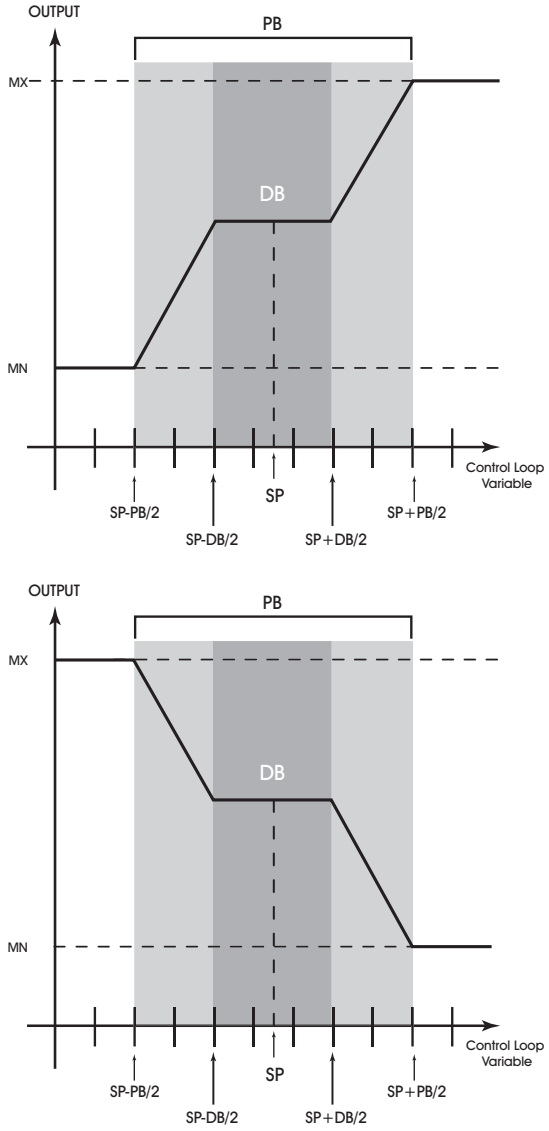

Figure 12-9: Normal Acting (above) and Reverse Acting (below), Proportional Control Output Response Showing a Dead Band Centered Around the Setpoint (SP)

By entering a value in deadband that is greater than the resolution of the measured variable sensor, you create a deadband that allows the loop to effectively reach setpoint. Be sure that the deadband selected does not exceed the size of the proportional band.

CAUTION

*Never change **DB** to a value greater than half of the proportional band **PB**. Doing so will eliminate the effects of PID control, resulting in on/off control.*

12.3.4 RESET CONTROL SETUP

The **(MR) Maximum Amount to Reset Setpoint** property specifies the maximum amount to reset the loop setpoint (**SP**) based on when reset is being used. Property **CS** takes into account the use of the maximum reset specified in **MR**.

The **(RC) Reset Variable** and **(RA) Reset Attribute** specify the object and property to be used as the Reset Variable.

The **(RS) Reset Setpoint** property specifies the value at which the reset action begins. When the value of the reset variable exceeds **RS**, reset action will be used in determining the calculated setpoint. Just as **SP** is the proportional control setpoint for the measured variable specified in **IC** and **IA**, **RS** is the reset control setpoint for the value of the reset variable selected by **RC** and **RA**.

The **(RL) Reset Limit** property specifies the value at which maximum reset is used. When the value of the reset variable is equal to **RL**, the maximum reset (**MR**) is used in determining the calculated setpoint (**CS**).

The relationship between **RL** and **RS**, as well as the sign (+ or –) of **MR**, determines how changes in the reset variable specified by **RC** and **RA** affect the calculated control setpoint **CS**. Refer to Figure 12-4. In the illustrations, references to ***SP*** generically refer to your currently used scheduled setpoint.

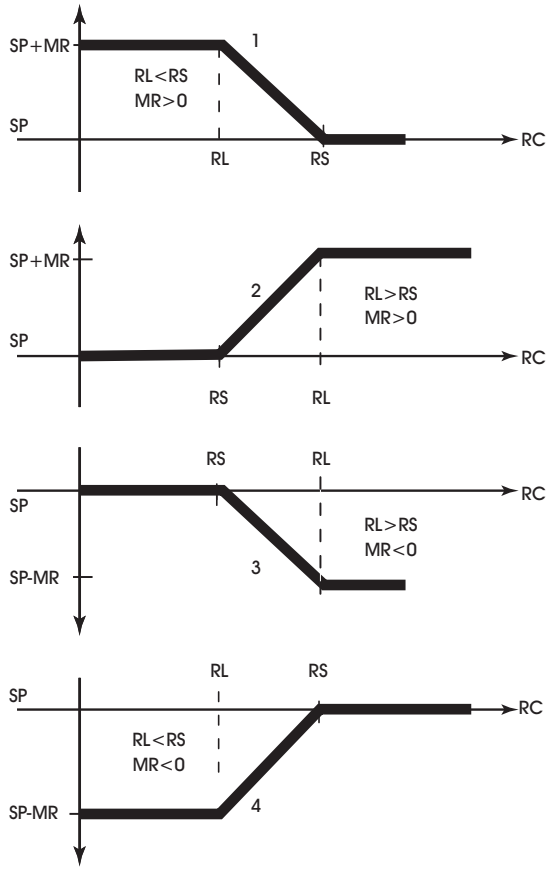

Figure 12-10: Four Forms of Reset Action

With appropriate values entered for these properties, the loop will provide simple closed loop feedback proportional control. This means that the actual measured performance of the control (from the measured variable input) is fed back to the controller and is compared with the effective setpoint for the loop. Any difference between the actual value of the measured variable and effective setpoint values is called error.

One problem with proportional only control is the changes in loop performance that occur when the condition being measured by the input sensor changes (e.g., the measured temperature changes when a door is opened and the room or space is flooded with cold air). As the loop environment changes, the proportional only control loop begins to cycle around an offset from the setpoint. Figure 12-5 illustrates the performance of a typical loop under proportional only control.

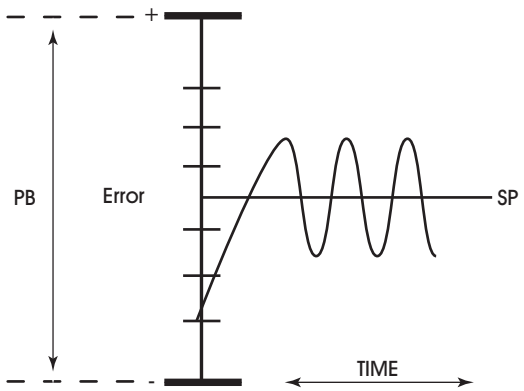

Figure 12-11: Proportional Only Control

Rather than responding exclusively to the loop error from moment to moment as is the case with proportional action, integral action is based on a summation of the error that has occurred over some period. This error sum is used to reset, or modify, the response of the control loop (output) based on a running average of the error. The amount of time over which the error averaging is accumulated is called the *reset period*.

The **(RP) Reset Period** property specifies the reset period (in seconds) over which the error history is accumulated. If **RP** = 10 seconds with a constant error of 2.0, then the error history would increase by 0.2 every second. In five seconds, the error history would be 1.0. At the end of ten seconds, the error history would be 2.0. Setting **RP** to 0 disables integral action making the loop proportional only. The longer **RP** is, the less effect it has on the control response. Figure 12-6 shows the response of a typical control loop when integral action is used in addition to proportional action (PI control). A value of 0 disables the reset period.

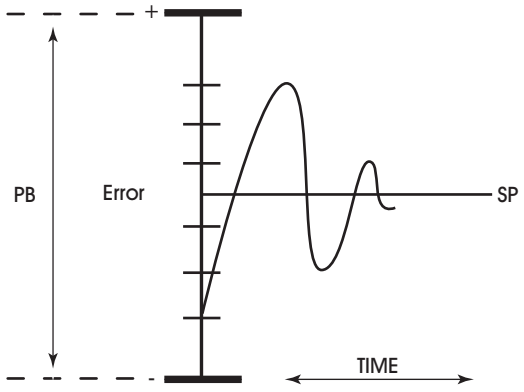

Figure 12-12: Proportional + Integral (PI) Control

At the start-up of the loop or following a change in setpoint (see Figure 12-6), the error is fairly large. Proportional action causes the loop output to accelerate toward the setpoint. However by the time the loop response reaches the setpoint value, it has gained inertia from the preceding proportional action. This causes the loop to overshoot the setpoint. As the loop exceeds the setpoint moving toward its first peak,

the error sum is accumulating. This slows down the acceleration, eventually causing the downturn in response.

As the error falls and then drops below the setpoint, the error sum will be reduced because now the error is in the opposite direction. The cycle continues in diminishing peaks until it finally converges at the setpoint as shown in Figure 12-6.

The proportional control action of the loop has a major effect on integral action. Increasing **PB** results in a smaller integral effect for a given value of **RP**. In general, decreasing the proportional band, **PB**, will increase the magnitude of the changes in **PO**.

Several important factors may not be obvious to inexperienced users of these DDC techniques.

First, whenever the error falls outside of the proportional band—that is, ±**PB**/2 from the setpoint, two important things happen: the controller's output is fully pegged in the appropriate direction, and the error sum stops accumulating. The control produces its maximum output because it must bring the error within the proportional band again. The error sum stops accumulating so that it does not "wind up" a massive error sum that would take many control cycles to dissipate. This feature is called anti reset windup.

Anti reset windup also makes the loop recover quickly when it reenters the proportional band. Another feature of anti reset windup is that the error history is limited to **PB**/2 because that is all that required to produce maximum output. Additional error accumulation would only slow down loop recovery.

To quicken loop response while eliminating overshoot, derivative action must be taken. Derivative action takes into account the rate of change in error and allows the loop to counter the effects of the error's rate of change on the control output. To find the change in error, subtract the current error (read every second by the PID loop) from the previous second's error. A percentage of this change (specified by **RT**) becomes the derivative contribution to the PID output.

The **(RT) Derivative Rate** property specifies a percentage of change in error that is to be used in calculating **PO**. The value is specified in percent per second. The point **RT** can have any value from 0.0 to 25.5%/second.

12.3.5 CALIBRATION

The **(TT) Travel Time** property is used to specify the total time in seconds (0 to 65,535 seconds) that it takes the motor actuator to go full stroke (from fully open to fully closed). **TT** is used to determine the current position (**CP**) of the motor. **TT** defaults to a value of 0 seconds.

CAUTION

*The travel time of a motor depends on the load that is applied to the motor. For accuracy, it is suggested that you determine **TT** when the motor is loaded. For spring loaded motors, the full stroke travel time from 0% to 100% may be different than the 100% to 0% travel time. You may choose to use the higher of the two travel times for **TT**. In this case, it is recommended that you perform regular calibrations on the motor.*

The actuator can be manually calibrated by enabling floating point control pair enable (**PE**=1), disabling PI control (**CE**=0) and setting **DP** to 0% or 100%. When the actuator is at the programmed position (after approximately **TT** seconds), set **CP** to 0% or 100% accordingly. Finally, be sure to return PI control (**CE**=1) if you want **DP** to be set automatically.

Floating point control loops can be calibrated automatically by the loop at programmable intervals. This is done using the recalibrate interval. The **(RI) Recalibration Interval** property specifies how often (if at all) the associated floating point control object is to be recalibrated.

RI is given in hours (0-255 hours). If **RI**=0, then recalibration of floating point control loops does not occur. If **RI**>0, recalibration of the associated floating point control loops occurs every **RI** hours.

The loop recalibrates the floating point control loops by driving the desired position (**DP**) to the fully closed position (0%) for the amount of time specified in the travel time property (**TT**). The loop then sets the current position to 0%, after which the recalibration is complete and the controller returns the desired position to its original value.

For floating point control objects, you can enable an automatic creep feature using property **CR**, the creep enable property. This feature is used to automatically calibrate the output when its desired position is either 0% or 100%. The automatic creep feature is performed in one of two ways: (1) the appropriate output is left on when the output signal is at 0% or 100%, or (2) the output is *creeped* (pulsed) at a rate of 1% per minute (the current position is set to 1% or 99%) when the output signal is at 0% or 100%. The value of the creep enable property (**CR**) selects the desired method.

These two methods of output correction (continuous on and automatic creep) are illustrated in Figure 7-13. This example shows a floating point control loop with a desired position of 100%.

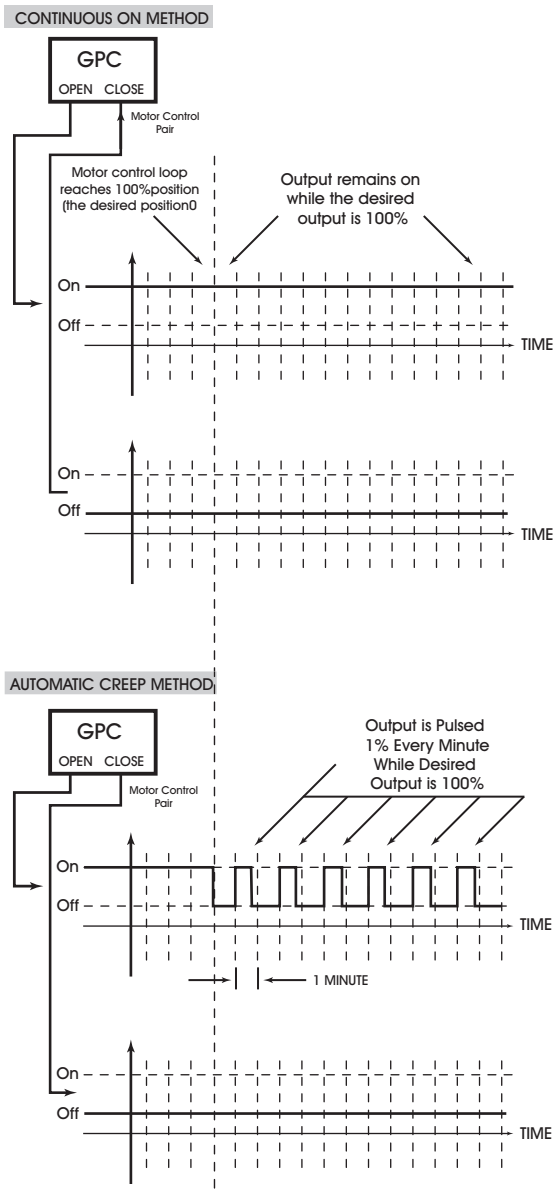

Figure 12-13 Auto Creep

12.3.6 STAT OVERRIDE OFFSET AND ADJUSTMENT

In order for a setpoint adjustment to be made from a STAT, the Universal Input that referenced a connected SmartSTAT must be linked to a control loop that adjusts the loop. Once linked, three properties are used to establish adjustment parameters between the STAT and the control loop. Three properties control the setpoint offset, display the current position of the override offset, and displays the remaining time duration for adjustment. These properties **(MO) STAT Maximum Override Offset** specifies the degree offset adjusted each time a user presses the up or down arrow from a specific thermostat. **(CO) Stat Current Override Offset** displays the current adjust setpoint position of the linked STAT. Finally, **(OR) STAT Override Time Remaining** displays the amount of time left for the effective setpoint adjustment made from the linked STAT.

12.3.7 ENABLING THE CONTROL LOOP

The **(CE) Enable Control Loop?** property enables and disables the loop. When **CE** = 0, the loop output is not updated but may be set manually. When **CE** = 1, the loop output is updated by the PID control loop and the corresponding analog output is controlled.

If the **(DP) Desired Position** of the motor is greater than the current position, the controller will drive the motor open by turning on the "increase" output for a calculated period of time. If the desired position is less than the current position, the controller will drive the motor closed by turning on the "decrease" output for a calculated period of time.

When the output increases, property **(O1) Output #1 (Load)** will output a true signal. When the output decreases, **(O2) Output #2 (Unload)** will output a true signal. These properties will be referenced by the AutoStuff feature of Binary Output objects to drive output signals.

CAUTION

Properties O1 and O2 must be tied directly to Binary Output objects using the output's AutoStuff feature and not a Remap or Netmap object. These properties may pulse on and off rapidly.

The desired position of the floating point control object can be set manually or calculated automatically by the PI algorithm. The automatic floating point control algorithm operates as follows. When the value of the selected measured variable is within the control loop's deadband, no control action is taken by the PI loop. When the value of the measured variable is outside the deadband, but within a programmable proportional band, the output is modulated using PI control according to the setpoint of the control loop. When the value of the measured variable is outside the deadband and beyond (either above or below) the proportional band, the output is set to either 0% or 100%, as appropriate.

12.4 THERMOSTATIC CONTROL

Thermostatic Control Loops are used to provide effective on/off binary control. When thermostatic control is enabled, the present-value can be used to control binary outputs or software values based on user-defined set points. By calculating a control setpoint, which takes into account different set points during warmup, unoccupied and night setback periods, and comparing it with the measured variable, the loop can determine the output value necessary to maintain the desired setpoint. The control loop can enforce a control deadband to prevent hysteresis and can be configured to operate based on one or multiple pre-defined schedule to allow the loop to differentiate setpoint control.

12.4.1 BASIC SETUP

To initially use a control loop, you must first setup and configure basic properties of the control loop.

12.4.1.1 MODE SETUP

The **(MD) Mode** defines the control sign of the thermostatic control loop. Two mode are available to apply a seasonal setup/setback setpoint for applications that require seasonal control (such as Fancoil units). There are four options available for Thermostatic Control loops:

. Heating in Winter (Else Off) - Provides heating control, independent of the current season.
. Heating in Winter (Else Seasonal Setback) - Provides heating control with integrated seasonal set-back.
. Cooling in Summer (Else Off) - Provides cooling control, independent of the current season.
. Cooling in Summer (Else Seasonal Setback) - Provides cooling control with integrated seasonal set-back.

12.4.1.2 MEASURED VARIABLE CONFIGURATION

The **(IO) Input Object** and **(IP) Input Property** properties specify the object and property to be used as the loop measured variable. It specifies the input to be used for the control loop's measured variable. Any object property within the MatrixBBC can be used as the measured variable by configuring these properties appropriately.

When the control loop is enabled, **(IV) Input's Present Value** will reflect the current value of the loop measured variable.

12.4.1.3 SETPOINT CONFIGURATION

Each control loop contains four setpoint properties, defining the control setpoint that is used for a specific schedule mode. The setpoint is expressed in the same kind of measurement units (engineering units) that the measured variable uses (e.g., degrees, cfm, inches of WC, etc.). This value is used with the setup/setback value and any reset to calculate the actual setpoint used to control the loop. If you intend to configure your MatrixBBC to perform four-mode scheduling (see Section 8 - Scheduling for more details), you may define a setpoint for each occupancy mode (Warmup, Occupied, Unoccupied, Night Setback).

. **(US) Unoccupied Setpoint** defines the control setpoint for Unoccupied periods.
. **(OS) Occupied Setpoint** defines the control setpoint for Occupied periods.
. **(WS) Warmup Setpoint** defines the control setpoint for Warmup periods.
. **(NS) Night Setback Setpoint** defines the control setpoint for Night Setback periods.
. **(SO) Seasonal Setup Setback Setpoint** defines the amount of setback applied to the set points listed above when the control loop is in the off.

When the control loop is later enabled (using **(CE) Enable Control Loop**), the **(CS) Calculated Control Setpoint** will reflect which setpoint is currently active.

The **(SM) Schedules to Follow** property allows control loop to reference a schedule and transition through programmed set points appropriately. Each bit in **SM** corresponds to one of up to 10 available schedules. These bits are summarized in Table 12-1.

Table 12-3 : Schedules to Follow

SM bit	Schedule
0	Schedule 1
1	Schedule 2
2	Schedule 3
3	Schedule 4
4	Schedule 5
5	Schedule 6
6	Schedule 7
7	Schedule 8
8	Schedule 9
9	Schedule 10

If you do not want to use schedule control, disable all of the bits and configure only the **(OS) Occupied Setpoint** for control.

The **(SS) Season** property dictates the current season mode to the control loop. The season can be adjusted automatically by the Season object, or by directly writing to this property using logic.

12.4.2 CONFIGURING LOOP PARAMETERS

The **(DB) Desired Control DeadBand** property specifies a control deadband for the thermostatic control loop. For a normal action control, this specifies the amount by which the temperature must drop below the cooling setpoint before the output is de-energized (**SP**-**DB)**. For a reverse action control. this specifies the amount by which the temperature must rise above the heating setpoint before the output is de-energized (**SP**+**DB)**. This response is illustrated in Figure 12-14.

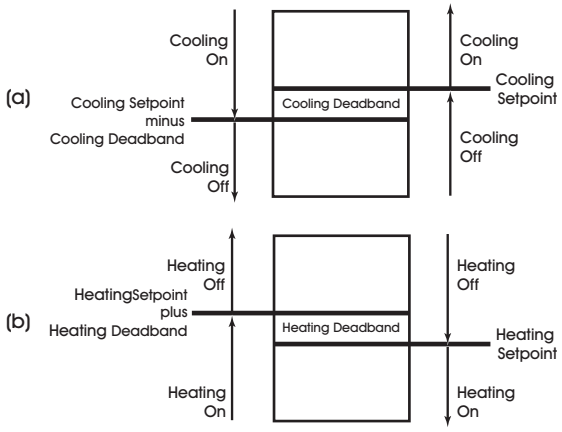

Figure 12-14: Deadband for a Normal Acting (a) and Reverse Acting (b) Thermostatic Control Loop

12.4.3 STAT OVERRIDE OFFSET AND ADJUSTMENT

In order for a setpoint adjustment to be made from a STAT, the Universal Input that referenced a connected SmartSTAT must be linked to a control loop that adjusts the loop. Once linked, three properties are used to establish adjustment parameters between the STAT and the control loop. Three properties control the setpoint offset, display the current position of the override offset, and displays the remaining time duration for adjustment. These properties **(MO) STAT Maximum Override Offset** specifies the degree offset adjusted each time a user presses the up or down arrow from a specific thermostat. **(CO) Stat Current Override Offset** displays the current adjust setpoint position of the linked STAT. Finally, **(OR) STAT Override Time Remaining** displays the amount of time left for the effective setpoint adjustment made from the linked STAT.

12.4.4 ENABLING THE CONTROL LOOP

The **(CE) Enable Control Loop?** property enables and disables the loop. When **CE** = 0, the loop output is not updated but may be set manually. When **CE** = 1, the loop output is updated by the PID control loop and the corresponding analog output is controlled.

Once enabled, the thermostatic control object will control based on the **(CS) Calculated Control Setpoint** property. This property represents the desired temperature in the area being controlled. The controller will begin with the setpoint value based on your current schedule mode (if schedules have been selected).

The value of **CS** is compared to the measured variable. The difference between **CS** and the measured variable will be stored in the **(DL) Demand Load** property. If the measured variable does not equal the calculated setpoint and is outside of the specified control deadband, then action will be taken to correct the measured variable.

The **present-value** property indicates the current state of the control loop. Control loop conditions are true (On) and false (typically Off).

SECTION 13: MISCELLANEOUS

This section describes control objects available under the Miscellaneous category of the MatrixBBC platform. The MatrixBBC supports a single Comm Status object, a single Season object, and a Single Manufacturing Object.

IN THIS SECTION

13.1 COMM STATUS

The Communication Status object allows event to be triggered within logic in the event of a BACnet MS/TP network failure. The object's **present-value** property indicates the current status of network communications. A value of 1 indicates good communications, whereas a value of 0 indicates bad communications.

13.1.1 CREATING THE COMM STATUS OBJECT IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. The Comm Status object is created by the technician when necessary, and are done in a dynamic manner.

To create a Comm Status object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the (MR) Max Comm Status property. By default, this value is set to 0, indicating no Comm Status object exists.
3. To create a Comm Status object, write a value of 1.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Comm Status object, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

13.1.2 COMMUNICATION STATUS OPTIONS

There are a few different types of fail scenarios that can be used to allow the GPC to consider a communication failure. The fail scenario is configured using property **(FM) Failure Mode Selection**. This feature begins operation after boot-up, based on the time value programmed in **(BD) Failure Mode Boot Delay in Second**.

Options for tracking communication failures include the following:

13.1.2.1 ALWAYS MARKED AS GOOD

This option will always consider communications in good standing. This option should be used if you intend to manually set the **present-value** using your own custom logic algorithm (which requires **out-of-service** to be set to true).

13.1.2.2 FAIL IF NOT PASSED A TOKEN

This option will set **present-value** to a value of 1 in the event that a BACnet MS/TP network token has not been passed to the GPC in the time period specific in **(FD) Failure Mode Delay Time in Seconds**.

13.1.2.3 FAIL IF NO DATA IS READ/WRITTEN

This option will set present-value to a value of 1 in the event that a read or write request has not been made to the GPC controller in the time period specific in **(FD) Failure Mode Delay Time in Seconds**.

13.2 SEASON

The Season object is used to declare the current season status for Thermostatic Control objects that are setup and configured to use seasonal setpoint setback.

13.2.1 CREATING THE SEASON OBJECT IN THE MATRIXBBC

By default, the MatrixBBC contains only a Device object for the purpose of initial network communications. The Season object is created by the technician when necessary, and are done in a dynamic manner.

To create a Season object, perform the following steps in NB-Pro:

1. Access the Device object of the MatrixBBC.
2. Find the (MQ) Max Season property. By default, this value is set to 0, indicating no Season object exists.
3. To create a Season object, write a value of 1.
4. Click *Update Value* in NB-Pro.

In order to see and view the newly created Season object, you must re-discover the object list of the MatrixBBC. To do this, select *Discovery>Discover Object List*, then click on the MatrixBBC in the Devices list in NB-Pro.

13.2.2 INDICATING THE CURRENT SEASON

By default, there is no internal control scheme in place that allows the GPC to determine the current seasonal status. This capability must be defined using your own logic. Once determined, write commands are made directly to the **present-value** property defines the current season setting, where:

. 0 = Summer
. 1 = Winter

13.2.3 CONTROLLING SEASONAL TSTAT LOOPS DIRECTLY

The Season object has the ability to automatically update the season property in Thermostatic Control loops. In this way, by simply writing a state to the present-value of the Season object, any logic driven by this value as well as any Thermostatic Control loops liked to via **(TC) T-STAT Objects Controlled By This Object** will automatically take the seasonal change into effect. The MatrixBBC will permit the first 32 Thermostatic Control Loops to be directly controlled. All other loops are intended for closed control, or pseudo-control.

13.2.4 OVERVIEW OF CURRENT SEASONAL STATES

The (TS) T-Stat Objects Current SS State property is a read-only property that reflects the current value of the (SS) Season property of each Thermostatic Control loop within the GPC.

13.3 MFG OBJECT

The Mfg. Object is a diagnostic object commonly used by AAM Technical Services to troubleshoot controller related issues relative to processor and memory. Much of the information provided here is purely diagnostic and does not carry any specific meaning for daily runtime usage. However, there are a few properties that are worthwhile to note.

13.3.1 (UT) UPTIME COUNTER IN SECONDS

The (UT) Uptime Counter in Seconds property indicates how many seconds that the GPC controller has been active since boot time. If you believe your device may be losing power, this is an excellent property to trend.

APPENDIX A: OBJECTS & PROPERTIES

The following tables contain listings of the BACnet objects and property assignments for the MatrixBBC. Each property is listed with its identifier number, data type, access code, storage, default value (if any) and a brief description of its functionality.

IN THIS SECTION

A.1 DEVICE OBJECT

NOTE

The Device object is represented in *NB*-Pro as follows:
Device Name xxxxxxxxxx
(where xxxxxxxxxx is the Unitary Controller serial number)

The instance must be a unique number from 0 to 4194302. By default, AAM sets the value in such a way that it is unique to AAM products based off the unit's serial number, however the user must ensure the device instance is unique on the job site's network.

Property	Identifier #	Data Type	Access	Default Value	Description
object_ identifier	75	BACnet ObjID	RW	Device (8), Instance	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	MatrixBBC	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Device (8)	indicates membership in a particular object type class.
system_status	112	BACnet ObjID	RO	0	indicates the current physical and logical status of the BACnet Device.
vendor_name	121	CharStr	RO	American Auto-Matrix	identifies the manufacturer of the BACnet Device.
vendor_ identifier	120	Unsigned	RO	-6	a unique vendor identification code, assigned by ASHRAE, which is used to distinguish proprietary extensions to the protocol.
model_name	70	CharStr	RO	NB-BBC1	indicates the vendor's name used to represent the model of the device.
firmware_ revision	44	CharStr	RO	*revision number*	indicates the level of firmware installed in the device.
application_ software_ version	12	CharStr	RO	*version number*	identifies the version of application software installed in the device.
protocol_ version	98	Unsigned	RO	1	indicates the version of the BACnet protocol supported by this BACnet Device.
protocol_ revision	139	Unsigned	RO	- 2	indicates the minor revision level of the BACnet standard.
protocol_ services_ supported	97	BACnet Services Supported	RO	-	indicates which standardized protocol services are supported by this device's protocol implementation.
protocol_ object_ types_ supported	96	BACnet Object Types Supported	RO	-	indicates which standardized object types are supported by this device's protocol implementation.
object_list	76	BACnet Array	RO	-	a list of each object within the device that is accessible through BACnet services.

Property	Identifier #	Data Type	Access	Default Value	Description
max_apdu_length_accepted	62	Unsigned	RO	480	specifies the maximum number of information frames the node may send before it must pass the token.
segmentation_supported	107	BACnet Segmentation	RO	Segmentation Both (0)	indicates whether the device supports segmentation of messages and, if so, whether it supports segmented transmission, reception, or both.
local_time	57	Time	RW	-	indicates the time of day to the best of the device's knowledge.
local_date	56	Date	RW	-	indicates the date to the best of the device's knowledge.
utc_offset	119	Integer	RW	240	indicates the number of minutes (-780 to +780) offset between local standard time and Universal Time Coordinated.
daylight_savings_status	24	Boolean	RW	0	indicates whether daylight savings time is in effect or not.
apdu_segment_timeout	10	Integer	RW	5000	indicates the timeout for segmented data packets.
apdu_timeout	11	Unsigned	RW	10000	indicates the amount of time, in milliseconds, between retransmissions of an APDU requiring acknowledgment for which no acknowledgment has been received.
number_of_apdu_retries	73	Unsigned	RW	3	indicates the maximum number of times that an APDU shall be retransmitted.
time_synchronization_recipients	116	List of BACnet recipients	RW	{}	a list of one device to which the device may automatically send a Time Synchronization request.
max_master	64	Unsigned	RO	127	specifies the highest possible address for master nodes and shall be less than or equal to 127.
max_info_frames	63	Unsigned	RO	5	specifies the maximum number of information frames the node may send before it must pass the token.
device_address_binding	30	List	RW	-	a list of the device addresses that will be used when the remote device must be accessed via a BACnet service request.
configuration_files	154	List	RO	-	defines the database file that is backed-up/restored.
database-revision	155	Integer	RO	-	defines the database revision of the device.
active-cov-subscriptions	152	List	RO	-	defines a list of active-COV subscriptions being maintained by the device.
last-restart-reason	196	Integer	RO	-	defines the reason for last restart.
time-of-device-restart	203	Date/ Time	RO	-	defines the last time and date stamp that the device restarted or started-up.
restart-notification-recipients	202	List	RO	-	defines a list of recipients that should receive a notification should the device restart.
time-synch-interval	204	Integer	RW	0	defines the amount of time, in minutes, that a time-synch message is sent by this device when configured to do so.

Property	Identifier #	Data Type	Access	Default Value	Description
align-intervals	193	Bool	RW	False (0)	defines if intervals for time-synchronizations are aligned to the top of the time interval.
interval-offset	195	Integer	RW	0	defines an interval offset (in minutes) to delay a time-sych.
last-restore-time	157	Date/ Time	RO	-	defines the last date and time the device was restored.
backup-failure-timeout	153	Integer	RW	-	defines the amount of time (in seconds) that backup failure will be considerd timed out.
slave-proxy-enable	172	List	RW	False|False	defines if slave-proxy capabilities are enabled for each RS-485 port of the device.
manual-slave-address-binding	170	List	RW	-	defines the list of MS/TP slaves that the device will send I-Am messages for.
slave-address-binding	171	List	RO	-	reflects active slaves that are under proxy representation by the MatrixBBC.
profile-name	168	CharStr	RO	6-NB-BBC1-11-R1	defines the profile name used by AAM Tools to correspond program files to a MatrixBBC controller.
FT	50772	Unsigned	RO	7	**Firmware Type** indicates which firmware is installed on the controller, reflecting MatrixBBC
OS	53075	Real	RO		**Kernel Version** indicates the version number of the kernel.
VE	54853	Real	RO		**Firmware Version** contains the version of the controller's firmware.
VO	56863	Real	RO		**I/O Firmware Version** contains the version of the controller's I/O firmware.
SR	54098	Unsigned	RO		**Flash Release Code** the release code of the firmware currently flashed on the controller, used primarily for technical support purposes.
CT	50004	Unsigned	RO	205	**Controller Type** factory-set controller type number for the controller.
FC	50755	Unsigned	RO		**Flash update count** indicates the number of times the controller has been flashed.
SN	54094	Unsigned	RO		**Serial Number** the factory-set serial number.
EM	50509	Boolean	RW	0	**Engineering Units** specifies the units to be used when returning values 0=English 1=Metric
PD	53316	Unsigned	RW	15	**Power-Up Delay** time delay (in seconds) that must elapse after the controller is reset before it begins control and alarming functions. 0=No delay 1-255=# of seconds
CP	50000	Unsigned	RO	-	**Baud Rate Line 1** reflects the baud rate for MS/TP Port 1.

Property	Identifier #	Data Type	Access	Default Value	Description
ID	51524	Unsigned	RO	-	**Unit Number (MSTP) Line 1** reflects the Unit ID/MAC Address assignement for MS/TP Port 1.
C2	49970	Unsigned	RO	-	**Baud Rate Line 2** reflects the baud rate for MS/TP Port 2.
I2	50226	Unsigned	RO	-	**Unit Number (MSTP) Line 2** reflects the Unit ID/MAC Address assignment for MS/TP Port 2.
DE	50245	Unsigned	RW	0	**Default Enable** used to return the database of the MatrixBBC to default conditions, where a single Device object, and STATbus objects (if licensed) will be available. 0=Normal operation 197=set properties to their default values.
RS	53843	Unsigned	RW	0	**Reset the MatrixBBC Control Engine?** used to reset the control engine of the MatrixBBC. Setting **RS** to 1 resets the control engine. 0=No 1=Reset Control Engine
MK	52555	Unsigned	RW	0	**Max Analog Input Objects** used to dynamically create a set number of corresponding object types within the database.
MM	52557	Unsigned	RW	0	**Max Analog Output Objects** used to dynamically create a set number of corresponding object types within the database.
M1	52529	Unsigned	RW	0	**Max Analog Value Objects** used to dynamically create a set number of corresponding object types within the database.
ML	52556	Unsigned	RW	0	**Max Binary Input Objects** used to dynamically create a set number of corresponding object types within the database.
MN	52558	Unsigned	RW	0	**Max Binary Output Objects** used to dynamically create a set number of corresponding object types within the database.
M2	52530	Unsigned	RW	0	**Max Binary Value Objects** used to dynamically create a set number of corresponding object types within the database.
MC	52547	Unsigned	RW	0	**Max Calendar Objects** used to dynamically create a set number of corresponding object types within the database.
MD	52548	Unsigned	RW	0	**Max Schedule Objects** used to dynamically create a set number of corresponding object types within the database.
MQ	52561	Unsigned	RW	0	**Max Season Objects** used to dynamically create a set number of corresponding object types within the database.
M9	52537	Unsigned	RW	0	**Max Input Select Objects** used to dynamically create a set number of corresponding object types within the database.
MA	52545	Unsigned	RW	0	**Max Remap Objects** used to dynamically create a set number of corresponding object types within the database.

Property	Identifier #	Data Type	Access	Default Value	Description
MB	52546	Unsigned	RW	0	**Max Netmap Objects** used to dynamically create a set number of corresponding object types within the database.
MI	52553	Unsigned	RW	0	**Max Program Objects** used to dynamically create a set number of corresponding object types within the database.
M6	52534	Unsigned	RW	0	**Max Logic Objects** used to dynamically create a set number of corresponding object types within the database.
M7	52535	Unsigned	RW	0	**Max Math Objects** used to dynamically create a set number of corresponding object types within the database.
M8	52536	Unsigned	RW	0	**Max MinMaxAvg Objects** used to dynamically create a set number of corresponding object types within the database.
MF	52550	Unsigned	RW	0	**Max Trend Objects** used to dynamically create a set number of corresponding object types within the database.
M3	52531	Unsigned	RW	0	**Max Motor Objects** used to dynamically create a set number of corresponding object types within the database.
M4	52532	Unsigned	RW	0	**Max Scale Objects** used to dynamically create a set number of corresponding object types within the database.
M5	52533	Unsigned	RW	0	**Max PWC Objects** used to dynamically create a set number of corresponding object types within the database.
ME	52549	Unsigned	RW	0	**Max Enthalpy Objects** used to dynamically create a set number of corresponding object types within the database.
MG	52551	Unsigned	RW	0	**Max Thermostatic Control Objects** used to dynamically create a set number of corresponding object types within the database.
MH	52552	Unsigned	RW	0	**Max PID Control Objects** used to dynamically create a set number of corresponding object types within the database.
MJ	52554	Unsigned	RW	0	**Max Staging Objects** used to dynamically create a set number of corresponding object types within the database.
MO	52559	Unsigned	RW	0	**Max Notification Class Objects** used to dynamically create a set number of corresponding object types within the database.
MR	52562	Unsigned	RW	0	**Max Comm Status Objects** used to dynamically create a set number of corresponding object types within the database.
MS	52563	Unsigned	RW	0	**Max Manufacturing Objects** used to dynamically create a set number of corresponding object types within the database.

Property	Identifier #	Data Type	Access	Default Value	Description
MT	52564	Unsigned	RW	0	**Max Broadcast Objects** used to dynamically create a set number of corresponding object types within the database.

A.2 ANALOG INPUTS (UIS)

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Analog Input (3), Instance *N* or Binary Input (3), Instance *N*	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Universal Input *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Analog Input (0)	indicates membership in a particular object type class.
present_value	85	Real	RW	0	indicates the current value, in engineering units, of the object.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
event_state	36	BACnet Event State	RO	0	provides a way to determine if this object has an active event state associated with it.
reliability	103	BACnet Reliability	RO	0	indicates whether the **present_value** is reliable as far as the device or operator can determine.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
units	117	BACnet Eng. Units	RW	95	indicates the measurement units of this object.
min_pres_value	69	Real	RW		indicates the lowest number that can be reliably used for the **present_value** property of this object.
max_pres_value	65	Real	RW		indicates the highest number that can be reliably used for the **present_value** property of this object.
resolution	106	Real	RO	-0.001525	indicates the smallest recognizable change in **present_value** in engineering units (read-only).
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_ class	17	Unsigned	RW	1	specifies the notification class to be used when handling and generating event notifications for this object.
high_limit	45	Real	RW	0.0	specifies a limit that the **present_value** must exceed before an event is generated.
low_limit	59	Real	RW	0.0	specifies a limit below which the **present_value** must fall before an event is generated.
deadband	25	Real	RW	0.0	specifies a range between the **high_limit** and **low_limit** properties within which the **present_value** must remain for a TO-NORMAL event to be generated
limit_enable	52	BACnet Limit Enable	RW	0	enables and disables reporting of High Limit and Low Limit off normal events and their return to normal.

Property	Identifier #	Data Type	Access	Default Value	Description
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
IF	51526	Real	RW	0	**Input Filter Delay or Weighted Gain** the de-bounce time (in seconds) during which the input must remain stable to avoid the signal being read as a digital bounce. In the case of a bounce, the object **reliability** is set to 1. For analog inputs, **IF** specifies a weighted gain used to smooth the values from a fluctuating input.
OF	53062	Real	RW		**Input Offset** specifies an offset amount to be added to the current value.
RH	53832	Real	RW	0	**Run Hours** indicates the number of hours **present_value**=1 for the input.
DF	50246	Unsigned	RW	0	**Display Format** specifies the way in which the stat will display the temperature. 0=##d 1=##.#d 2=##dF 3=##.#dF 4=None
LS	52307	BACnet ObjID	RW	-	**STAT Linked Schedule** specifies the corresponding BACnet Schedule object linked to the thermostat for scheduling occupancy and overrides.
LL	52300	BACnet ObjID	RW	-	**STAT Linked Loop** specifies the corresponding control loop object linked to the thermostat for setpoint adjustments.
ED	50500	Unsigned	RW	0	**STAT Override Minutes** specifies the amount of time, in minutes, a schedule will be overridden when commanded.
TM	54349	Unsigned	RW	0	**STAT Show The Time** specifies if the current time should be displayed on the STAT LCD screen.
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.

Property	Identifier #	Data Type	Access	Default Value	Description
ST	54100	Unsigned	RW	0	**Sensor Type** specifies the type of sensor connected to the input. 0=Digital 2=MN..MX 0 to 5V 3=MN..MX 4 to 20mA 6=MN..MX 0 to 10V 7=Thermistor -30.0 to 230.0 8=MN..MX 0 to 20mA 9=SMARTStat Temperature 10=SMARTStat Humidity 11=Curve 1 12=Curve 2 13=Curve 3 14=Curve 4 15=Curve 5 16=Curve 6 17=Curve 7 18=Curve 8
GI	51017	Unsigned	RW	0	**GID of I/O Device** indicates the global identification number of the STATbus device associated with the universal input. If the input does not have a STATbus device mapped to it, **GI** will be 0.
I#	51491	Unsigned	RW	0	**Input Index of I/O Device** indicates the UI number that the Analog Input object will focus upon. For example, if you monitor UI2 on an IOX1-1, set **I#** = 2

A.3 BINARY INPUTS (UIS) AND (DIS)

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Binary Input (3), Instance *n*	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Universal Input *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Binary Input (3)	indicates membership in a particular object type class.
present_value	85	Enum	RW	0	indicates the current value, in engineering units, of the object.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
event_state	36	BACnet Event State	RO	0	provides a way to determine if this object has an active event state associated with it.
reliability	103	BACnet Reliability	RO	0	indicates whether the **present_value** is reliable as far as the device or operator can determine.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
polarity	84	BACnet Polarity	RW	0	indicates the relationship between the physical state of the output and the logical state represented by the **present_value** property. If the **polarity** property is NORMAL, then the ACTIVE state of the **present_value** property is also the ACTIVE or ON state of the physical output as long as **out_of_service** is FALSE. If the Polarity property is REVERSE, then the ACTIVE state of the **present_value** property is the INACTIVE or OFF state of the physical output as long as **out_of_service** is FALSE.
inactive-text	46	CharStr	RW	Off	specifies the text for an OWS to use when the present-value = Inactive.
active-text	4	CharStr	RW	On	specifies the text for an OWS to use when present-value = Active.
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_ class	17	Unsigned	RW	0	specifies the notification class to be used when handling and generating event notifications for this object.
alarm_value	6	BACnet BinaryPV	-	-	specifies the value that the Present_Value property must have before a TO-OFFNORMAL event is generated.
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.

Property	Identifier #	Data Type	Access	Default Value	Description
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
IV	51542	Unsigned	RO	0	**Inverted Present Value** indicates if the current present-value is inverted or reversed polarity.
RH	53832	Real	RW	0	**Run Hours** indicates the number of hours **present_value**=1 for the input.
IF	51526	Real	RW	0	**Input Filter Delay or Weighted Gain** the de-bounce time (in seconds) during which the input must remain stable to avoid the signal being read as a digital bounce. In the case of a bounce, the object **reliability** is set to 1. For analog inputs, **IF** specifies a weighted gain used to smooth the values from a fluctuating input.
MD	52548	Unsigned	RW	0	**Pulse Counter Mode** Defines the method used by the input to count pulses. 0=Rising Edges 1=Falling Edges 2=Both 3=Disable 255=Unsupported Feature
VR	54866	Unsigned	RW	0	**IVR Setting (For Pulse Counting Mode)** specifies the user adjusted hardware jumper position for the input.
PT	53332	Real	RW	0	**Pulse Threshold (For Pulse Counting Mode)** specifies the voltage threshold that must be sensed by the MatrixBBC in order to detect a pulse.
NP	52816	Unsigned	RW	0	**Number of Pulses Accumulated** defines the total amount of pulses counted by MatrixBBC.
SF	54086	Real	RW	0	**Pulse Multiplier** specifies a multiplier to apply against the number of pulses accumulated.
SV	54102	Real	RO	0	**Scaled Pulse Count** is the result of **NP** x **SF**
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.

Property	Identifier #	Data Type	Access	Default Value	Description
ST	54100	Unsigned	RW	0	**Sensor Type** specifies the type of sensor connected to the input. 0=Digital 2=MN..MX 0 to 5V 3=MN..MX 4 to 20mA 6=MN..MX 0 to 10V 7=Thermistor -30.0 to 230.0 8=MN..MX 0 to 20mA 9=SMARTStat Temperature 10=SMARTStat Humidity 11=Curve 1 12=Curve 2 13=Curve 3 14=Curve 4 15=Curve 5 16=Curve 6 17=Curve 7 18=Curve 8
GI	51017	Unsigned	RW	0	**GID of I/O Device** indicates the global identification number of the STATbus device associated with the universal input. If the input does not have a STATbus device mapped to it, **GI** will be 0.
I#	51491	Unsigned	RW	0	**Input Index of I/O Device** indicates the UI number that the Analog Input object will focus upon. For example, if you monitor UI2 on an IOX1-1, set **I#** = 2
ST	54100	Unsigned	RW	0	**Sensor Type** specifies the type of sensor connected to the input. 0=Digital 2=MN..MX 0 to 5V 3=MN..MX 4 to 20mA 4=Curve 1 5=Curve 2 6=MN..MX 0 to 10V 7=Thermistor -30.0 to 230.0 8=MN..MX 0 to 20mA 9=SMARTStat Temperature 10=SMARTStat Humidity

A.4 PIECEWISE CURVES

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (305), Instance 1-8	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Piecewise Curve *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (305)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
X1	55345	Real	RW	0	**Point 1's value in raw counts** specifies the x coordinate of point 1.
X2	55346	Real	RW	0	**Point 2's value in raw counts** specifies the x coordinate of point 2.
X3	55347	Real	RW	0	**Point 3's value in raw counts** specifies the x coordinate of point 3.
X4	55348	Real	RW	0	**Point 4's value in raw counts** specifies the x coordinate of point 4.
X5	55349	Real	RW	0	**Point 5's value in raw counts** specifies the x coordinate of point 5.
X6	55350	Real	RW	0	**Point 6's value in raw counts** specifies the x coordinate of point 6.
X7	55351	Real	RW	0	**Point 7's value in raw counts** specifies the x coordinate of point 7.
X8	55352	Real	RW	0	**Point 8's value in raw counts** specifies the x coordinate of point 8.
X9	55353	Real	RW	0	**Point 9's value in raw counts** specifies the x coordinate of point 9.
XA	55361	Real	RW	0	**Point 10's value in raw counts** specifies the x coordinate of point 10.
XB	55362	Real	RW	0	**Point 11's value in raw counts** specifies the x coordinate of point 11.
Y1	55601	Real	RW	0	**Point 1's value in engineering units** specifies the y coordinate of point 1.
Y2	55602	Real	RW	0	**Point 2's value in engineering units** specifies the y coordinate of point 2.
Y3	55603	Real	RW	0	**Point 3's value in engineering units** specifies the y coordinate of point 3.
Y4	55604	Real	RW	0	**Point 4's value in engineering units** specifies the y coordinate of point 4.
Y5	55605	Real	RW	0	**Point 5's value in engineering units** specifies the y coordinate of point 5.
Y6	55606	Real	RW	0	**Point 6's value in engineering units** specifies the y coordinate of point 6.

Property	Identifier #	Data Type	Access	Default Value	Description
Y7	556017	Real	RW	0	**Point 7's value in engineering units** specifies the y coordinate of point 7.
Y8	55608	Real	RW	0	**Point 8's value in engineering units** specifies the y coordinate of point 8.
Y9	55609	Real	RW	0	**Point 9's value in engineering units** specifies the y coordinate of point 9.
YA	55617	Real	RW	0	**Point 10's value in engineering units** specifies the y coordinate of point 10.
YB	55618	Real	RW	0	**Point 11's value in engineering units** specifies the y coordinate of point 11.

A.5 ANALOG OUTPUTS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Analog Output (1), Instance 1-72	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Analog Output *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Analog Output (1)	indicates membership in a particular object type class.
present_value	85	Real	RW	0.0	indicates the current value, in engineering units, of the object.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
event_state	36	BACnet Event State	RO	0	provides a way to determine if this object has an active event state associated with it.
reliability	103	BACnet Reliability	RO	0	indicates whether the **present_value** is reliable as far as the device or operator can determine.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
units	117	BACnet Eng. Units	RW	95	indicates the measurement units of this object.
min_pres_value	69	Real	RW	0.0	indicates the lowest number that can be reliably used for the **present_value** property of this object.
max_pres_value	65	Real	RW	100.0	indicates the highest number that can be reliably used for the **present_value** property of this object.
resolution	106	Real	RO	0.024414	indicates the smallest recognizable change in **present_value** in engineering units (read-only).
priority_array	87	BACnet Array	RO	NULL	contains prioritized commands that are in effect for this object.
relinquish_ default	104	Real	RW	0.0	the default value to be used for the **present_value** property when all command priority values in the **priority_array** property have a NULL value.
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_ class	17	Unsigned	RW	1	specifies the notification class to be used when handling and generating event notifications for this object.
high_limit	45	Real	RW	0.0	specifies a limit that the **present_value** must exceed before an event is generated.
low_limit	59	Real	RW	0.0	specifies a limit below which the **present_value** must fall before an event is generated.
deadband	25	Real	RW	0.0	specifies a range between the **high_limit** and **low_limit** properties within which the **present_value** must remain for a TO-NORMAL event to be generated

Property	Identifier #	Data Type	Access	Default Value	Description
limit_enable	52	BACnet Limit Enable	RW	0	enables and disables reporting of High Limit and Low Limit off normal events and their return to normal.
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
O1	53041	BACnet ObjID	RW	-	**AutoStuff Input Object** specifies the object identifier target for pulling values in.
P1	53297	Unsigned	RW	-	**AutoStuff Input Property** specifies the property for the object identifier target.
Q1	53553	Unsigned	RW	-	**AutoStuff Mode/Priority** specifies the priority-level or mode for pulling data in.
FB	50754	CharStr	RO		**Feedback Text** provides diagnostic feedback information regarding the status of how the output is being controlled.
LS	52307	Real	RW	0	**Minimum Scaled Voltage** specifies the percentage of the total output for **present_value**=**min_pres_value**.
HS	51283	Real	RW	100	**Maximum Scaled Voltage** specifies the percentage of the total output for **present_value**=**max_pres_value**.
OV	53078	Real	RO		**Actual Output Voltage** provides the actual voltage outputted by the I/O Processor.
OC	53059	Real	RO		**Actual Output Current** provides the actual current outputted by the I/O Processor.
UT	54612	Real	RW	-	**Update Threshold** specifies the time, in seconds, in which the physical output voltage or current is updated.
RH	53832	Unsigned	RW	-	**Run Hours** specifies, in hours, how long the Analog Output has sent constant voltage or current.
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.

Property	Identifier #	Data Type	Access	Default Value	Description
GI	51017	Unsigned	RW	0	**GID of I/O Device** indicates the global identification number of the STATbus device associated with the analog output. If the output does not have a STATbus device mapped to it, **GI** will be 0.
O#	53027	Unsigned	RW	0	**Output Index of I/O Device** indicates the AO number that the Analog Output object will focus upon. For example, if you monitor AO2 on an IOX1-1, set **O#** = 2
MN	52307	Real	RW	0	**Minimum Scaled Voltage** specifies the percentage of the total output for **present_value**=**min_pres_value**.
MX	51283	Real	RW	100	**Maximum Scaled Voltage** specifies the percentage of the total output for **present_value**=**max_pres_value**.
OU	53077	Boolean	RO	-	**Actual Output Value** specifies the actual output state of the output. This may differ from the current value because of delays and other effects.
UT	54612	Real	RW	0	**Update Threshold** specifies a threshold value by which **present_value** must change before the output is updated.

A.6 BINARY OUTPUTS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Binary Output (4), Instance 1-72	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Digital Output *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	Enum	RO	Binary Output (4)	indicates membership in a particular object type class.
present_value	85	Enum	RW	0	indicates the current value, in engineering units, of the object.
status_flags	111	Bit Str	RO	0	four flags that indicate the general health of the program.
event_state	36	Enum	RO	0	provides a way to determine if this object has an active event state associated with it.
reliability	103	Enum	RO	0	indicates whether the **present_value** is reliable as far as the device or operator can determine.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
polarity	84	Enum	RW	0	indicates the relationship between the physical state of the output and the logical state represented by the **present_value** property. If the **polarity** property is NORMAL, then the ACTIVE state of the **present_value** property is also the ACTIVE or ON state of the physical output as long as **out_of_service** is FALSE. If the Polarity property is REVERSE, then the ACTIVE state of the **present_value** property is the INACTIVE or OFF state of the physical output as long as **out_of_service** is FALSE.
inactive-text	46	CharStr	RW	Off	specifies the text for an OWS to use when the present-value = Inactive.
active-text	4	CharStr	RW	On	specifies the text for an OWS to use when present-value = Active.
minimum_off_time	66	Unsigned	RW	0	specifies the minimum number of seconds that the **present_value** shall remain in the INACTIVE state after a write to the **present_value** property causes that property to assume the INACTIVE state.
minimum_on_time	67	Unsigned	RW	0	indicates the minimum number of seconds that the **present_value** shall remain in the ACTIVE state after a write to the **present_value** property causes that property to assume the ACTIVE state.
priority_array	87	BACnet Array	RO	NULL	contains prioritized commands that are in effect for this object.
relinquish_default	104	Real	RW	7	the default value to be used for the **present_value** property when all command priority values in the **priority_array** property have a NULL value.
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_class	17	Unsigned	RW	0	specifies the notification class to be used when handling and generating event notifications for this object.
feedback_value	40	BACnet BinaryPV	-	Inactive	specifies the value that the Present_Value property must have before a TO-OFFNORMAL event is generated.

Property	Identifier #	Data Type	Access	Default Value	Description
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
O1	53041	BACnet ObjID	RW	-	**AutoStuff Input Object** specifies the object identifier target for pulling values in.
P1	53297	Unsigned	RW	-	**AutoStuff Input Property** specifies the property for the object identifier target.
Q1	53553	Unsigned	RW	-	**AutoStuff Mode/Priority** specifies the priority-level or mode for pulling data in.
OU	53077	Unsigned	RO	-	**Actual Output State** specifies the actual output state of the output. This may differ from the current value because of delays and other effects.
PW	53335	Real	RW	0	**Pulse Width when Output is On** specifies the “on” time (**present_value**=1) in seconds (0.0 to 25.5) that the output should remain on after a transition from the off to on state. 0=Disabled 0.1-25.5=pulse “on” duration in seconds
RH	53832	Real	RW	0	**Run Hours** indicates the number of hours **present_value**=1 for the input.
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.
AF	49478	CharStr	RO		**AutoStuff Feedback Text** provides diagnostic feedback information regarding the status of how the output is being controlled.
GI	51017	Unsigned	RW	0	**GID of I/O Device** indicates the global identification number of the STATbus device associated with the analog output. If the output does not have a STATbus device mapped to it, **GI** will be 0.
O#	53027	Unsigned	RW	0	**Output Index of I/O Device** indicates the BO number that the Binary Output object will focus upon. For example, if you monitor BO on an IOX1-1, set **O#** = 2

A.7 STATBUS SUMMARY

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (204), Instance *0*	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	STATbus *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (204)	indicates membership in a particular object type class.
PU	53333	Unsigned	RW	0	**User PIN** Defines the PIN password for accessing the user menu (setpoint adjust, and override functions) from a STAT/RHT product.
PI	53321	Unsigned	RW	3300	**Install PIN** Defines the PIN password for accessing the install menu from a STAT/ RHT product.
PS	53331	Unsigned	RW	1100	**Service PIN** Defines the PIN password for accessing the service menu from a STAT/RHT product.
LT	52308	Unsigned	RW	5	**STAT Inactivity Logout Timer** Defines the amount of time, in minutes, that
S1	54065	BACnet Array	RO	-	**STATBus 1 Devices** Provides a read-only display of all STAT and IOX modules connected to STATbus Port 1.
CR	50002	Unsigned	RW	0	**Configure Remote I/O** Used to configure the MatrixBBC to perform Expandable I/O mapping configuration. 0 = Normal 1 = MatrixBBC to Bus 2 = Edit I/O GIDs

A.8 STATBUS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (450), Instance *1*	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	STATbus 1	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (450)	indicates membership in a particular object type class.
G1	50993	Unsigned	RW	0	**GID Device 1** indicates the global identification number of device 1.
G2	50994	Unsigned	RW	0	**GID Device 2** indicates the global identification number of device 2.
G3	50995	Unsigned	RW	0	**GID Device 3** indicates the global identification number of device 3.
G4	50996	Unsigned	RW	0	**GID Device 4** indicates the global identification number of device 4.
G5	50997	Unsigned	RW	0	**GID Device 5** indicates the global identification number of device 5.
G6	50998	Unsigned	RW	0	**GID Device 6** indicates the global identification number of device 6.
G7	50999	Unsigned	RW	0	**GID Device 7** indicates the global identification number of device 7.
G8	51000	Unsigned	RW	0	**GID Device 8** indicates the global identification number of device 8.
G9	51001	Unsigned	RW	0	**GID Device 9** indicates the global identification number of device 9.
GA	51009	Unsigned	RW	0	**GID Device 10** indicates the global identification number of device 10.
GB	510010	Unsigned	RW	0	**GID Device 11** indicates the global identification number of device 11.
GC	51011	Unsigned	RW	0	**GID Device 12** indicates the global identification number of device 12.
GD	51012	Unsigned	RW	0	**GID Device 13** indicates the global identification number of device 13.
SM	54093	BitStr	RO	0	**Status Map** 1=unconfigured 2=duplicate

A.9 PROGRAMS 1-64

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Program (16), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Program *N* (unloaded)	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	- Program (16)	indicates membership in a particular object type class.
program_state	92	BACnet Program State	RO	0	indicates the current logical state of the process executing the application program this object represents.
program_change	90	BACnet Program Request	RW	0	used to request changes to the operating state of the process this object represents.
reason_for_halt	100	BACnet Program Error	RO	0	indicates the reason why the program was halted.
description_of_halt	29	CharStr	RO	-	describes the reason why a program has been halted.
program_location	91	CharStr	RO	Sec0:0x0000	indicate the current location within the program code, for example, a line number or program label or section name.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
reliability	103	BACnet Reliability	RO	0	indicates whether the program is running/waiting (no fault detected) or is unreliable (process-error)
out_of_service	81	Boolean	RO	0	indicates whether or not the process this object represents is not in service.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
$1	42033	Boolean	RW	0	**Enable Single-Step Mode?** specifies whether the single-step, line by line debugging mode is enabled. 0=No 1=Yes
$D	42052	Unsigned	RW	0	**Delay Time Remaining** specifies the number of seconds remaining when an **SWAIT** or **MWAIT** statement is encountered in the SPL program.
$E	42053	Unsigned	RW	0	**Error Code** indicates the SPL error code that is returned when the program aborts.
$F	42054	Unsigned	RO	-	**Device Instance Not Found** indicates a device instance that could not be communicated with when attempting to interact with objects from remote devices. This property displays the last device id that an error occured with.

Property	Identifier #	Data Type	Access	Default Value	Description
$W	42071	Unsigned	RW	0	**Trappable Error Action** specifies how the SPL program should handle trappable errors. 0=Abort on Error 1=Wait on Error
$N	46062	Unsigned	RW	0	**Number of Program Attributes** indicates the number of initialized properties defined in the program using the PROP statement.
%A	42305	Unsigned	RW	0	**Register A Value** indicates the value of program register A.
%B	42306	Unsigned	RW	0	**Register B Value** indicates the value of program register B.
%C	42307	Unsigned	RW	0	**Register C Value** indicates the value of program register C.
%D	42308	Unsigned	RW	0	**Register D Value** indicates the value of program register A.
%E	42309	Unsigned	RW	0	**Register E Value** indicates the value of program register E.
%F	42310	Unsigned	RW	0	**Register F Value** indicates the value of program register F.
%G	42311	Unsigned	RW	0	**Register G Value** indicates the value of program register G.
%H	42312	Unsigned	RW	0	**Register H Value** indicates the value of program register H.
%I	42313	Unsigned	RW	0	**Register I Value** indicates the value of program register I.
%J	42314	Unsigned	RW	0	**Register J Value** indicates the value of program register J.
%K	42315	Unsigned	RW	0	**Register K Value** indicates the value of program register K.
%L	42316	Unsigned	RW	0	**Register L Value** indicates the value of program register L.
%M	42317	Unsigned	RW	0	**Register M Value** indicates the value of program register M.
%N	42318	Unsigned	RW	0	**Register N Value** indicates the value of program register N.
%O	42319	Unsigned	RW	0	**Register O Value** indicates the value of program register O.
%P	42320	Unsigned	RW	0	**Register P Value** indicates the value of program register P.

A.10 FILE0

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	File (10), Instance 0	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	FILE0	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	File (10)	indicates membership in a particular object type class.
description	28	CharStr	RO	Flash Upgrade	provides a description of the object. This object is intended for flash updates.
file_type	43	CharStr	RO	Flash Upgrade	identifies the intended use of this file.
file_size	42	Unsigned	RO	327680	indicates the size of the file data.
modification_ date	71	BACnet Date Time	RO	NULL	indicates the last time this object was modified.
archive	13	Boolean	RW	1	indicates whether the File object has been saved for historical or backup purposes.
read_only	99	Boolean	RO	0	indicates whether or not the file data may be changed through the use of a BACnet Atomic Write File service.
file_access_ method	41	BACnet File Access Method	RO	1	indicates the type(s) of file access supported for this object.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.

A.11 PLB1-64

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	File (10), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	RAM*N*, PLB*N*, or LOGO*N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	File (10)	indicates membership in a particular object type class.
description	28	CharStr	RO	Flash Upgrade	provides a description of the object. This object is intended for flash updates. The **description** is the same as the **file_type**.
file_type	43	CharStr	RO	RAM*N*	identifies the intended use of this file. The possible file types are: Empty Region *n* System File *n* Trend File *n* Table File *n* Program Logic Block *n* Program Reference Block *n* Program Control Block *n* Display List *n* Custom Logo *n*
file_size	42	Unsigned	RW	-	indicates the size of the file data. Writing a value of zero to this property will clear out any program or other file currently loaded.
modification_ date	71	BACnet Date Time	RO	NULL	indicates the last time this object was modified.
archive	13	Boolean	RW	0	indicates whether the File object has been saved for historical or backup purposes.
read_only	99	Boolean	RO	0	indicates whether or not the file data may be changed through the use of a BACnet Atomic Write File service.
file_access_ method	41	BACnet File Access Method	RO	1	indicates the type(s) of file access supported for this object.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.

A.12 ANALOG PID

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (500), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Analog PID *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (500)	indicates membership in a particular object type class.
present_value	85	Real	RW	0	indicates the current calculated analog output value determined by the control loop. For direct control of AO's, this value is retrieved by the AO's AutoStuff feature.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
CE	49989	Boolean	RW	0	**Enable Control Loop?** enables/disables PID control. 0=No 1=Yes
CS	50003	Real	RO	0	**Calculated Control Setpoint** specifies the effective (calculated) setpoint accounting for setup/ setback, manual setpoint adjustments, etc.
DB	50242	Real	RW	0	**Desired Control Deadband** specifies the deadband that is used to control cycling around the setpoint. If the current value of the input object is between **SP**-(**DB**/2) and **SP**+(**DB**/2), the measured variable is considered to be at its setpoint.
IO	51535	BACnet ObjID	RW		**Input Object** specifies the object identifier that will be used as the measured variable for control calculations.
IP	51536	Unsigned	RW		**Input Property** specifies the property of the object identifier that will be used as the measure variable for control calculations.
IV	51542	Real	RO		**Input's Present Value** reflects the current value of the referenced input object-property specified using **IO** and **IP.**
OO	53071	BACnet ObjID	RW		**Interlock Override Object** specifies the object identifier that will be used to detect a boolean value for interlocking.
OP	53072	Unsigned	RW		**Interlock Override Property** specifies the property of the object identifier that will be used to detect a boolean value for interlocking.
OV	53078	Boolean	RO		**Interlock Override's Present State** reflects the current value of the referenced interlock object-property used for interlocking.
OH	53064	Real	RW	100	**Output High Limit** defines the maximum output for the PID control loop.
OL	53068	Real	RW	0	**Output Low Limit** defines the minimum output for the PID control loop.

Property	Identifier #	Data Type	Access	Default Value	Description
PB	53314	Real	RW	0	**Proportional Control Band** specifies a range, centered around the loop setpoint **SP**, where the output signal is proportional. If the value of the selected input object is outside the proportional band, the proportional component of the PID calculation is clamped at **OL** or **OH** as appropriate.
PO	53327	Real	RW	0	**Percent Output Value** displays the calculated output of the PID control loop. **PO** ranges from **OL** to **OH**.
RT	53844	Real	RW	0	**Derivative Rate** specifies a a percentage of the amount of derivative error that is contributed each second to the PID output of the control loop (0.0 to 25.5%). 0.0=Disable 0.1 to 25.5=Derivative rate in%/second
RP	53840	Unsigned	RW	0	**Reset Period** specifies a time, in seconds (0 to 65,535) over which the output of the control loop should be adjusted (reset) using integral action. 0=Disabled 1 to 65,535=Integral reset period, in seconds
RC	53827	BACnet ObjID	RW	0	**<Reset Feature> Reset Object** specifies the object to be used as the reset variable for the PID control loop.
RA	53825	Unsigned	RW	00	**<Reset Feature> Reset Property** specifies the property associated with the object specified in **RC** to be used as the reset variable for the PID control loop.
MR	52562	Real	RW	0	**<Reset Feature> Maximum Amount to Reset Setpoint** the maximum amount to reset the control loop setpoint.
RL	53836	Real	RW	0	**<Reset Feature> Limit for Maximum Reset** specifies the reset limit of the control loop. When **RV** reaches a value of **RL**, the control loop setpoint will be reset by the maximum amount **RV**.
RS	53843	Real	RW	0	**<Reset Feature> Setpoint at which Reset Action Begins** specifies the setpoint of the control loop at which reset action begins.
SG	54087	Unsigned	RW	0	**Control Action** specifies whether the controller's output should be increased or decreased when error is positive. 0=Normal (increase for positive error) 1=Reverse (decrease for positive error)
SM	54093	BitStr	RW	0	**Schedules to Follow** enables scheduled alarm controlling for the associated PID control loop by selecting one or more of the available schedule control objects. 0=Schedule disabled 1=Schedule enabled **SM** is a bitmap where: bit 0=Schedule 1 up to... bit 31=Schedule 32
US	54611	Real	RW	0.0	**Unoccupied Setpoint <Sched=0>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Unoccupied Mode.

Property	Identifier #	Data Type	Access	Default Value	Description
WS	55123	Real	RW	0.0	**Warmup Setpoint <Sched=1>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Warmup Mode.
OS	53075	Real	RW	0.0	**Occupied Setpoint <Sched=2>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Occupied Mode.
NS	52819	Real	RW	0.0	**Night Setback Setpoint <Sched=3>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Night Setback Mode.
SR	54098	Unsigned	RW	100	**Soft Start Ramp** specifies the maximum percentage change per minute for the associated output under he following conditions: when the controller is initially powered up or reset; upon transitions from unoccupied to occupied mode, upon cancellation of an interlock failure or fire condition, or when a control loop is initially enabled.
DL	50252	Real	RO	0.0	**Demand Load** indicates the heating/cooling demand in terms of the temperature separation from setpoints
MO	52559	Real	RW	-	**STAT Maximum Override Offset (used as +/-)** specifies the maximum adjust amount that the control setpoint can be adjusted from a linked STAT.
CO	49999	Real	RO	-	**STAT Current Override Offset** indicates the current setpoint offset commanded by the user.
OR	53074	Unsigned	RO	-	**STAT Override Time Remaining** indicates the current remaining time for override mode.
FB	50754	CharStr	RO	-	**Feedback Text** indicates diagnostic feedback of the control loop for troubleshooting.

A.13 PULSE PAIR PID

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (501), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Pulse-Pair PID *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (501)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
CE	49989	Unsigned	RW	0	**Enable Control Loop?** enables/disables floating point control for the associated control loop. 0=No 1=Yes
CF	49990	Unsigned	RW	0	**Communication Failure Enable?** specifies what action to take in the event that a communication failure is detected. 0=No 1=Yes
CP	50000	Real	RO	0	**Current Position** indicates the current position of the motor.
CR	50002	Unsigned	RW	0	**Creep Enable** specifies how the controller handles automatic calibrations at minimum and maximum positions. 0=Drive motor constantly if **DP**=0% or **DP**=100% 1=Creep motor output by 1% per minute if **DP**=0% or **DP**=100%
SG	54087	Unsigned	RW	0	**Control Action** specifies whether the controller's output should be increased or decreased when the control signal is positive. 0=Normal (increase for positive error) 1=Reverse (decrease for positive error)
CS	50003	Real	RO	0	**Calculated Control Setpoint** indicates the calculated control setpoint. This value accounts for any reset or setup/setback action on the loop setpoint.
DB	50242	Real	RW	0	**Desired Control Deadband** specifies the deadband that is used to control cycling around the setpoint. If the current value of the input object is between **SP**-(**DB**/2) and **SP**+(**DB**/2), the measured variable is considered to be at its setpoint
DP	50256	Real	RW	0	**Desired Position** specifies the desired output position (0-100%) of the associated motor.
IO	51535	BACnet ObjID	RW	-	**Input Object** specifies the object identifier that will be used as the measured variable for control calculations.
IP	51536	Unsigned	RW	-	**Input Property** specifies the property of the object identifier that will be used as the measure variable for control calculations.

Property	Identifier #	Data Type	Access	Default Value	Description
IV	51542	Real	RO	-	**Input's Present Value** reflects the current value of the referenced input object-property specified using **IO** and **IP.**
PB	53314	Real	RW	0	**Proportional Control Band** specifies a range, centered around the loop setpoint **SP**, where the output signal is proportional.
RP	53840	Unsigned	RW	0	**Reset Period** specifies a time, in seconds (0 to 65,535) over which the output of the control loop should be adjusted (reset). 0=Diabled 1 to 65,535=Reset period, in seconds
RC	53827	BACnet ObjID	RW	0	**<Reset Feature> Reset Object** specifies the object to be used as the reset variable for the PID control loop.
RA	53825	Unsigned	RW	00	**<Reset Feature> Reset Property** specifies the property associated with the object specified in **RC** to be used as the reset variable for the PID control loop.
MR	52562	Real	RW	0	**<Reset Feature> Maximum Amount to Reset Setpoint** the maximum amount to reset the control loop setpoint.
RL	53836	Real	RW	0	**<Reset Feature> Limit for Maximum Reset** specifies the reset limit of the control loop. When **RV** reaches a value of **RL**, the control loop setpoint will be reset by the maximum amount **RV**.
RS	53843	Real	RW	0	**<Reset Feature> Setpoint at which Reset Action Begins** specifies the setpoint of the control loop at which reset action begins.
SM	54093	BitStr	RW	0	**Schedules to Follow** enables scheduled alarm controlling for the associated PID control loop by selecting one or more of the available schedule control objects. 0=Schedule disabled 1=Schedule enabled **SM** is a bitmap with: bit 0=Schedule 1 up to... bit 31=Schedule 32
US	54611	Real	RW	0.0	**Unoccupied Setpoint <Sched=0>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Unoccupied Mode.
WS	55123	Real	RW	0.0	**Warmup Setpoint <Sched=1>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Warmup Mode.
OS	53075	Real	RW	0.0	**Occupied Setpoint <Sched=2>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Occupied Mode.
NS	52819	Real	RW	0.0	**Night Setback Setpoint <Sched=3>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Night Setback Mode.
TT	54356		RW	0	**Motor Travel Time** specifies the time, in seconds (0-3000), that it takes the motor to move from its fully closed to its fully open positions.

Property	Identifier #	Data Type	Access	Default Value	Description
RI	53833	Unsigned	RW	0	**Motor Recalibrate Interval** specifies a time interval in hours (0-255) the defines how often the associated floating point control loop is recalibrated. 0=Calibration disabled **RI** > 0 =Recalibrate every **RI** hours
DL	50252	Real	RO	-	**Demand Load** indicates the heating/cooling demand in terms of the temperature separation from setpoints
MO	52559	Real	RW	-	**STAT Maximum Override Offset (used as +/-)** specifies the maximum adjust amount that the control setpoint can be adjusted from a linked STAT.
CO	49999	Real	RO	-	**STAT Current Override Offset** indicates the current setpoint offset commanded by the user.
OR	53074	Unsigned	RO	-	**STAT Override Time Remaining** indicates the current remaining time for override mode.
O1	53041	Boolean	RO	-	**Output #1 (Load)** indicates open action as determined by the control loop. For direct control of BO's, this value is retrieved by the BO's AutoStuff feature.
O2	53042	Boolean	RO	-	**Output #2 (Unload)** indicates closed action as determined by the control loop. For direct control of BO's, this value is retrieved by the BO's AutoStuff feature.
FB	50754	CharStr	RO	-	**Feedback Text** indicates diagnostic feedback of the control loop for troubleshooting.

A.14 THERMOSTATIC CONTROL

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (502), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Thermostatic Control *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (502)	indicates membership in a particular object type class.
present_value	85	Real	RW	0	indicates the current output value for control. For direct control of BO's, this value is retrieved by the BO's AutoStuff feature.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
CE	49989	Boolean	RW	0	**Enable Control Loop?** enables/disables thermostatic control for the associated control loop. 0=Disabled 1=Enabled
MD	52548	Unsigned	RW	0	**Mode** Specifies the control sign for the loop, based on season or schedule. 0=Heating in Winter <Else Off> 1=Heating in Winter <Else Seasonal Setback> 2=Cooling in Summer <Else Off> 3=Cooling in Summer <Else Seasonal Setback>
SS	54099	Boolean	RW	0	**Season** Defines the current season, used specifically for seasonal setback applications.
US	54611	Real	RW	0.0	**Unoccupied Setpoint <Sched=0>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Unoccupied Mode.
WS	55123	Real	RW	0.0	**Warmup Setpoint <Sched=1>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Warmup Mode.
OS	53075	Real	RW	0.0	**Occupied Setpoint <Sched=2>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Occupied Mode.
NS	52819	Real	RW	0.0	**Night Setback Setpoint <Sched=3>** specifies the setpoint under which the control loop shall operate when a selected, properly configured schedule from **SM** is in Night Setback Mode.
DB	50242	Real	RW	0	**Desired Control DeadBand** specifies a control hysteresis that is used to keep **present_value** from toggling when the value is on the border between two states.
SO	54095	Real	RW	0.0	**Seasonal Setup/Setback <for Modes 1 and 3>** specifies the amount of setback to apply to all four setpoints when seasonal setback is in effect.

Property	Identifier #	Data Type	Access	Default Value	Description
SM	54093	BitStr	RW	0	**Schedules to Follow** enables scheduled alarm controlling for the associated PID control loop by selecting one or more of the available schedule control objects. 0=Schedule disabled 1=Schedule enabled **SM** is a bitmap with: bit 0=Schedule 1 up to... bit 31=Schedule 32
IO	51535	BACnet ObjID	RW	-	**Input Object** specifies the object identifier that will be used as the measured variable for control calculations.
IP	51536	Unsigned	RW	-	**Input Property** specifies the property of the object identifier that will be used as the measure variable for control calculations.
IV	51542	Real	RO	-	**Input's Present Value** reflects the current value of the referenced input object-property specified using **IO** and **IP.**
CS	50003	Real	RO	0	**Calculated Control Setpoint** specifies the calculated (actual) control setpoint that is used by the thermostatic control loop. **CS** accounts for the effects of seasonal setup/setback (**SO**) during scheduled seasonal modes.
MO	52559	Real	RW	-	**STAT Maximum Override Offset (used as +/-)** specifies the maximum adjust amount that the control setpoint can be adjusted from a linked STAT.
CO	49999	Real	RO	-	**STAT Current Override Offset** indicates the current setpoint offset commanded by the user.
OR	53074	Unsigned	RO	-	**STAT Override Time Remaining** indicates the current remaining time for override mode.
DL	50252	Real	RO	-	**Demand Load** indicates the heating/cooling demand in terms of the measured variable separation from setpoints
SF	54086	CharStr	RO	-	**Schedule Feedback** indicates diagnostic feedback relative to how schedule control is affecting the control loop.
MF		CharStr	RO	-	**Mode Feedback** indicates diagnostic feedback relative to how the current mode is affecting the control loop.
TF		CharStr	RO	-	**Temperature Feedback** indicates diagnostic feedback relative to when action will occur based on programmed setpoints, schedule mode, etc.

A.15 SCHEDULES

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Schedule (17), Instance 1-32	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Schedule *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Schedule (17)	indicates membership in a particular object type class.
present_value	85	-any-	RO	-	specifies the current value written to the list-of-object-property-references by the object.
effective_period	32	BACnet Range	RW	-	defines the effective range for the schedule.
weekly-schedule	123	BACnetArray	RW	-	defines the weekly operating schedule (Monday thru Sunday). Each day of the week may contain up to 20 time,value pairs that provide programmatic schedule control.
exception_schedule	38	BACnetArray	RW	-	defines the exception-schedule, typically overriding the weekly-schedule. Exceptions are based off a Calendar reference, date reference, or even an object reference (boolean). Up to 5 exception events may be programmed into the exception schedule, each with a maximum of 20 time, value pairs.
schedule-default	174	-any-	RW	-	defines the data-type and default value that the schedule shall assume if no weekly or exception data is programmed for a specific day. This value is also used for situations where a **weekly-schedule** entry commands the schedule to resume default operations.
list_of_object_property_references	54	BACnet List	RW	-	defines the list of object properties which the schedule will write values to based on time, value pairs entered into the **weekly-schedule** or **exception-schedule**.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
reliability	103	BACnet Reliability	RO	0	indicates whether the **present_value** is reliable as far as the device or operator can determine.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service. When **out-of-service** = true, the **present-value** property is freely writable. Any value written to **present-value** during times when **out-of-service** is true will be written down to the object-properties referenced in **list-of-object-property-references**.
priority-for-writing	88	Unsigned	RW	11	defines the priority level used for writing values to commandable objects such as AOs and BOs.
DT	50260	-any-	RW	2	**Schedule Default Data Type** defines the datatype used for programmatic scheduling. Valid data types include: 1=Boolean 2=Unsigned 3=Integer 4=Real 9=Enumerated 10=Date 11=Time 12=BACnet ObjectID
FB	50754	Boolean	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the schedule.

Property	Identifier #	Data Type	Access	Default Value	Description
CU	50005	Boolean	RO	-	**Currently Unoccupied Flag <0>** specifies if the Schedule is currently in Unoccupied mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Unsigned data-type and use the classic four-mode AAM Schedules.
CW	50007	Boolean	RO	-	**Currently Warmup Flag <1>** specifies if the Schedule is currently in Warmup mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Unsigned data-type and use the classic four-mode AAM Schedules.
CO	49999	Boolean	RO	-	**Currently Occupied Flag <2>** specifies if the Schedule is currently in Occupied mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Unsigned data-type and use the classic four-mode AAM Schedules.
CS	50003	Boolean	RO	-	**Currently Night Setback Flag <3>** specifies if the Schedule is currently in Night Setback mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Unsigned data-type and use the classic four-mode AAM Schedules.
ON	53070	Boolean	RO	-	**Currently On Flag <1 or 2>** specifies if the Schedule is currently in Off or in Unoccupied mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Enumerated data-type where 0=Unoccupied and 1=Occupied
OF	53062	Boolean	RO	-	**Currently Off Flag <0 or 3>** specifies if the Schedule is currently in On or in Occupied mode. In order a schedule to be unoccupied, the Schedule must be configured to use an Enumerated data-type where 0=Unoccupied and 1=Occupied
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.

A.16 CALENDARS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Calendar (6), Instance 1-32	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Schedule *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Calendar (6)	indicates membership in a particular object type class.
present_value	85	Boolean	RO	-	specifies the known local-date of the Device matches with an entry in the **datelist**.
datelist	23	List	RW	-	specifies the list of date ranges, dates, or week-n-day entries that make up special events or other programmatic data.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
OS	53075	Boolean	RW	False (0)	**Auto Delete Stale Entries** clears expired valid M/D/Y entries from the datelist once they have expired (when the local-date is after the entry in the list). This service does not remove wild-card based entries and only applies to M/D/Y entries that are considered one-shot entries.

A.17 NOTIFICATION CLASS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Notification Class (15), Instances (0 - 9)	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	NOTIFICATIONCLASS *n*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Notification Class (15)	indicates membership in a particular object type class.
notification_class	17	Unsigned	RO	1	specifies the notification class to be used when handling and generating event notifications for this object.
priority	86	BACnet Array	RW	2	specifies the priority to be used for event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively.
ack_required	1	BACnet Event Trans. Bits	RW	1, 0, 1	three separate flags that indicate whether acknowledgment shall be required in notifications generated for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL event transitions, respectively.
recipient_list	102	BACnet List	RW	-	a list of one or more recipient destinations to which notifications shall be sent when event-initiating objects using this class detect the occurrence of an event.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
A1	49457	Boolean	RO	False (0)	defines is recipient 1 of the recipient-list is active and ready to route alarms.
A2	49458	Boolean	RO	False (0)	defines is recipient 2 of the recipient-list is active and ready to route alarms.
A3	49459	Boolean	RO	False (0)	defines is recipient 3 of the recipient-list is active and ready to route alarms.
A4	49460	Boolean	RO	False (0)	defines is recipient 4 of the recipient-list is active and ready to route alarms.
A5	49461	Boolean	RO	False (0)	defines is recipient 5 of the recipient-list is active and ready to route alarms.

A.18 MATH

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (301), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Math *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (301)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
present-value	85	Real	RO	-	provides the result of the Math operation.
OP	53072	Unsigned	RW	0	**Operation** specifies the operation to be performed on the selected objects. 0=Disabled 1=Addition 2=Subtraction 3=Multiplication 4=Division 5=Minimum 6=Maximum 7=Average
I1	45361	BACnet ObjID	RW	0	**Input Object 1** specifies the first input object
A1	49457	Unsigned	RW	0	**Input property 1** specifies the property associated with the first input object.
I2	45362	BACnet ObjID	RW	0	**Input object 2** specifies the second input object.
A2	49458	Unsigned	RW	0	**Input property 2** specifies the property associated with the second input object.
GT	51028	Boolean	RO	-	**Input 1 Is > Input 2?** indicates if the value of Input 1 is greater than the value of Input 2. A true indication will be returned if true, else false.
GE	51013	Boolean	RO	-	**Input 1 is >= Input 2?** indicates if the value of Input 1 is greater or equal to the value of Input 2. A true indication will be returned if true, else false.
LT	52308	Boolean	RO	-	**Input 1 is < Input 2?** indicates if the value of Input 1 is less than the value of Input 2. A true indication will be returned if true, else false.
LE	52293	Boolean	RO	-	**Input 1 is <= Input 2?** indicates if the value of Input 1 is less than or equal to the value of Input 2. A true indication will be returned if true, else false.
ET	50516	Boolean	RO	-	**Input 1 is = Input 2?** indicates if the value of Input 1 is equal to the value of Input 2. A true indication will be returned if true, else false.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.

A.19 LOGIC

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (303), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Logic *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (303)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
present-value	85	Real	RO	-	provides the result of the operation.
I1	51505	BACnet ObjID	RW	0	**Input Object 1** specifies the first input object.
A1	49457	Unsigned	RW	0	**Input Property 1** specifies the property associated with the first input object.
I2	51506	BACnet ObjID	RW	0	**Input Object 2** specifies the second input object.
A2	49458	Unsigned	RW	0	**Input Property 2** specifies the property associated with the second input object.
I3	51507	BACnet ObjID	RW	0	**Input Object 3** specifies the third input object.
A3	49459	Unsigned	RW	0	**Input Property 3** specifies the property associated with the third input object.
I4	51508	BACnet ObjID	RW	0	**Input Object 4** specifies the fourth input object.
A4	49460	Unsigned	RW	0	**Input Property 4** specifies the property associated with the fourth input object.
I5	51509	BACnet ObjID	RW	0	**Input Object 5** specifies the fifth input object.
A5	49461	Unsigned	RW	0	**Input Property 5** specifies the property associated with the fifth input object.
I6	51510	BACnet ObjID	RW	0	**Input Object 6** specifies the sixth input object.
A6	49462	Unsigned	RW	0	**Input Property 6** specifies the property associated with the sixth input object.
I7	51511	BACnet ObjID	RW	0	**Input Object 7** specifies the seventh input object.
A7	49463	Unsigned	RW	0	**Input Property 7** specifies the property associated with the seventh input object.
I8	51512	BACnet ObjID	RW	0	**Input Object 8** specifies the eighth input object.
A8	49464	Unsigned	RW	0	**Input Property 8** specifies the property associated with the eighth input object.

Property	Identifier #	Data Type	Access	Default Value	Description
AI	49481	BitStr	RO	-	**Active Inputs** indicates which logic inputs are being actively monitored.
IV	51542	BitStr	RO	-	**Input Current Values** indicates the current value of each monitored input.
RV	53846	BitStr	RW	0	**Inverted [Reversed] Inputs** defines the logic polarity of each input.
OP	53072	Unsigned	RW	0	**Operation** specifies the logic operation to be performed on the selected objects. 0=Diabled 1=OR 2=AND 3=NOT <Input 1> 4=XOR

A.20 MIN/MAX/AVG

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (301), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Min/Max/Avg *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (301)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
HV	51286	Real	RW	0	**High Value** displays the highest value of the inputs selected.
LV	52310	Real	RW	0	**Low Value** displays the lowest value of the inputs selected.
AV	49494	Real	RW	0	**Average Value** displays the arithmetic mean of the inputs selected.
I1	51505	BACnet ObjID	RW	-	**Input Object 1** specifies the object from which the first property to be used for min/max/avg calculations can be selected.
A1	49457	Unsigned	RW	-	**Input Property 1** specifies the property in the object selected in **I1** to be used as the first input min/max/avg calculations.
I2	51506	BACnet ObjID	RW	-	**Input Object 2** specifies the object from which the second property to be used for min/max/avg calculations can be selected.
A2	49458	Unsigned	RW	-	**Input Property 2** specifies the property in the object selected in **I2** to be used as the second input min/max/avg calculations.
I3	51507	BACnet ObjID	RW	-	**Input Object 3** specifies the object from which the third property to be used for min/max/avg calculations can be selected.
A3	49459	Unsigned	RW	-	**Input Property 3** specifies the property in the object selected in **I3** to be used as the third input min/max/avg calculations.
I4	51508	BACnet ObjID	RW	-	**Input Object 4** specifies the object from which the fourth property to be used for min/max/avg calculations can be selected.
A4	49460	Unsigned	RW	-	**Input Property 4** specifies the property in the object selected in **I4** to be used as the fourth input min/max/avg calculations.

A.21 ENTHALPY

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (308), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Enthalpy *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (308)	indicates membership in a particular object type class.
present-value	85	Real	RO	-	provides the result of the operation.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
TO	54351	BACnet ObjID	RW	-	**Temperature Object** specifies the temperature object identifier.
TP	54352	Unsigned	RW	-	**Temperature Property** specifies the temperature property, corresponding to **TO**.
TV	54358	Real	RO	-	**Temperature Value** indicates the current measured value of the defined object property.
HO	51279	BACnet ObjID	RW	-	**Humidity Object** specifies the humidity object identifier.
HP	51280	Unsigned	RW	-	**Humidity Property** specifies the temperature property, corresponding to **HO**.
HV	51286	Real	RO	-	**Humidity Value** indicates the current measured value of the defined object property.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.
units	117	Enum	RO	24	**units** indicates the engineer unit that present-value reflects.

A.22 SCALING

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (306), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Scale *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (306)	indicates membership in a particular object type class.
present-value	85	Real	RO	-	specifies the calculated scaled value.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
IO	51535	BACnet ObjID	RW	-	**Input Object** specifies the object to be scaled.
IP	51536	Unsigned	RW	0	**Input Property** specifies the property associated with the object specified in **IC** to be scaled.
X1	55345	Real	RW	0	**Input range X1 value** specifies the minimum value of the input.
X2	55346	Real	RW	0	**Input range X2 value** specifies the maximum value of the input.
Y1	55601	Real	RW	0	**Output range Y1 value** specifies the minimum value of the scaled output.
Y2	556012	Real	RW	0	**Output range Y2 value** specifies the maximum value of the scaled output.

A.23 INPUT SELECT

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (300), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Input Select *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (300)	indicates membership in a particular object type class.
present-value	85	Real	RO	-	indicates the value of the property which has been selected.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
I1	45361	BACnet ObjID	RW	-	**Input Object 1** specifies the object from which the first input property will be chosen.
A1	49457	Unsigned	RW	0	**Input Property 1** specifies the property associated with the first input object.
I2	45362	BACnet ObjID	RW	-	**Input Object 2** specifies the object from which the second input property will be chosen.
A2	49458	Unsigned	RW	0	**Input Property 2** specifies the property associated with the second input object.
SC	540834	BACnet ObjID	RW	-	**Selection Object** specifies the object from which the property used for selection will be chosen.
SA	54081	Unsigned	RW	0	**Selection Property** specifies the property to be used as the selection criteria. If the specified property has a value of 0, **present-value** will take the value of the property specified in **I1** and **A1**. If the specified property has a value of 1, **present-value** will take the value of the property specified in **I2** and **A2**.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.

A.24 STAGING

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (309), Instance 1-16	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Staging *n*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (240)	indicates membership in a particular object type class.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
SM	54093	Unsigned	RW	-	**Staging Mode** defines the staging application to be used 0=Object Turned Off 1=Delay On/Delay Off 2=Threshold Based Staging
LM	52301	Unsigned	RW	-	**Lead/Lag/Leveling Mode** defines the staging function performed by the object 0=Normal <First On/Last Off> 1=Automatic Wear Leveling
NS	52819	Unsigned	RW	-	**Number of Stages <Max Loading>** defines the number of stage outputs to use 2=2 Stages 3=3 Stages 4=4 Stages 5=5 Stages 6=6 Stages 7=7 Stages 8=8 Stages
IO	51535	BACnet ObjID	RW	-	**Input Object** defines the input object used as the measured variable to perform staging.
IP	51536	Unsigned	RW	0	**Input Property** defines the object property corresponding to **IO**.
IV	51542	Real	RO		**Input Value** reflects the current value of the input object property.
II	51529	Boolean	RW	-	**Invert the Input?** specifies if the input sign should be reversed.
IS	51539	Boolean	RW	-	**Invert the Setpoints?** specifies if the setpoints should be inverted for heating or cooling.
OO	53071	BACnet ObjID	RW	-	I**nterlock Override Object** defines the input object used as the measured variable to perform interlock overrides.
OP	53072	Unsigned	RW	-	**Interlock Override Property** defines the object property corresponding to **OO**.
OM	53069	BitStr	RW	-	**Interlocking Staging Map** specifies the state of each stage output when interlocking is enabled.
OS	53075	CharStr	RO	-	**Interlock Status** Provides feedback regarding the operational status of interlocking.

Property	Identifier #	Data Type	Access	Default Value	Description
SU	54101	Real	RW	0.0	**Loading Setpoint** defines the setpoint at which stages are enabled.
SL	54092	Real	RW	0.0	**Unloading Setpoint** defines the setpoint at which stages are disabled.
P1	53297	Real	RW	0.0	**Stage 1 Setpoint** defines the setpoint at which stage action will occur.
P2	53298	Real	RW	0.0	**Stage 2 Setpoint** defines the setpoint at which stage action will occur.
P3	53299	Real	RW	0.0	**Stage 3 Setpoint** defines the setpoint at which stage action will occur.
P4	53300	Real	RW	0.0	**Stage 4 Setpoint** defines the setpoint at which stage action will occur.
P5	53301	Real	RW	0.0	**Stage 5 Setpoint** defines the setpoint at which stage action will occur.
P6	53302	Real	RW	0.0	**Stage 6 Setpoint** defines the setpoint at which stage action will occur.
P7	53303	Real	RW	0.0	**Stage 7 Setpoint** defines the setpoint at which stage action will occur.
P8	53304	Real	RW	0.0	**Stage 8 Setpoint** defines the setpoint at which stage action will occur.
LD	52292	Unsigned	RW	60	**Loading Interval <Seconds>** defines the amount of time, in seconds, that the Staging object will delay between enabling stages.
UD	54596	Unsigned	RW	60	**Unloading Interval <Seconds>** defines the amount of time, in seconds, that the Staging object will delay between disabling stages.
LR	52306	Unsigned	RO	-	**Seconds Until Next Loading Event Could Occur** indicates the amount of time remaining until the next stage is enabled.
UR	54610	Unsigned	RO	-	**Seconds Until Next Unloading Event Could Occur** indicates the amount of time remaining until the next stage is disabled.
PR	53330	Unsigned	RO	-	**Present Stages of Loading** indicates the number of stages being loaded by the Staging object.
S1	54065	Boolean	RO	False	**Stage 1 Status** Indicates the current status of this specific stage.
S2	54066	Boolean	RO	False	**Stage 2 Status** Indicates the current status of this specific stage.
S3	54067	Boolean	RO	False	**Stage 3 Status** Indicates the current status of this specific stage.
S4	54068	Boolean	RO	False	**Stage 4 Status** Indicates the current status of this specific stage.
S5	54069	Boolean	RO	False	**Stage 5 Status** Indicates the current status of this specific stage.
S6	54070	Boolean	RO	False	**Stage 6 Status** Indicates the current status of this specific stage.

Property	Identifier #	Data Type	Access	Default Value	Description
S7	54071	Boolean	RO	False	**Stage 7 Status** Indicates the current status of this specific stage.
S8	54072	Boolean	RO	False	**Stage 8 Status** Indicates the current status of this specific stage.
R1	53809	Real	RW	0.0	**Stage 1 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R2	53810	Real	RW	0.0	**Stage 2 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R3	53811	Real	RW	0.0	**Stage 3 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R4	53812	Real	RW	0.0	**Stage 4 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R5	53813	Real	RW	0.0	**Stage 5 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R6	53814	Real	RW	0.0	**Stage 6 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R7	53815	Real	RW	0.0	**Stage 7 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
R8	53186	Real	RW	0.0	**Stage 8 Run Hours** indicates the current amount of time, in hours, this specific stage has operated.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.

A.25 BROADCAST

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (143), Instance 0	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Broadcast Outside Air Temp	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (143)	indicates membership in a particular object type class.
present-value	85	Real	RO	-	indicates the value of the property which has been selected.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
BM	49741	Unsigned	RW	0	**Broadcast Mode** specifies whether the controller should send or receive broadcasts 0=Disabled 1=Send 2=Receive
BZ	49498	Boolean	RW	0	**Broadcast Zone/Global** specifies whether the controller's broadcasts are sent to the zone or broadcast globally. 0=Zone broadcast 1=Global broadcast
ZN	55886	Unsigned	RW	0	**Zone Number** specifies the zone number to broadcast to.
IO	51535	BACnet ObjID	RW	-	**Input Object** specifies the input object to be broadcast.
IP	51536	Unsigned	RW	0	**Input Property** specifies the property associated with the object specified in **IC** to be broadcast.
BT	49748	Unsigned	RW	5	**Broadcast Time Interval <1 to 20 Mins>** specifies the amount of time, in minutes, that the object will send the broadcast message.
ES	50515	Unsigned	RO	0	**Elapsed Seconds Since Broadcast** specifies the amount of time, in seconds, since the last broadcast message was sent successfully.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.

A.26 REMAP

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (304), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Remap Object *n*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (304)	indicates membership in a particular object type class.
present-value	85	varies	RO	-	indicates the value of the property which is mapped to the output.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
O1	53041	BACnet ObjID	RW	-	**Input Object** defines the input object.
P1	53297	Unsigned	RW	-	**Input Property** defines the property of **IO**.
O2	53042	BACnet ObjID	RW	-	**Output Object** defines the object to send the **present-value** to.
P2	53298	Unsigned	RW	0	**Output Property** defines the property of **O2**.
Q2	53554	Unsigned	RW	255	**Output Priority** defines the writing priority, if the value is being written to a commandable object.
TO	54351	BACnet ObjID	RW	-	**Trigger Object** defines the trigger object.
TP	54352	Unsigned	RW	0	**Trigger Property** defines the property of **TO**.
TB	54338	Unsigned	RW	0	**Trigger Biasing** defines the biasing mode for sending values.
RS	53843	CharStr	RO	-	**Remap Status** indicates the current status of remapping.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.
RM	53837	Unsigned	RW	0	**Remap Mode** defines the mode for remaps 0=Disabled 1=Continuous 2=When Triggered <Else NULL> 3=When Triggered

A.27 NETMAP

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (307), Instance 1-64	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Netmap Object *n*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Proprietary (307)	indicates membership in a particular object type class.
present-value	85	varies	RO	-	indicates the value of the property which is mapped to the output.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
O1	53041	BACnet ObjID	RW	-	**Input Object** defines the input object.
P1	53297	Unsigned	RW	-	**Input Property** defines the property of **IO**.
II	51529	Unsigned	RW	-	**Input Device Instance** defines the device instance corresponding to **O1** and **P1**.
O2	53042	BACnet ObjID	RW	-	**Output Object** defines the object to send the **present-value** to.
P2	53298	Unsigned	RW	-	**Output Property** defines the property of **O2**.
OI	53065	Unsigned	RW	-	**Output Device Instance** defines the device instance corresponding to **O2** and **P2**.
Q2	53554	Unsigned	RW	-	**Output Priority** defines the writing priority, if the value is being written to a commandable object.
TO	54351	BACnet ObjID	RW	-	**Trigger Object** defines the trigger object.
TP	54352	Unsigned	RW	-	**Trigger Property** defines the property of **TO**.
TB	54338	Unsigned	RW	-	**Trigger Biasing** defines the biasing mode for sending values.
TM	54349	Unsigned	RW	-	**Time Between Writes in Seconds** defines the amount of time, in seconds, to delay between writes.
ET	50516	Unsigned	RO	-	**Elapsed Time Since Last Write** defines the amount of time, in seconds, since the last write occurred.
RS	53843	CharStr	RO	-	**Remap Status** indicates the current status of remapping.
FB	50754	CharStr	RO	-	**Feedback Text** provides diagnostic details regarding the operation of the object.

Property	Identifier #	Data Type	Access	Default Value	Description
RM	53837	Unsigned	RW	RAM -	**Remap Mode** defines the mode for remaps 0=Disabled 1=Continuous 2=When Triggered <Else NULL> 3=When Triggered

A.28 ANALOG VALUE

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Analog Value (2), Instance 1-1000	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Analog Value *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	BACnet ObjType	RO	Analog Value (2)	indicates membership in a particular object type class.
present_value	85	Real	RW	0.0	indicates the current value, in engineering units, of the object.
status_flags	111	BACnet Status Flags	RO	0	four flags that indicate the general health of the program.
event_state	36	BACnet Event State	RO	0	provides a way to determine if this object has an active event state associated with it.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
units	117	BACnet Eng. Units	RW	95	indicates the measurement units of this object.
priority_array	87	BACnet Array	RO	NULL	contains prioritized commands that are in effect for this object.
relinquish_ default	104	Real	RW	0.0	the default value to be used for the **present_value** property when all command priority values in the **priority_array** property have a NULL value.
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_ class	17	Unsigned	RW	1	specifies the notification class to be used when handling and generating event notifications for this object.
high_limit	45	Real	RW	0.0	specifies a limit that the **present_value** must exceed before an event is generated.
low_limit	59	Real	RW	0.0	specifies a limit below which the **present_value** must fall before an event is generated.
deadband	25	Real	RW	0.0	specifies a range between the **high_limit** and **low_limit** properties within which the **present_value** must remain for a TO-NORMAL event to be generated
limit_enable	52	BACnet Limit Enable	RW	0	enables and disables reporting of High Limit and Low Limit off normal events and their return to normal.
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.

Property	Identifier #	Data Type	Access	Default Value	Description
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.

A.29 BINARY VALUE

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Binary Value (5), Instance 1-1000	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Binary Value *N*	represents a name for the object that is unique internetwork-wide.
object_type	79	Enum	RO	Binary Value (5)	indicates membership in a particular object type class.
present_value	85	Enum	RW	0	indicates the current value, in engineering units, of the object.
status_flags	111	Bit Str	RO	0	four flags that indicate the general "health" of the program.
event_state	36	Enum	RO	0	provides a way to determine if this object has an active event state associated with it.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
inactive-text	46	CharStr	RW	Off	specifies the text for an OWS to use when the present-value = Inactive.
active-text	4	CharStr	RW	On	specifies the text for an OWS to use when present-value = Active.
minimum_off_ time	66	Unsigned	RW	0	specifies the minimum number of seconds that the **present_value** shall remain in the INACTIVE state after a write to the **present_value** property causes that property to assume the INACTIVE state.
minimum_on_ time	67	Unsigned	RW	0	indicates the minimum number of seconds that the **present_value** shall remain in the ACTIVE state after a write to the **present_value** property causes that property to assume the ACTIVE state.
priority_array	87	BACnet Array	RO	NULL	contains prioritized commands that are in effect for this object.
relinquish_ default	104	Real	RW	7	the default value to be used for the **present_value** property when all command priority values in the **priority_array** property have a NULL value.
time_delay	113	Unsigned	RW	0	specifies the minimum period of time in seconds during which the **present_value** must be different from the **alarm_value** property before a TO-OFFNORMAL event is generated or must remain equal to the **alarm_value** property before a TO-NORMAL event is generated.
notification_ class	17	Unsigned	RW	0	specifies the notification class to be used when handling and generating event notifications for this object.
alarm-value	40	BACnet BinaryPV	-	Inactive	specifies the value that the Present_Value property must have before a TO-OFFNORMAL event is generated.
event_enable	35	BACnet Event Trans. Bits	RW	0	three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
acked_ transitions	0	BACnet Event Trans. Bits	RW	7	three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events.
notify_type	72	BACnet Notify Type	RW	0	specifies whether the notifications generated by the object should be Events or Alarms.

Property	Identifier #	Data Type	Access	Default Value	Description
event_time_stamps	130	BACnet ARRAY	RO	-	defines the time stamps for when alarms for each limit type have been sent.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
EA	50497	Boolean	RW	False	**Enable Alarming** specifies if alarming should be enabled for the object. When set to False, all alarming properties will be unavailable for selection.

A.30 COMM STATUS

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (400), Instance 0	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Comm Status	represents a name for the object that is unique internetwork-wide.
object_type	79	Enum	RO	Proprietary (400)	indicates membership in a particular object type class.
present_value	85	Enum	RW	0	indicates the current value, in engineering units, of the object.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
out_of_service	81	Boolean	RW	0	indicates whether or not the process this object represents is not in service.
FM	50765	Unsigned	RW	-	**Failure Mode Selection** defines the method used to determine comm failure. 0=Always mark as good 1=Fail if not passed a token 2=Fail if no data is read/written.
FD	50756	Unsigned	RW	10	**Failure Mode Delay Time in Seconds** defines the amount of time, in seconds, that sould elapse before a comm failure verified.
BD	49732	Unsigned	RW	20	**Failure Mode Boot Delay in Seconds** defines the amount of time, in seconds, that sould elapse after initial boot-time before a comm failure verified.

A.31 SEASON

Property	Identifier #	Data Type	Access	Default Value	Description
object_identifier	75	BACnet ObjID	RO	Proprietary (402), Instance 0	a numeric code that is used to identify the object.
object_name	77	CharStr	RW	Season	represents a name for the object that is unique internetwork-wide.
object_type	79	Enum	RO	Proprietary (402)	indicates membership in a particular object type class.
present_value	85	Enum	RW	0	indicates the current value, in engineering units, of the object.
profile-name	168	CharStr	RO	6-BBC-51-R1	defines the profile name used by AAM Tools to correspond program files to the MatrixBBC controller.
TC	54339	BitStr	RW	-	**T-STAT Objects Controlled By This Object** defines each T-STAT loop controlled by the present-value state of this object,where: bit 0=Thermostatic Control Loop 1 up to.... bit 31=Thermostatic Control Loop 32
TV	54358	BitStr	RO	-	**T-STAT Objects Value in Summer Mode** indicates if the corresponding T-STAT loop is in summer mode.
TS	54355	BitStr	RO	-	**T-STAT Objects Current SS State** reflects the current value of property SS on each T-STAT loop object.