DINIO-RS485

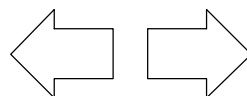## user's manual

ANIMATICS®

Defining the Future in Motion Control

Overview

The DINIO-RS485 is a DIN rail mounted I/O expansion board that communicates over RS-485 with the SmartMotor™ versions 3 and 4 product lines. Hereafter, the SmartMotor™ version 3 and 4 products will be simply be referred to as "SmartMotor™". It provides a convenient way to expand the SmartMotor's on board 7 TTL I/O to as many as 1632 optically-isolated, 24V I/O. Half of these are inputs and the other half are outputs.

The DINIO-RS485 has five primary functions:

· 24V, optically isolated inputs
· 24V, optically isolated sourcing outputs
· RS-485 interface
· Command processor
· Animatics peripheral expansion bus
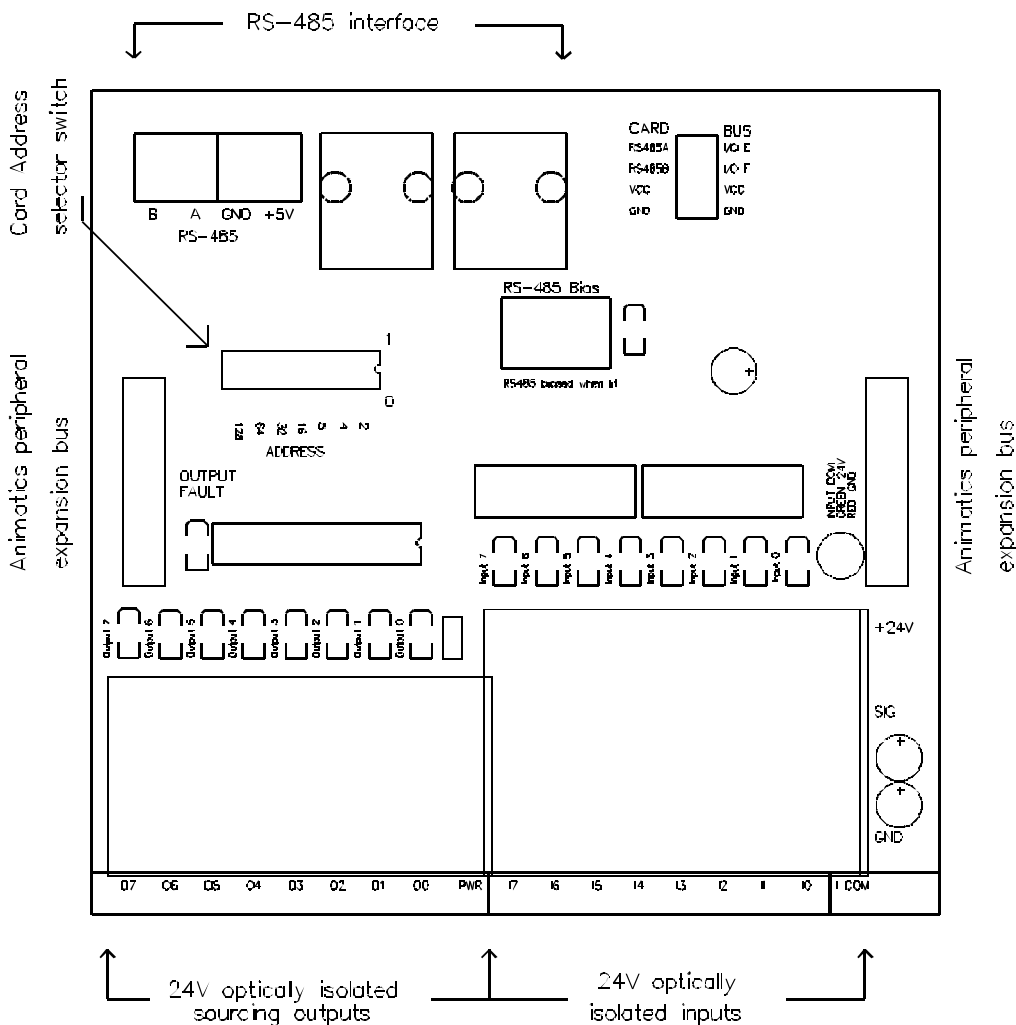
The ratings for the DIN-RS232 are given below. Absolute maximum ratings are levels beyond which damage may occur.

Absolute maximum I/O bus power:	31VDC
Absolute minimum I/O bus power:	-0.3VDC
Absolute maximum I/O voltage:	No higher than applied bus voltage
Absolute minimum I/O voltage:	-0.3VDC
Absolute maximum output load current:	200 mA, single output, 650 mA total
Absolute maximum RS-485 comm voltage:	5.5VDC
Absolute minimum RS-485 comm voltage:	-0.3VDC
Recommended I/O bus operating voltage:	12VDC to 30VDC
Nominal I/O input current:	5.7 mA (each input)

24V OPTICALLY ISOLATED INPUTS

There are eight 24V inputs that are optically isolated form the SmartMotor™. You may have noticed that, even though there are eight inputs, there are nine three stack terminal blocks on the input section of the unit. Eight are for the mentioned inputs, while the right-most one is used to select whether the eight inputs are sourcing or sinking.

All eight inputs share the same ground return reference and all eight are selected to be sourcing or sinking through a single wire jumper in the right most terminal labeled "I COM." If you do not select the input to be sourcing or sinking, the input will not function correctly. The DINIO-RS485 is normally shipped from the factory setup for a sourcing input.

All nine of the connections on the top row of the three stack terminals are tied to the +24V bus and all the terminals in the bottom row are all connected to 24V return. The middle row is connected as indicated on the silkscreen.

*If you do not select the input to be sourcing or sinking, the input will not function correctly. The DINIO-RS485 is normally shipped from the factory setup for a sourcing input.*
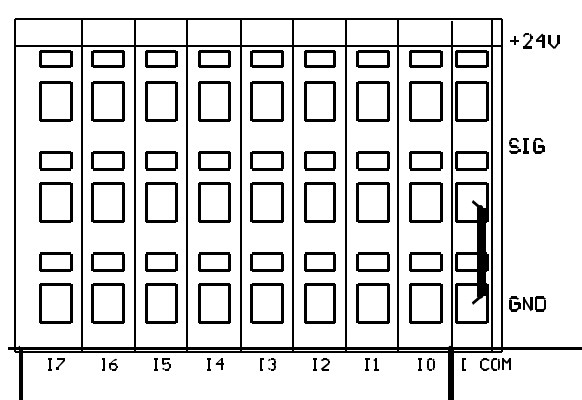
Sourcing/sinking input selection jumper wire as shown selects sourcing inputs

To set up the inputs to function with a sourcing device, connect a wire between the I COM signal (middle) terminal and the I COM ground (bottom) terminal. All of the inputs should be connected to the input signal (middle) terminals. If you are using two wire devices, like a proximity sensor, you may optionally connect the other device lead to the associated +24V terminal - it is there only for convenient hookup. That is not to say, of course, that it is optional to connect the other lead of the two wire device, but that you may find it convenient to use the connection on the DINIO-RS485.

Conversely, to set up the inputs to function with a sinking device, connect a wire between the I COM signal (middle) terminal and the I COM +24V (top) terminal. If you are using two wire devices, you may optionally connect the other device lead to the associated GND terminal. Again, this does not imply that it is optional to connect the other lead of the two wire device, but that you may find it convenient to use the connection on the DINIO-RS485.

A representative schematic of one of the inputs is given below:
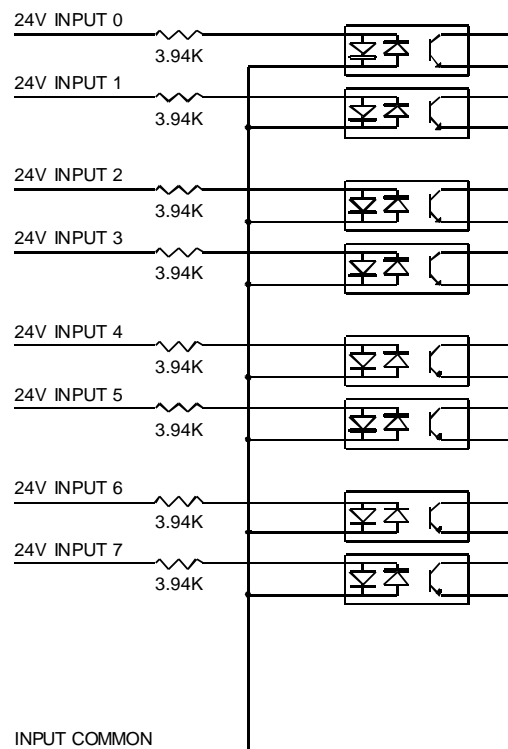
By default, the input +24V is connected to the output +24V. The power voltages of the input and output stages can be separated by removing the jumper adjacent to the PWR two stack terminal block. In any case, the user must provide +24V power to the input terminal block in order for the DINIO-RS485 unit to function.

the user must provide +24V power to the input terminal block in order for the DINIO-RS485 unit to function

24V OPTICALLY ISOLATED OUTPUTS

There are eight optically-isolated outputs on the DINIO-RS485. You may have noticed that, while there are eight outputs, there are nine two stack terminal blocks. Eight are for the mentioned outputs, while the right-most one is for an optional connection to +24V power.

Optional output power connection
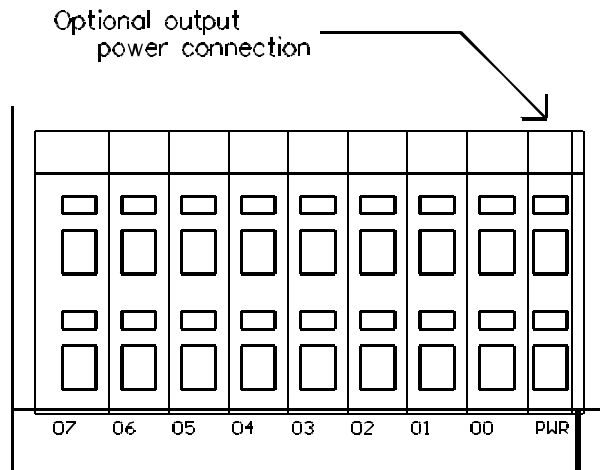
The two-stack output terminal labeled PWR allows you to power the inputs and outputs separately by removing the jumper adjacent to the PWR two stack terminal. That is, the jumper connects the +24V of the input and output sections of the DINIO-RS485. If you remove the jumper, you must power the output section of the unit at the two stack PWR terminal block - it will not operate without it.

The outputs are sourcing only and are equipped with anti-parallel diodes to easily interface to relay and solenoid coils. A schematic of the output is given below:
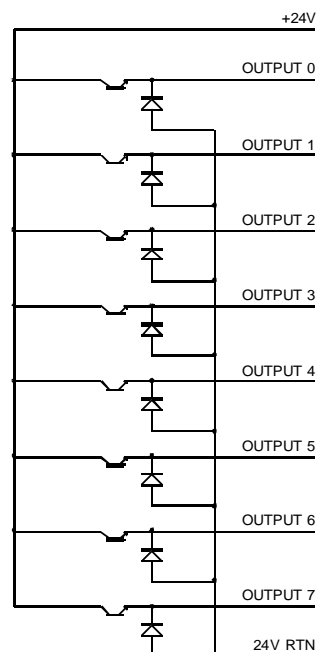
If you remove the jumper, you must power the output section of the unit at the two stack PWR terminal block - it will not operate without it.

+24V

OUTPUT 0

OUTPUT 1

OUTPUT 2

OUTPUT 3

OUTPUT 4

OUTPUT 5

OUTPUT 6

OUTPUT 7

24V RTN

All nine of the bottom row connections on the two stack terminal blocks are connected to +24V return. This is the same +24V return as on the input stage. The input and output stages are not isolated from each other, but they are isolated from the SmartMotor™ interface.

RS-485 INTERFACE

The RS-485 interface is the only means of the communicating with the SmartMotor™. Its baud rate defaults to 9600, but can communicate at 2400, 4800, 19200, 38400 and 76800 (the version 4 SmartMotor™ cannot communicate at a baud rate of 76800m however). The RS-485 interface is accessible through either of two RJ6 modular connectors. There are two connectors to allow for the convenient connection of multiple DINIO-RS485 units.

The RS-485 interface is set to be a slave by default. This means that the interface is always "listening" if nothing is being transmitted from a DINIO-RS485. In other words, the lines are floating inputs. You may notice a pushbutton switch and LED labeled RS-485 BIAS. If this button is pressed, the LED will either light or go dark. If the LED is lit green, the RS-485 is biased with on-board pull-up, pull-down and terminating resistors as shown in the following schematic:

+5V

RS485A

RS485B

510 Ohm

301 Ohm

510 Ohm
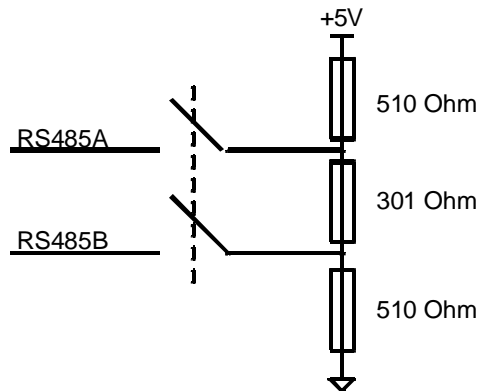
Due to the nature of RS-485, the biasing resistors, or their equivalent, must exist somewhere in a RS-485 communication bus. The terminator is required in particularly noisy environments or high baud rates. If these resistors are absent, intermittent or continuous communication errors may result. In many cases, RS-485 transceivers that are designed as masters, or hosts, already incorporate this biasing, but not the terminating resistor.

The RS-485 interface is a three pin hookup on the two RJ6 modular connectors. The pinout is shown in the following diagram:

No connect
RS-485 +
RS-485 -
No connect
GND
No connect
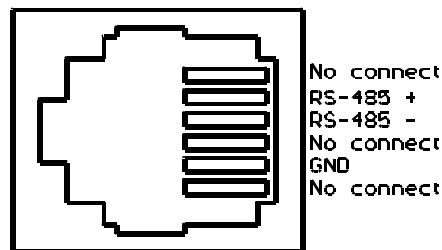
COMMAND PROCESSOR

The command processor functions as the "brain" of the DINIO-RS485. It processes commands received via the RS-485 interface to control communications, read inputs, set outputs and report its status.

Communications control is limited to setting the baud rate. The command to do this is BAUDn, where n is the baud rate. For example, BAUD19200 sets the baud rate to 19200 bps. The DINIO-RS485 serial communications is set up for 8 data bits, 1 stop bit and no parity. These parameters cannot be modified by the user.

Reading inputs is accomplished by requesting the DINIO-RS485 to report its input values via the the RN, RNA, RO, ROA and RIOA commands.

RNn reports the state of the nth individual input.

RNA   reports the state of all inputs for all DINIO-RS485 cards connected on the RS-485 bus.
ROn   reports the state the nth individual output.
ROA   reports the state of all outputs for all DINIO-RS485 cards connected to the RS-485 bus.
RIOA   reports both inputs and outputs states for all DINIO-RS485 cards connected to the RS-485 bus.

The format and method by which these values can be accessed is controlled by the IOF and RADELAY commands.

IOF   sets up the format of the return value of the above report commands
RADELAY   sets the delay between reported characters when using RNA, ROA and RIOA. This avoids serial data collisions.

The outputs can be turned on and off by the OR and OS commands.

ORn   resets (turns off) the nth output.
OSn   sets (turns on) the nth output.

The value of the I/O are stored in the local byte-wide variables ab[0 .. 203].

There are also a number of status and informational report commands. These are RSA, RSP and RSPA.

RSA   reports the value of the selector switches of all DINIO-RS485 connected to the RS-485 bus.
RSP   reports the sample period and firmware version of the card.
RSPA   reports the sampler period of all DIOIO-RS485 cards connected on the RS-485 bus.

Finally, there is the reset command ZA.

ZA   reboots all DINIO-RS485 cards on the RS-485 network to their power up condition.
Zsn   clears the syntax error flag on the DINIO-RS485 containing the nth input/output.
ZsA   clears the syntax error flag on all DINIO-RS485 cards on the RS-485 bus.

Before getting into detail about the commands, we first have to go over how to identify individual IO over a serial communications network. In the case of the SmartMotor™, each unit has a unique, individual address. Commands or requests for information are preceded by an address, followed by the desired command. In the case of the DINIO-RS485, all units have the same address, decimal 244. Thus, when using the DINIO-RS495 with SmartMotors, do not assign a SmartMotor™ the address 244.

Cards are not differentiated from one another by their address, but a sub-address set by the 7 bit DIP switch on the card.   The weird thing about the DIP switch is that there is no switch posi-

tion for the 0th bit. The labeling under the DIP switch starts with 2. The 0th bit is not available because it is always 0. When you set the 2's switch position, but the others are all at 0, the value that the DINIO-RS485 "sees" is 2, even though you set only one bit. Similarly, if the DIP switch is set to the value of the DIP switch sub-address is 128 + 32 + 8 = 168.

This DIP switch sub-address is important if you want to read the byte value of the eight inputs or eight outputs on the DINIO-RS485 card. The byte value of any set of eight inputs on any particular card is stored in the variable ab[DIP switch setting], while the byte value of the eight outputs is stored in the variable ab[DIP switch setting + 1]. So, in our example above, where the DIP switch is set to 168, the byte value of the inputs is stored in ab[168] and the byte value of the outputs is in ab[169]. The index i of the ab[i] variable holding the inputs is always even, while that of the output is always odd.

The inputs of the DINIO-RS485 follow the convention of the SmartMotor™ and are active low. That is, a zero in a particular position means that current is flowing through the input. For example, if the value of ab[168] is 57, that means that inputs 1 and 2 are active. That is, there is current flowing through them.

The outputs of the DINIO-RS485 are sourcing, so they are tracked as active high. That is, a 1 in a particular position means that it is sourcing current. For example, if the value of ab[169] is 57, that means that outputs 0, 5, 6, and 7 are sourcing current.

If you don't want to deal with byte values, you can also access the state of individual inputs and outputs by there ID number. Take a look at the card and note that the inputs and outputs are numbered 0 through 7. The ID of an individual input is determined by the equation

   input ID = (4 x DIP switch sub-adddress) + input number.

An output is specified by

   output ID = (4 x DIP switch sub-address) + output number.

On the card with sub-address 168, the 4th input (counting from 0) is input number 676. The 2nd output is 674.

Suppose that you have four DINIO-RS485 cards. You set the address of the first one to 0 (all switches set to 0), the second to 2, the third to 4 and the fourth to 32. The inputs of these four cards are:

Input	*#0*	*#1*	*#2*	*#3*	*#4*	*#5*	*#6*	*#7*
Card #0	0	1	2	3	4	5	6	7
Card #2	8	9	10	11	12	13	14	15
Card #4	16	17	18	19	20	21	22	23
Card #32	128	129	130	131	132	133	134	135

The outputs of these four cards are:

Output	*#0*	*#1*	*#2*	*#3*	*#4*	*#5*	*#6*	*#7*
Card #0	0	1	2	3	4	5	6	7
Card #2	8	9	10	11	12	13	14	15
Card #4	16	17	18	19	20	21	22	23
Card #32	128	129	130	131	132	133	134	135

Note that the ID numbering scheme for inputs and outputs are the same. The 2nd input has the same ID number as the 2nd output on any given card. With that firmly in our grasp, let us proceed to the detailed command descriptions.

BAUD*n*

BAUD sets the data rate at which the DINIO-RS485 communicates over its RS-485 channel. This is the only communications parameter the user can modify. The choices in baud rate are 2400, 4800, 9600, 19200, 38400 and 76800. The default baud rate is 9600.

Example syntax: `BAUD38400      'sets the baud rate to 38.4kbps.`

IOF=*value*

IOF determines how the DINIO-RS485 will respond to commands to report the states of the inputs and outputs. The following table defines the behavior for various values of IOF:

The index i of the ab[i] variable holding the inputs is always even, while that of the output is always odd

IOF Behavior

1   If an RN command (report state of input) is received, the DINIO-RS485 will return the byte value of all eight inputs of the specified card in the format the global address (ASCII 128), "ab[DIP switch setting]="value. The user should be aware that this directly overwrites the value of the variable ab[DIP switch setting] within the SmartMotor™.

2   If an RN command (report state of input) is received, the DINIO-RS485 will globally return the bit value of the specified input. The format of the return will be the global address (ASCII 128), "z="value. The user should be aware that this directly overwrites the value of the variable z within the SmartMotor™.

4   If an RO command (report state of output) is received, the DINIO-RS485 will globally return the byte value of all eight outputs of the specified input. The format of the return will be the global address (ASCII 128), "ab[DIP switch setting]="value. The user should be aware that this directly overwrites the value of the variable ab[DIP switch setting] within the SmartMotor™.

8   If an RO command (report state of output) is received, the DINIO-RS485 will globally return the bit value of the specified input. The format of the return will be the global address (ASCII 128), "z="value. The user should be aware that this directly overwrites the value of the variable z within the SmartMotor™.

16 Upon receipt of the OR or OS commands, the DINIO-RS485 will globally report the byte value of its outputs in addition to setting or resetting its outputs. The report will be in the form of the global SmartMotor™ address (ASCII 128), followed by "ab[DIP switch setting+1]"=value. The user should be aware that this directly overwrites the value of the variable ab[DIP switch setting] within the SmartMotor™.

32 Upon receipt of the OR or OS commands, the DINIO-RS485 will globally report the bit value of the specified output in addition to setting or resetting its outputs. The report will be in the form of the global SmartMotor™ address (ASCII 128), followed by z=value. The user should be aware that this directly overwrites the value of the variable z within the SmartMotor™.

The format of the return will be the global address (ASCII 128), "z="value. The user should be aware that this directly overwrites the value of the variable z within the SmartMotor™.

The user should be aware that this directly overwrites the value of the variable ab[DIP switch setting] within the SmartMotor™.

Example syntax: `IOF=3      'sets the value of IOF to 3`

These values can be combined as desired. For example, if IOF=3, the DINIO-RS485 will report both its byte and specified bit value. Crazily enough, the default setting of the DINIO-RS485 is 63, where all of the six above conditions are true.

OR*n*

The OR command causes the specified output to be reset, or to stop sourcing current. The output is specified by the ID number n, which ranges between 0 and 815.

Example syntax: `OR7        'Turn off output 7 on card #0`
                        `OR132      'Turn off output 4 on card #32`

Not content with just turning off the output, the DINIO-RS485 can tell everyone about it. If the fifth bit of IOF is set (that is, IOF=16), the DINIO-RS485 will globally report the byte value of its outputs. The exact syntax of the transmission will be

```
<ASCII 128>ab[DIP switch setting + 1]=byte value<carriage return>.
```

Note that, since there are eight IO ID numbers per card, eight different ORn commands refer to the same card. For example, OR0 through OR7 all do the same thing.

Similarly, if the sixth bit of IOF is set (that is, IOF=32), the DINIO-RS485 will globally emit the bit value of the output that just got reset, assigned to the variable z. This is of limited utility, as it will always cause the DINIO-RS485 to emit

```
<ASCII 128>z=0<carriage return>.
```

If both the fifth and sixth bits are set, the DINIO-RS485 will do both, one after another. If neither bit is set, the DINIO-RS485 will not transmit anything when the OR command is issued.

OSn

The OS command sets the specified output, having it source current. The output is specified by the ID number n, which ranges between 0 and 815.

```
Example syntax:      OS7          'Turn on output 7 on card #0
                     OS132        'Turn on output 4 on card #32
```

The DINIO-RS485 can tell about setting its output, as well as doing it. If the fifth bit of IOF is set (that is, IOF=16), the DINIO-RS485 will globally report the byte value of its outputs. The exact syntax of the transmission will be

```
<ASCII 128>ab[DIP switch setting + 1]=byte value<carriage return>.
```

Note that, since there are eight IO ID numbers per card, eight different OSn commands refer to the same card. For example, OS0 through OS7 all do the same thing.

Similarly, if the sixth bit of IOF is set (that is, IOF=32), the DINIO-RS485 will globally emit the bit value of the output that just got reset, assigned to the variable z. This is of limited utility, as it will always cause the DINIO-RS485 to emit

```
<ASCII 128>z=0<carriage return>.
```

If both the fifth and sixth bits are set, the DINIO-RS485 will do both, one after another. If neither bit is set, the DINIO-RS485 will not transmit anything when the OS command is issued.

RADELAY=value

There are several commands that cause all DINIO-RS485 cards to report their input or output values. When such a command is received by the DINIO-RS485, the card awaits its turn to transmit its character set onto the bus, where its turn is determined by its DIP switch sub-address. The RADELAY command introduces an additional delay on top of this, where the unit of the specified value is measured in approximately 0.25 millisecond increments. Note that this command is not specific to any sub-address. It is applied to all DINIO-RS485 cards on the communications bus.

```
Example syntax:      RADELAY=4    'Wait 1 extra millisecond between
                                   character sets
                     RADELAY=36   'Wait 9 extra millisecond between
                                   character sets
```

The valid range of values for RADELAY is 0 to 255, or 0 to 63.75 milliseconds, with a default value of 7, or 1.75 milliseconds.

Note that, since there are eight IO ID numbers per card, eight different OSn commands refer to the same card. For example, OS0 through OS7 all do the same thing.

Rab[number]

Rab[number] causes the DINIO-RS485 whose DIP switch setting corresponds to number to reports the byte value of the all eight inputs or outputs on the specified DINIO-RS485 card. If number is even, Rab[number] will cause the specified DINIO-RS485 card to respond with a global address, ASCII 128, followed by "ab[number]=" and the byte value of the eight inputs. Similarly, if number is odd, Rab[number] will cause the card to respond with a global address, ASCII 128, followed by "ab[number]=" and the byte value of the eight outputs.

```
Example syntax:        Rab[2]      'Card #2 globally reports the byte
                                   value of its inputs
                       Rab[3]      'Card #2 globally reports the byte
                                   value of its outputs
```

The valid range of values for number is 0 to 203.

Rab[card,bit]

Rab[card,bit] causes the DINIO-RS485 whose DIP switch sub-address corresponds to card to globally transmit the bit value of the input or output specified by bit. If card is even, the value of the input corresponding to bit is globally transmitted by first issuing a global address, ASCII 128, followed by "z=" and the bit value of the specified input. If card is odd, the output value corresponding to bit is globally transmitted by first issuing the global address, ASCII 128, followed by "z=" and the bit value of the specified output.

```
Example syntax:        Rab[2,2]    'Card #2 globally reports the bit value
                                   of its 2nd input
                       Rab[3,2]    'Card #2 globally reports the bit value
                                   of its 2nd output
```

The valid range of card is 0 to 203, while that for bit is 0 through 7.

RBsn

The RBs command reports whether the specified DINIO-RS485 has detected a syntax error. The card is specified by the value of n. The valid range of number is between 0 and 815 and relates to the particular DINIO-RS485 DIP switch ID number as described earlier in this document. For convenience, this is restated:

  n = 4 x DIP switch ID + input number.

That is, if the DIP switch ID is set to 2, all values of n between 8 and 15 refer to the same card. Similarly, values of number between 64 and 71 all refer to the card with the DIP switch ID of 16. The format of the reply is the global address, ASCII 128, followed by "z=" and the bit value of the card's syntax error bit.

```
Example syntax:        RBs7        'Request the syntax error bit value of
                                   card #0
```

RIOA

The RIOA causes all DINIO-RS485 on the communications bus to globally respond with the byte value of its inputs, followed by the byte value of its outputs. Each card will reply in its turn, as defined by its DIP switch sub-address, lowest sub-address first. Each card will delay its output as defined by the RADELAY parameter. The exact format of the output will be

```
  <ASCII 128>ab[DIP switch sub-address]=value<carriage return>,
```

followed by

```
<ASCII 128>ab[DIP switch sub-address + 1]=value<carriage return>.
```

Example syntax: RIOA   'Requests all cards to globally report
                                    byte value of IO states

RNn

The RN command causes the specified DINIO-RS485 card to globally respond with the value
of the requested input. The DINIO-RS485 will provide the byte or bit value depending on the
state of IOF. If the 0th bit of IOF is set, the card will transmit the byte value. If the 1st bit is set,
the card will transmit the bit value. If both are set, the card will transmit both. If neither bit is
set, the card will not reply at all. Weird, huh?

The input that you are querying is defined by the ID number n, which ranges between 0 and
815.

Example syntax: RN1   'Request the input state of input #1 on
                                    card #0
                        RN132   'Request the input state of input #4 on
                                    card #32

If the 0th bit of IOF is set, the structure of the reply will be the global address, followed by
"ab[number]="value.

The exact syntax is:

```
<ASCII 128>ab[DIP switch sub-address]=value<carriage return>.
```

Note that, since there are eight IO ID numbers per card, eight different RNn commands refer to
the same card. For example, RN0 through RN7 all do the same thing.

If the 1st bit of IOF is set, the structure of the reply will be the global address, followed by
"z="value. The exact syntax is:

```
<ASCII 128>z=value<carriage return>.
```

RNA

The RNA command causes all DINIO-RS485 cards to globally transmit the byte value of its eight
inputs. The card with the lowest DIP switch sub-address goes first. Before transmitting any-
thing, each card will wait the delay specified by RADELAY. The exact syntax of the each trans-
mission will be

```
<ASCII 128>ab[DIP switch sub-address]=value<carriage return>.
```

Example syntax: RNA

ROn

The RO command causes the specified DINIO-RS485 card to globally respond with the value
of the requested output. The DINIO-RS485 will provide the byte or bit value depending on the
state of IOF. If the 0th bit of IOF is set, the card will transmit the byte value. If the 1st bit is set,
the card will transmit the bit value. If both are set, the card will transmit both. If neither bit is
set, the card will not reply at all.

The output that you are querying is defined by the ID number n, which ranges between 0 and
815.

*If neather bit 0 or 1 of
IOF is set, the card will
not reply at all.*

Example syntax: `RO1`   `'Request the input state of input #1 on`
                                              `card #0`
                      `RO132`   `'Request the input state of input #4 on`
                                              `card #32`

If the 0th bit of IOF is set, the structure of the reply will be the global address, followed by "ab[number]="value. Note that this means RO0 through RO7 all return the same value, as do any other set of ROn commands, where n is an integral multiple of 8.The exact syntax is:

  `<ASCII 128>ab[DIP switch sub-address+1]=value<carriage return>.`

Note that, since there are eight IO ID numbers per card, eight different ROn commands refer to the same card. For example, RO0 through RO7 all do the same thing.

If the 1st bit of IOF is set, the structure of the reply will be the global address, followed by "z="value. The exact syntax is:

  `<ASCII 128>z=value<carriage return>.`

ROA

The ROA command causes all DINIO-RS485 cards to globally transmit the byte value of its eight outputs. The card with the lowest DIP switch sub-address goes first. Before transmitting anything, each card will wait the delay specified by RADELAY. The exact syntax of the each transmission will be

  `<ASCII 128>ab[DIP switch sub-address+1]=value<carriage return>.`

Example syntax: `ROA`   `'Request byte value of all outputs on`
                                              `all cards`

RSA

The RSA command will cause all DINIO-RS485 cards to globally transmit the value of its DIP switch sub-address. This happens in sequence, where the card with the lowest DIP switch sub-address goes first. Before transmitting anything, each card will wait the delay specified by RADELAY. The exact syntax of the each transmission will be

  `<ASCII 128><DIP switch sub-address><carriage return>`

Note that this is one of the very few commands that returns only a value and not an expression. This is because it has little utility in the SmartMotor™. It is envisioned that this only has value in debugging.

Example syntax: `RSA`   `'Report the switch settings of all`
                                              `cards on the bus`

RSPn

The RSPn command causes the DINIO-RS485 to globally transmit its sample period and firmware number. The card that you are querying is defined by the ID number n, which ranges between 0 and 815.

Example syntax: `RO1`   `'Request the input state of input #1 on`
                                              `card #0`
                      `RO132`   `'Request the input state of input #4 on`
                                              `card #32`

Note that this means RO0 through RO7 all return the same value, as do any other set of ROn

commands, where n is an integral multiple of 8.

The structure of the reply will be the global address, followed by the sample rate in hundredths of microseconds, a "/" and then the firmware version number. The exact syntax is:

```
<ASCII 128><sample period>"/"<version number><carriage return>.
```

Note that, since there are eight IO ID numbers per card, eight different RSPn commands refer to the same card. For example, RSP0 through RSP7 all do the same thing.

Note that this is one of the very few commands that returns only a value and not an expression. This is because it has little utility in the SmartMotor™. It is envisioned that this only has value in debugging.

RSPA

The RSPA command causes every DINIO-RS485 on the bus to globally transmit its sample period and firmware number. This happens in sequence, where the card with the lowest DIP switch sub-address goes first. Before transmitting anything, each card will wait the delay specified by RADELAY.

```
Example syntax:        RSPA         'All cards report sample period and
                                    firmware version
```

The structure of the reply will be the global address, followed by the sample rate in hundredths of microseconds, a "/" and then the firmware version number. The exact syntax is:

```
<ASCII 128><sample period>"/"<version number><carriage return>.
```

Note that this is one of the very few commands that returns only a value and not an expression. This is because it has little utility in the SmartMotor™. It is envisioned that this only has value in debugging.

ZA

The ZA commands causes all DINIO-RS485 cards to reset to their power up condition. This means that all outputs are not sourcing current, and IOF=63.

Zsn

The Zsn command resets the historical syntax error bit of the specified DINIO-RS485 card. The card whose bit that you are resetting is defined by the ID number n, which ranges between 0 and 815. Note that, since there are eight IO ID numbers per card, eight different Zsn commands refer to the same card. For example, Zs0 through Zs7 all do the same thing.

```
Example syntax:        Zs1          'Reset the syntax error bit on card #0
                       Zs123        'Reset the syntax error bit on card #15
```
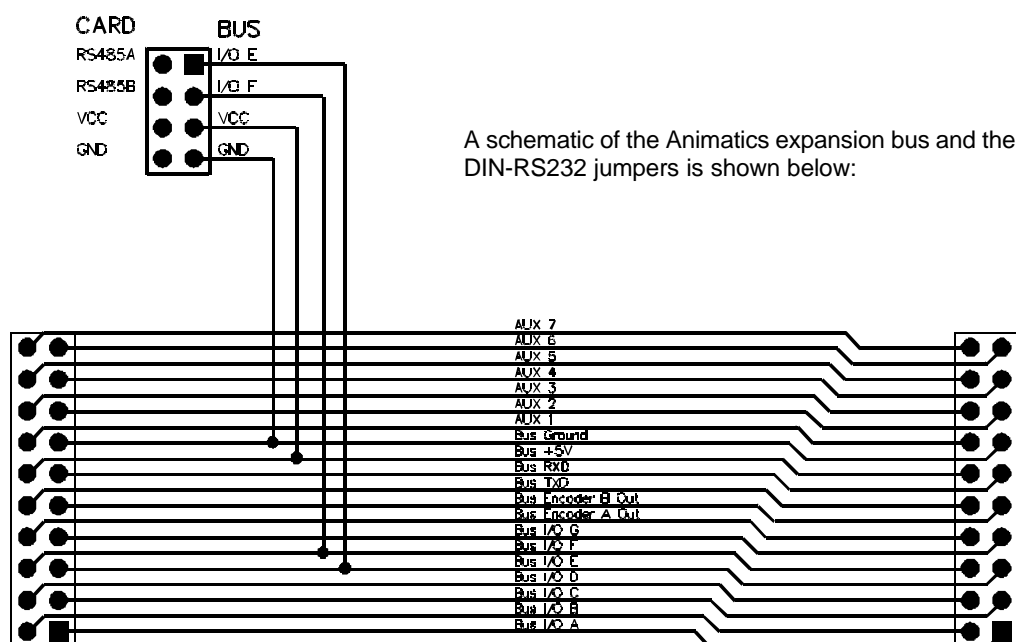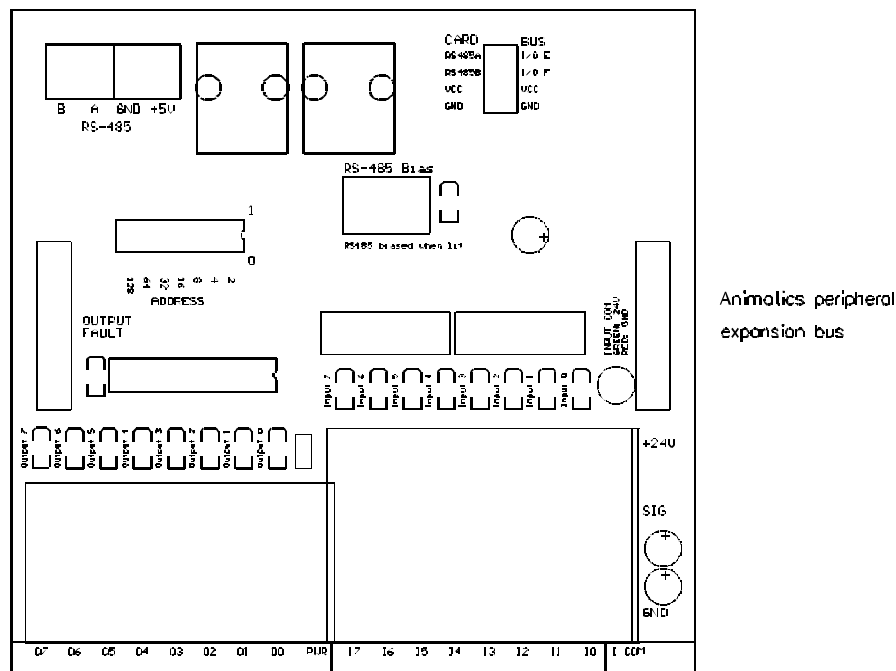
ZsA

The Zsn command resets the historical syntax error bit of the specified DINIO-RS485 card. The card with the lowest DIP switch sub-address goes first. Before transmitting anything, each card will wait the delay specified by RADELAY.

```
Example syntax:        ZsA          'Reset the syntax error bit
                                    on all cards
```

ANIMATICS PERIPHERAL EXPANSION BUS

The Animatics Expansion Bus provides a convenient way to connect several SmartMotor™s and their DIN-rail mount expansion modules without using any additional cable. Every Animators DIN-rail mount expansion module has two expansion bus connectors, for receiving signals from modules on either end of it. The bus passes through the module without any direct connection to any SmartMotor™ I/O or expansion module function. To make use of the bus, it has to be connected to some SmartMotor™ I/O or expansion bus card through the jumpers. The jumpers on the DINIO-RS485 are intended primarily to allow the user to connect the RS-485 connections, +5V power and ground to the expansion bus.

The jumpers on the DINIO-RS485 are intended primarily to allow the user to connect the RS-485 connections, +5V power and ground to the expansion bus.

Animatics peripheral expansion bus

A schematic of the Animatics expansion bus and the DIN-RS232 jumpers is shown below:

APPLICATION

The diagram below shows a typical connection among two SmartMotors, two DINIO-RS485s and a PLC. The two DINIO-RS485s have their DIP switches set to sub-addresses 0 and 2, respectively.
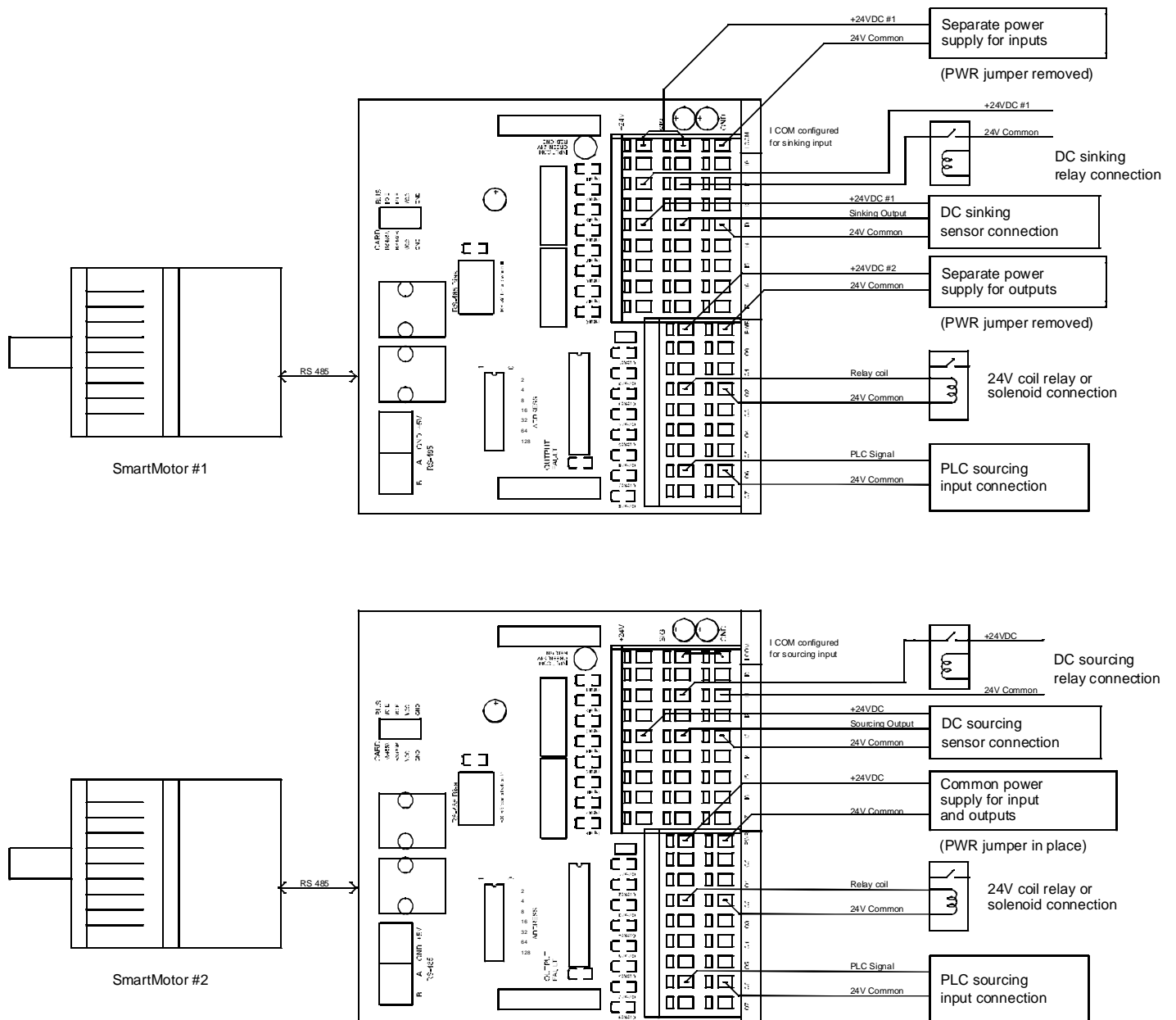
In general, the SmartMotor™ communicates with the DINIO-RS485 through its channel 1 serial port. On the SmartMotor™, the serial port is initialized through the OCHN command. Commands are then issued from the SmartMotor™ to the DINIO-RS485 to read the I/O states and set or reset the outputs.

You may have noted that, in the detailed description of the commands, the DINIO-RS485 always begins its response with a global address. This is so that every SmartMotor™ and DINIO-RS485 on the communications bus receives I/O information simultaneously, even if only one device requested the data.

After the global address, the DINIO-RS485 replies with a variable and then the appropriate value, as determined by the value of IOF. It does not reply with just a value - it always replies with an expression. This simplifies the code in the SmartMotor™. It gets the answer as if it were receiving a command.

For example, let's say that SmartMotor™ #1 issues the command RN6 and IOF=1. This means that SmartMotor™ #1 is requesting the byte value of all eight inputs of the DINIO-RS485 whose DIP switch sub-address is 0. As we are talking about byte values, the commands RN0, RN1, RB2, RN3, RN4, RN5, RN6 and RN7 all are requesting the same thing. As stated earlier, the inputs of the DINIO-RS485 are represented by negative logic - a zero value means the input has current going through it. Let's say that there is no current going through any of the inputs on card #0. The exact reply of the DINIO-RS485 will be:

```
<ASCII 128>ab[0]=255<carriage return>.
```

Let's say that inputs 0, 2 and 7 on the same card have current going through them. The exact reply would then be:

```
<ASCII 128>ab[0]=122<carriage return>.
```

It may not be intuitively obvious that this is true. Many people feel that working with byte values makes coding more efficient, while others feel that it is thinly veiled intellectual snobbery.

For those who prefer to work with individual bits, set IOF=2. This causes the DINIO-RS485 to respond with only the value of the particular input or output specified. It responds with the variable z, not ab[n]. Let's say that there is current flowing through input #6 on card #0. If IOF=2 and the command RN6 is issued, card #0 will respond with

```
<ASCII 128>z=0<carriage return>.
```

Note that this means that the SmartMotor™ should use the variable z only when dealing with expanded I/O. Similarly, the SmartMotor™ should only use the ab[n] variables when dealing with expanded I/O.

So let's tie this all together with an example. Let's suppose that we want SmartMotor™ #1 to move to position a when the DC sinking relay at input #1 closes. Once SmartMotor™ #1 completes its move, the output connected to PLC at output #6 sources for 100 milliseconds.

If you believe in working with byte values, the code in SmartMotor™ #1 would be:

```
SADDR1                          'Set the address to 1
OCHN(RS4,1,N,9600,1,8,C)        'Initialize the on-board, channel 1
                                RS485 port
PRINT1(#244,"IOF=1",#13)        'Address of all DINIO-RS485 is 244
                                  decimal
                                'Setup DINIO-RS485s to reply in bytes
WHILE 1                         'Infinite loop to scan inputs
ab[0]=255                       'Initialize ab[0] to all inactive
                                states
```

```
PRINT1(#244,"Rab[0]",#13)       'Ask card #0 to report its input byte
                                 value
WAIT=40                         'Wait 10 milliseconds for the reply
IF ab[0]&2==0                   'If input #0 is active
  ab[0]=255                     'Clear ab[0] again
  MP                            'Set up the move
  A=100
  V=200000
  P=a                           'Go to position a
  G
  TWAIT                         'Wait for the move to finish
  PRINT1(#244,"OS2",#13)         'Set output #2 on card #0
  WAIT=400                      'Delay 100 milliseconds
  PRINT(#244,"OR2",#13)         'Reset output #2 on card #0
ENDIF
LOOP
```

For those who feel that working with bytes is thinly veiled intellectual snobbery, the code in SmartMotor™ 1 becomes:

```
SADDR1                          'Set the address to 1
OCHN(RS4,1,N,9600,1,8,C)        'Initialize the on-board, channel 1
                                RS485 port
PRINT1(#244,"IOF=1",#13)        'Address of all DINIO-RS485 is 244
                                 decimal
                                'Setup DINIO-RS485s to reply in bytes
WHILE 1                         'Infinite loop to scan inputs
z=1                             'Initialize ab[0] to all inactive
                                 states
PRINT1(#244,"RN1",#13)          'Ask card #0 the value of input #0
WAIT=40                         'Wait 10 milliseconds for the reply
IF z==0                         'If the input #2 is active
  z=1                           'Clear ab[0] again
  MP                            'Set up the move
  A=100
  V=200000
  P=a                           'Go to position a
  G
  TWAIT                         'Wait for the move to finish
  PRINT1(#244,"OS2",#13)         'Turn on output #2 on card #0
  WAIT=400                      'Delay 100 milliseconds
  PRINT(#244,"OR2",#13)         'Turn off output #2 on card #0
ENDIF
LOOP
```

Note that the examples shows multiple DINIO-RS485 connected to a single SmartMotor™, but not more that one SmartMotor™ capable of interfacing with any DINIO-RS485. There is no reason to preclude this configuration, but care must be taken in the code to prevent data collisions on the RS-485 bus.