

Ultra Series Controllers

Wireless 2-Way Relay Controllers with I/O, A/D, & LCD
Expandable Using On-Board Ultra Port and XR Relay Expansion Port

5-Year Repair
or Replace
Warranty!!!



Ultra Series wireless relay controllers set new standards for performance, connectivity, and expandability, offering a universal control solution for today's most demanding computer control applications.

Ultra 16 controllers integrate 2-way wireless communications with 16 On-Board Relays, and 16 User-Programmable TTL/CMOS Data Lines (Shared with 8-Channel, 8-Bit/10-Bit Analog to Digital Converters). All data ports are detachable if they are not needed. In addition, Ultra series controllers can connect to an optional character LCD display, ideal for remote messaging applications. The Ultra Controller offer 256 levels of software controlled LCD Backlight Brightness and Contrast. Ultra series controllers have two XR Expansion Relay ports, allowing you to control a 64 banks of 8 relays, for a total of 512 relays per wireless connection. You can also control an unlimited number of Ultra Series controllers from a Single computer, offering the most scaleable control networking system available.

Control an Unlimited Number of Remote
Supports up to 115.2K Baud RS-232 Operation
Supports 40KBPS Data Transmission Speeds
Capable of Sending Commands up to 1,000 Feet
Status LEDs Indicate Device Operation

Superior 48/72-Bit Noise Rejection Protocol Ensures Reliable Consistent Operation
Includes Power Supply, Serial Cable, and Antenna

**THIS DEVICE IS NOT FCC APPROVED.
THIS DEVICE SHOULD NEVER BE USED IN LIFE SAVING APPLICATIONS.
THIS DEVICE SHOULD NEVER BE USED IN APPLICATIONS THAT COULD RESULT IN INJURY OR LOSS OF LIFE IF THE DEVICE WERE TO FAIL**

V2.0 Firmware Now Shipping

Release Date: August 22, 2005

Updated Product Description Begins on Page 37.

FAST FACTS

PLEASE READ THIS SECTION IF YOU READ NOTHING ELSE!

- The Mechanical Drawing can be found on the Product Description Page for the WIOADR16x on our web site at www.controlanything.com.
- A Terminal Program such as HyperTerminal is NOT SUITABLE for Sending Wireless Commands.
- This is a 2-Way Confirming Wireless Device
- Your program must be written to begin all data transmissions with the ASCII Character Code 254.
- Every time you send a command, ASCII Character code 85 or other data will be sent back to you upon successful communication.
- Your program must be written so that it waits for a response code after every command sent.
- The command set for a remote device resides in the remote device, the TranSendES knows nothing of the commands being sent.
- You can increase noise rejection by using "Keys" to remotely communicate with a remote device.
- You can use keys to control an unlimited number of devices, individually or simultaneously.
- A key is a device number. There are three keys that make up a complete device number.
- The Ultra Series Controllers Support 5-40KBPS wireless communication speeds.
- Dip Switches are used to select RS-232 or Wireless mode, along with appropriate communications speeds, displayed on the optional LCD display.
- You can increase wireless performance by reviewing the TranSendES product manual.
- The remote device accepts only one data packet and rejects all others, preventing commands from executing multiple times.
- A new command is defined as a new serial transmission from your computer to the AirControl transmitter.
- This device has a 64-Byte Serial Receive Buffer.
- XR Relay Expansion Ports allow you to control up to a total of 512 relays individually or simultaneously (16 on board, 496 Expanded).
- The Optional Character LCD has a Software Controlled LED Backlight and Contrast Adjustment feature, 256 of each.
- The LED Backlight is Limited to a brightness level of 40, but can be over-ridden, see this manual for more details.
- **Never mix networking methods with devices that are within radio range of each other.**

5-Year Repair or Replace Warranty

Warranty

NCD Warrants its products against defects in materials and workmanship for a period of 5 years. If you discover a defect, NCD will, at its option, repair, replace, or refund the purchase price. Simply return the product with a description of the problem and a copy of your invoice (if you do not have your invoice, please include your name and telephone number). We will return your product, or its replacement, using the same shipping method used to ship the product to NCD.

This warranty does not apply if the product has been modified or damaged by accident, abuse, or misuse.

30-Day Money-Back Guarantee

If, within 30 days of having received your product, you find that it does not suit your needs, you may return it for a refund. NCD will refund the purchase price of the product, excluding shipping/handling costs. This guarantee does not apply if the product has been altered or damaged.

Copyrights and Trademarks

Copyright 2000 by NCD. All rights reserved. Other brand and product names are trademarks of registered trademarks of their respective holders.

Disclaimer of Liability

NCD is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs or recovering, reprogramming, or reproducing any data stored in or used with NCD products.

Technical Assistance

Technical questions should be e-mailed to Ryan Sheldon at ryan@controlanything.com. Technical questions submitted via e-mail are answered up to 20 times daily. Technical support is also available by calling (417) 646-5644.

NCD Contact Information

Mailing Address:

National Control Devices
P.O. Box 455
Osceola, MO 64776

Telephone:

(417) 646-5644

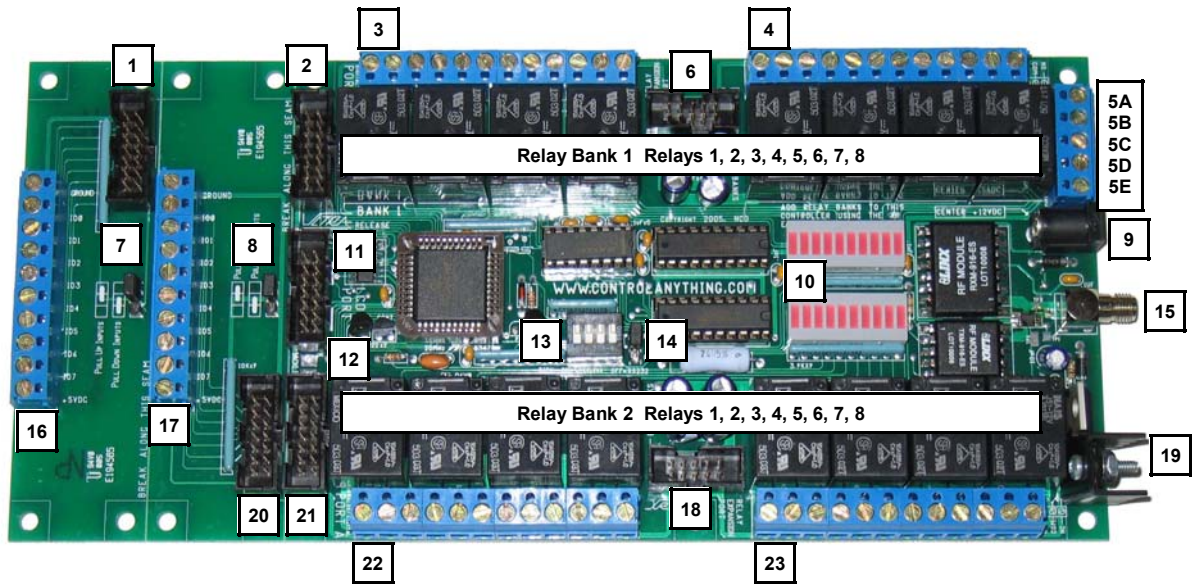
FAX:

(417) 646-8302

Internet:

ryan@controlanything.com
www.controlanything.com
www.controleverything.com

1	14-Position Connector, Connects to #2, Allows External Connection to the I/O Portion of the Controller, Cable Included
2	14-Position Connector, Connects to #1, Allows External Connection to the I/O Portion of the Controller, Cable Included
3	Relay Bank 1, Relay 1-4 Connections
4	Relay Bank 1, Relay 5-8 Connections
5	A: +12VDC B: Power Supply Ground C: RS-232 Ground D: RS-232 Input E: RS-232 Output
6	XR Relay Expansion Port, Allows You to Connect 31 Additional Banks of 8 Relays, This Port Addresses only Odd Numbered Relay Banks
7	Pull Up Pull Down Jumper. This Jumper Keep the inputs on the controller from "Floating" by pulling them to +5 or Ground through a 10K Resistor Network
8	Pull Up Pull Down Jumper. This Jumper Keep the inputs on the controller from "Floating" by pulling them to +5 or Ground through a 10K Resistor Network
9	2.1mm Center Positive +12VDC Power Connector Used to Power the Relay Controller Board, 1.25A or Greater Supply Recommended
10	LED Status Lights Indicate Communication and Relay Status. Details for this LED Bank Shown on Next Page
11	40x2 Character LCD Display Interface Connector for use with NCD 40x2 Displays ONLY
12	Solder these connections to the LED Backlight of the LCD Display. Match K and A Markings on LCD and Controller



13	DIP Switches Allow you to Set Communication Method (RS-232/Wireless) and Communication Speed (Baud/Bitrate). See Next page for detailed info.
14	RS-232/OC-232 Jumper. Set to RS-232 for Normal RS-232 Communications. Set to OC-232 when Using RSB Serial Booster on an E3C Network.
15	RPSMA Antennal Connector for 916 MHz Data Reception and Transmission
16	IO Data Connectors Allow you to Connect Sensors, Switches, and TTL Logic to the Controller
17	IO Data Connectors Allow you to Connect Sensors, Switches, and TTL Logic to the Controller
18	XR Relay Expansion Port, Allows You to Connect 31 Additional Banks of 8 Relays, This Port Addresses only Even Numbered Relay Banks
19	Heat Sink may get hot, please allow 2" of height for this component. Some ventilation preferred but not required for some installations.
20	14-Position Connector, Connects to #21, Allows External Connection to the I/O Portion of the Controller, Cable Included
21	14-Position Connector, Connects to #20, Allows External Connection to the I/O Portion of the Controller, Cable Included
22	Relay Bank 2, Relay 1-4 Connections
23	Relay Bank 2, Relay 5-8 Connections

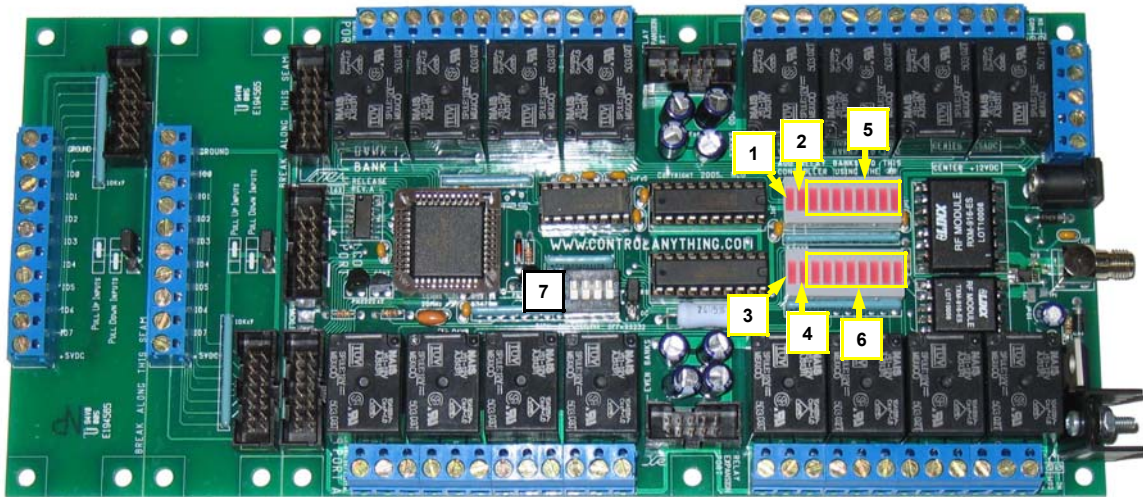
INCLUDED ACCESSORIES: Antenna

RECOMMENDED WIRELESS ACCESSORIES:
Power Supply
TS916ES Wireless Device Communicator

RECOMMENDED RS232 ACCESSORIES:
QS12 Quick Start Kit
RSB Serial Booster (When Using an E3C Network)

1	Waiting for Command (OFF WHEN POWERED UP IN WIRELESS MODE, FLASHING WHEN POWERED UP IN RS232 MODE)
2	Ready to Receive LED (ON WHEN POWERED UP)
3	Device Enabled LED, LED is Toggled by Device Networking Commands (ON WHEN POWERED UP)
4	DATA Send LED, LED Flashes when Data is Sent via Wireless or RS232 Communication (OFF WHEN POWERED UP)
5	Left to Right: Relay Bank 1, Relays 1-8 LED Status Indicators
6	Left to Right: Relay Bank 2, Relays 1-8 LED Status Indicators

NOTICE: Processor may be damaged if LEDs 1 and 3 Stay Lit and All Other LEDs are OFF when Powered UP



7 | DIP Switches Allow you to Set Communication Method (RS-232/Wireless) and Communication Speed (Baud/Bitrate). See Below for Details

Dip Switch 1: OFF RS-232 Communication Mode				Dip Switch 1: ON Wireless Communication Mode			
DIP 2	DIP 3	DIP 4	Baud Rate	DIP 2	DIP 3	DIP 4	Bit Rate
OFF	OFF	OFF	38.4K**	OFF	OFF	OFF	40K**
ON	OFF	OFF	2400	ON	OFF	OFF	5K
OFF	ON	OFF	4800	OFF	ON	OFF	7.5K
ON	ON	OFF	9600	ON	ON	OFF	14.4K
OFF	OFF	ON	19.2K	OFF	OFF	ON	24K
ON	OFF	ON	38.4K	ON	OFF	ON	33.6K
OFF	ON	ON	57.6K	OFF	ON	ON	38.4K
ON	ON	ON	115.2K	ON	ON	ON	40K

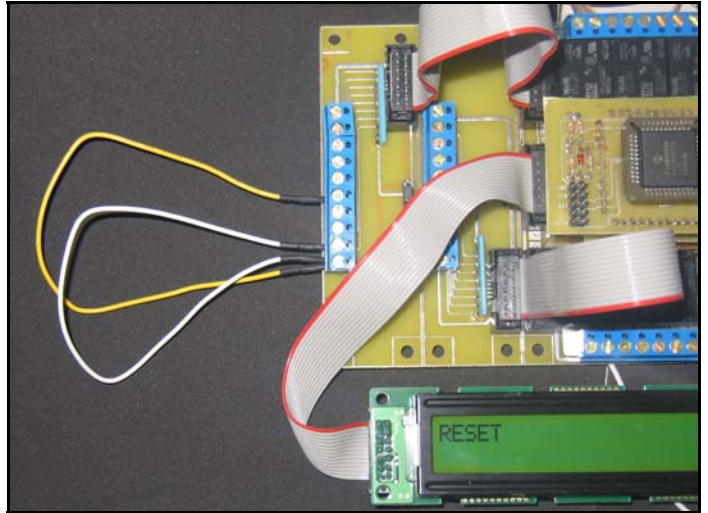
**** Test Mode Used for Configuring Controller. Networking Commands do NOT Function in this mode. Test Mode is REQUIRED for Programming RS-232 E3C Device Numbers, Wireless Device Number Keys, and Hardware RESET Function. Some Setting Will Have No Effect Until Controller Is Removed from Test Mode**

Hardware Reset Procedure

This procedure can be equated with clearing the bios on a computer. The photo shows our last prototype wired for a hardware RESET. The Pull-Up/Pull-Down Jumper is in the Pull-Down Position (lower two posts connected). A hardware RESET can only be initiated while in TEST MODE, the connections for a hardware test are shown in the photo. There are two wires installed. The first wire connects between the +5VDC and IO4. The second wire connects between the +5VDC and IO7. The hardware RESET feature will restore EEPROM settings (except device numbers) to factory default values.

This procedure is not likely to be needed, however, if you ever experience any communications problems after changing some vital settings, this procedure will restore the controller to known safe settings. Connect all wiring as shown and then power cycle the controller. After the splash screen appears, the LCD will flash some other information very quickly. Next, the word "RESET" will appear in the upper left corner of the LCD display upon successful completion of this procedure. Next, you should remove the wiring from the controller and power cycle the controller for normal operation.

If problems persist, it may be necessary to set the TranSend to factory default settings, which may greatly slow communications, but should resolve any problems you may be having.



Sending Wireless vs. RS-232 Commands

As you probably know, the Ultra series controllers can be controlled from a wireless connection or via the built-in RS-232 port, allowing wireless or direct wired communication to the controller. It is NOT possible to communicate via wireless and RS-232 at the same time.

We will discuss the actual mechanics of sending wireless and RS-232 commands on the next page of this manual. The most important thing to understand about sending commands is that the command structure is the same, whether you are communicating wirelessly or via RS-232.

In fact, communicating to a wireless device is almost exactly like communicating to a directly wired RS-232 device. Most commands are the same, the way these commands function is the same, and the results will be the same.

The only exception being that commands used for wireless EWC networking are slightly different than commands used for RS-232 E3C networking.

It is also important to realize that some commands only function in test mode. In general, test mode should be used for configuration and diagnostic purposes only. Some commands will not function properly in test mode (such as networking commands). Also, test mode communications data rates are typically slower to ensure maximum communications reliability.

Test mode is used to configure and test basic functions of the controller. It can be equated with "Safe Mode" on a windows system. You MUST use test mode to configure some parameters, such as an E3C device number, Default Configuration Bits, and EWC keys, and many more parameters.

To place an Ultra controller in test mode, dip switches 2, 3, and 4 must be in the off position when power is applied to the board. Dip switch 1 chooses communications method. In the on position, the Ultra controller will only accept wireless commands. In the off position, commands will only be accepted from the RS-232 serial port. Again, all dip switches must be set prior to powering the controller.

If you have the optional character LCD display, then you will always know when you are in test mode. Test mode is limited to 38.4K Baud via RS-232 or 40KBPS via wireless connection.

In test mode, it is normal for the relays to cycle through a simple test pattern. Test mode will display the E3C device number or the EWC Keys if using wireless communications.

Keep in mind, EWC Keys are used to designate a wireless device number. An E3C device number is used to identify a directly wired device connected to the serial port. Keys are specific to wireless devices only. E3C device numbers are specific to RS-232 directly wired devices only.



To Enter RS-232 Test Mode:

Turn Off Dip Switches 1-4.
Power Up Controller.



To Enter Wireless Test Mode:

Turn On Dip Switch 1.
Turn Off Dip Switches 3-4.
Power Up Controller.

Sending Commands to NCD Devices

NCD Devices are capable of sending and receiving data via RS-232 serial communications and are compatible with just about any computer or microcontroller ever produced, including the Macintosh, Amiga, Basic Stamp, and of course, Windows & DOS based machines.

Regardless of the system you are using, you will need access to a programming language that supports program control of the serial port on your system.

A terminal program is not suitable for sending commands to NCD devices. Commands should be sent using ASCII character codes 0-255 rather than ASCII characters (A, B, C etc.). See "ASCII Codes vs. Characters" on this page.

Most systems require you to open the appropriate serial port (COM port) prior to sending or receiving data.

Because there are so many different ways to send and receive data from various languages on various platforms, we will provide generic instructions that can be easily converted to your favorite language.

For example, if this manual says "Send ASCII 254", the user will need to translate this instruction into a command that is capable of sending ASCII character code 254.

To Send ASCII 254 from Visual Basic, you will use the following line:

```
MSComm1.Output = Chr$(254)
```

In Qbasic, you can send ASCII 254 using the following line of code:

```
Print #1, Chr$(254);
```

Note that sending ASCII character code 254 is NOT the same as sending ASCII characters 2, 5, and 4 from a terminal program. Typing 2, 5, and 4 on the keyboard will transmit three ASCII character codes.

In your program, you will need to make provisions for reading data from the serial port. Your programming language will support commands for reading data from NCD devices.

For your convenience, we have provided several programming examples in Visual Basic 6 for communicating with our products. These examples should greatly speed development time. You may want to visit www.controleverything.com for the latest software and programming examples.

Programming examples for NCD devices are much more extensive for Visual Basic 6 users than for any other programming language. If you are not a VB programmer, you may consider looking at the VB6 source code, as it is easily translated into other popular languages.

Regardless of your programming background, the provided Visual Basic 6 source code is very easy to understand and will likely resolve any communication questions you may have. VB6 programming examples may be viewed in any text editor.

ASCII Codes vs. Characters

The differences between ASCII codes and ASCII characters tend to generate a lot of confusion among first-time RS-232 programmers. It is important to understand that a computer only works with numbers. With regard to RS-232 data, the computer is only capable of sending and receiving numbers from 0 to 255.

What confuses people is the simple idea that the numbers 0 to 255 are assigned letters. For instance, the number 65 represents the letter A. The number 66 represents the letter B. Every character (including numbers and punctuation) is assigned a numeric value. This standard of assignments is called ASCII, and is a universal standard adopted by all computers with an RS-232 serial port.

ASCII characters codes can be clearly defined as numbers from 0 to 255.

ASCII characters however are best defined as letters, A, B, C, D, as well as punctuation, !@#\$%, and even the numbers 0-9.

Virtually all programming languages permit you to send ASCII in the form of letters or numbers. If you wanted to send the word "Hello" out the serial port, it is much easier to send the letters H, e, l, l, and o than it is to send the ASCII character codes that represent each letter.

For the purposes of controlling NCD devices however, it is much easier to build a numeric command set. Especially when communicating to devices where you want to speak to lots of outputs (which are numbered), inputs (which are also numbered), or control specific devices using their device number (from 0 to 255).

Put simply, it is easier to control NCD devices using ASCII character codes 0 to 255 than it is to use ASCII characters A, B, C, D, etc.

Because terminal programs are ASCII character based, it may be difficult to generate the proper series of keystrokes that would be necessary to activate a particular function. Therefore, they are not suitable for controlling NCD devices. In a real world control application, a terminal program would not likely be used to control NCD devices anyway. Therefore, a programming language that supports the transmission and reception of ASCII character codes 0 to 255 is highly recommended.

Wireless or RS-232?

It doesn't matter which communication method you choose, the Ultra Series Relay Controllers use the same communication structure for either one. Weather you are sending RS-232 commands directly to the Ultra controller or you are sending wireless commands via the TransSendES wireless device communicator, there is really not much of a difference, the information on this page applies to both communication methods.

Before Going any Further....

We highly recommend installing our Ultra Series Configuration and Test Software, Available for free download from our web site. The Example Source Code and .EXE programs will be references heavily in this manual. This Software can be found in the product description page for the WIOADR165 or WIOADR1610 relay controllers on our web site at www.controlanything.com. This comprehensive software package includes extensive source code for VB6 as well as a full install program and many compiled .exe programs suitable for immediate testing of controller functions.

Configuring Ultra Series Controllers

Ultra Series Controllers can be configured for advanced options and settings while in test mode (turn off dip switches 2, 3, and 4 and power cycle the controller).

Ultra Series Controllers are configured using the "Ultra Configuration and Test Utility". This application can be used with RS-232 or wireless communications, the only requirement being the controller MUST be in test mode. When in test mode, configuration memory can be modified. While in normal runtime operation, configuration memory is write protected, preventing accidental changing of vital settings.

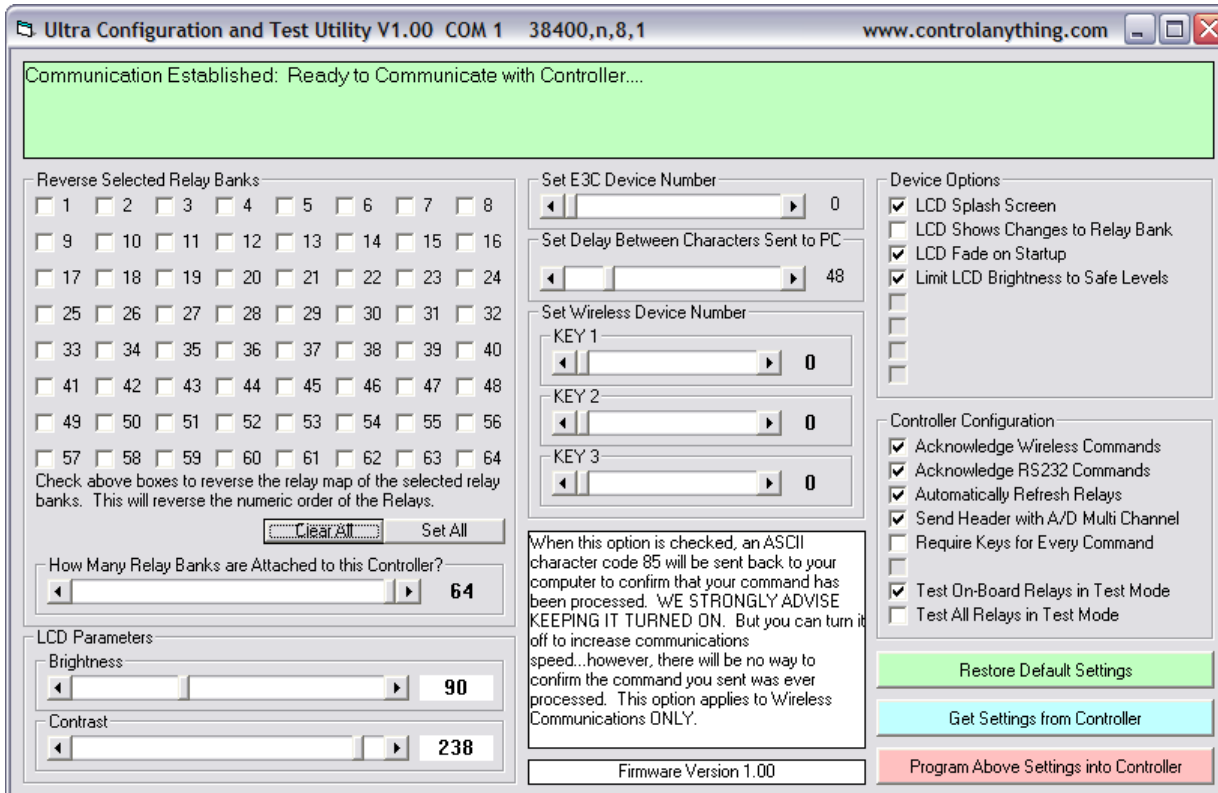
Since the remote device must be in test mode during configuration, it is possible to modify settings on an individual device, even when there are neighboring devices within radio range. However, you should NEVER have more than one device in test mode at one time. Doing so can cause corruption of important settings that may require a hardware reset (discussed on the fourth page of this manual).

The following commands are used exclusively by the Ultra Configuration and Test Utility. We DO NOT recommend issuing these commands without carefully examining the source code. Though these settings are worthy of mention, we DO NOT encourage their use until you have studied our controller carefully. These commands are critical to the proper functionality of our Ultra series controllers and are best left to our Ultra Configuration and Test Utility.

IF YOU INSIST ON EXPERIMENTING WITH THESE COMMANDS, IT MAY BECOME NECESSARY TO ISSUE A HARDWARE RESET SHOULD THE CONTROLLER BECOME UNRESPONSIVE. SEE THE FOURTH PAGE OF THIS MANUAL TO RECOVER THE CONTROLLER TO FACTORY DEFAULT SETTINGS.

- 254, 0, 0 Reply for 2-Way Communications Test
Returns 85 when in Normal Mode
Returns 86 when in Test Mode
- 254, 0, 1 Recovers Default Settings
Returns 85 when Complete
- 254, 0, 2 Store Settings into Controller 25 Parameters are Stored
Data Storage should be done using the Ultra Series Configuration Utility (Source Included).
- 254, 0, 3 Retrieve Settings from Controller 25 Parameters are Returned. Data Should be Retrieved using the Ultra Series Configuration Utility (Source Included)
- 254, 0, 4 Reports Back the Settings of the DIP Switches
- 254, 0, 5, Configuration Word Bit Setup Function
Parameters (0-15) Clear Configuration Word Bits Below.
Parameters (16-31) Set Configuration Word Bits Below.
 - Bit 0: LCD Splash Screen Settings
 - Bit 1: LCD Shows Changes to Relay Bank
 - Bit 2: LCD Fade on Startup
 - Bit 3: Limit LCD Brightness to Safe Levels
 - Bit 4: Not Used
 - Bit 5: Not Used
 - Bit 6: Not Used
 - Bit 7: Not Used
 - Bit 8: Acknowledge Wireless Commands
 - Bit 9: Acknowledge RS232 Commands
 - Bit 10: Automatically Refresh Relays
 - Bit 11: Sent header with A/D Multi-Channel Commands
 - Bit 12: Require Keys for Every Command
 - Bit 13: Not Used
 - Bit 14: Test On-Board Relays in Test Mode
 - Bit 15: Test All Relays in Test Mode

You may clear or set any of these bits at any time, even when the controller is NOT in test mode. The Power-Up default values for these bits can only be changed in Test mode. You should use the configuration utility to better understand what these options do (use the white box in the lower middle of the configuration screen).



I/O Data Ports: The Basics of Inputs, Outputs, and Bit Manipulation:

Ultra Series Controllers have two 8-Bit I/O ports. I/O ports can be used to read the on/off status of switches and sensors. These I/O ports can also be used as TTL/CMOS compatible outputs.

For those of you who are new to electronics, a TTL/CMOS input/output can be a little complicated to understand. So here, we will discuss the basics of what all this means.

TTL/CMOS compatibility simply means the controller uses a standard 0 volts and +5 volt logic system. These are standard voltages used by many TTL/CMOS compatible chips, microcontrollers, and other electronic devices. In other words, zero volts is logic level 0 (OFF) while +5 volts is logic level 1 (ON). So when you think of CMOS/TTL compatibility, it just means zero volts = OFF = 0 while +5 volts = ON = 1.

Understanding Inputs and Outputs:

The I/O ports on the Ultra Series controllers can be configured as inputs or outputs. In other words, these data ports can be used to read 0 or +5 volt signals, or they can be used to deliver 0 or +5 volt signals. In a practical application, inputs are useful for reading switches while outputs are useful for lighting LEDs. Inputs and outputs can be used for thousands of different applications.

Using Outputs

As you probably already know, there are two 8-Bit I/O data ports on the Ultra series controllers, providing a total of 16 data lines that can be used for just about any application you can think of. If you were to use them as outputs, you could connect these 16 data lines to 16 LEDs. You could very easily write a program that will turn each of the LEDs on or off in any combination you choose. When an output is turned on, the output will deliver +5 volts out the port pin. When the output is turned off, the output will be set to zero volts.

Output Precautions

You will quickly damage the outputs if any of these pins are accidentally connected to +5V or Ground while in output mode. For this reason, it is pretty typical to connect a port pin to a 220 Ohm resistor before connecting it to anything else. It may be OK to use higher value resistors, but never go below 220 Ohms if you choose to use this protection method.

Using Inputs

Inputs are useful for reading the on/off status of switches and other sensors. Because this is a TTL/CMOS compatible device, each input is expected to have +5VDC or zero volts on the inputs. When you request an input status, the controller will return a 0 or 1 based on the status of the input. Keep in mind, there are 16 I/O lines that you can use as inputs or outputs. This allows you to read 16 switches or turn on/off 16 LEDs.

Input Precautions

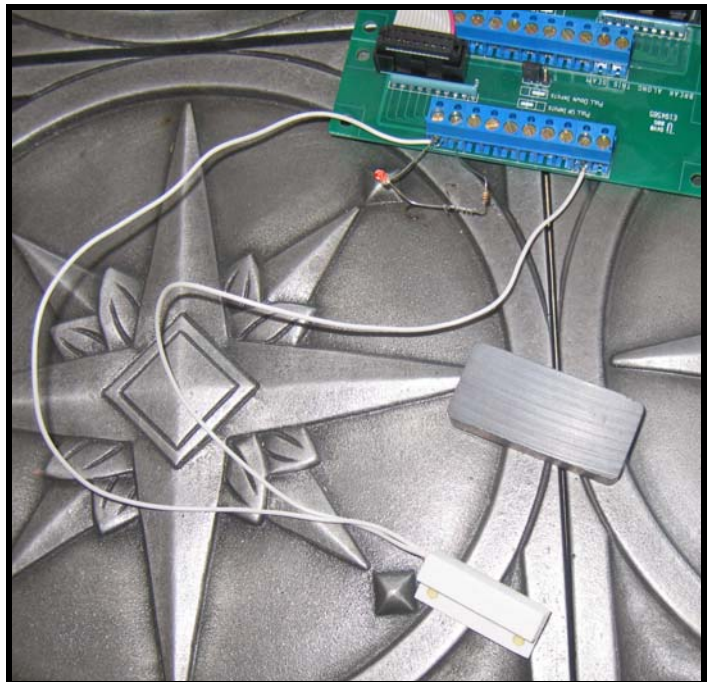
Inputs should never exceed the 0-5VDC voltage range. In addition, it is very important to keep inputs from floating. A floating input is an input that is not connected to anything. As a rule, you should NEVER leave an input floating. Floating inputs can be read as a 0 or a 1 in your program with no predictability. Inputs should always connect to ground or +5VDC. For this reason, we have provided a pull-up/pull-down jumper on each set of inputs. When this jumper is set to pull-up, inputs will be read as logic level 1. When set to pull-down, inputs will read as logic level 0. These resistors are used to keep the inputs quiet. They have no affect and will not damage the controller if the port is used as an output. You can remove the jumper entirely to allow the inputs to float.

Mixing Inputs and Outputs:

You can use the 16 I/O data lines in any way you wish. Inputs and outputs can be mixed on the same I/O data port. Or you may choose to separate inputs on one data port and outputs on another data port. You decide how to use the I/O data ports. Later, we will discuss the use of the built-in analog-to-digital converts. The entire port is used for A/D conversion, so it is not possible to mix I/O on port A if A/D conversion functions are used.

Reading Outputs

We have added a new function to the Ultra Series controllers. This new function allows you to ask the controller what the output status is. In other words, you can send a command to turn on an LED. You can then ask the controller if the LED is ON or OFF by reading the output status. This is not to be confused with reading an input status, which would turn off the LED and read the voltage on the port pin, returning a 0 or 1 to your program.



The Photo Above Shows an LED connected to an I/O data line on Port B through a 220 Ohm current limiting resistor. The Cathode of the LED is connected to ground. The anode is connected to the resistor. The other side of the resistor is connected to IO0. Sending ASCII Character Code 254, 7, 24 turns on the LED. Sending ASCII Character Code 254, 7, 8 turns off the LED.

This photo also Shows a magnetic switch connected between the data line IO7 on the I/O data port and ground. The data buss is set to "Pull Up" mode. When data line 7 is read, the controller will always return a 1 because the input is pulled up. Placing the magnet near the magnet reed switch will ground out IO7, causing the controller to report back a 0. To read the input, simply send ASCII Character Codes 254, 7, 47. The controller will report back a 1 if the magnet is away from the switch or a 0 if the magnet is near the switch.

I/O Data Ports: Digital I/O Test Program

Port Bit Manipulation Functions are demonstrated using the "Port Bit Functions" program installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

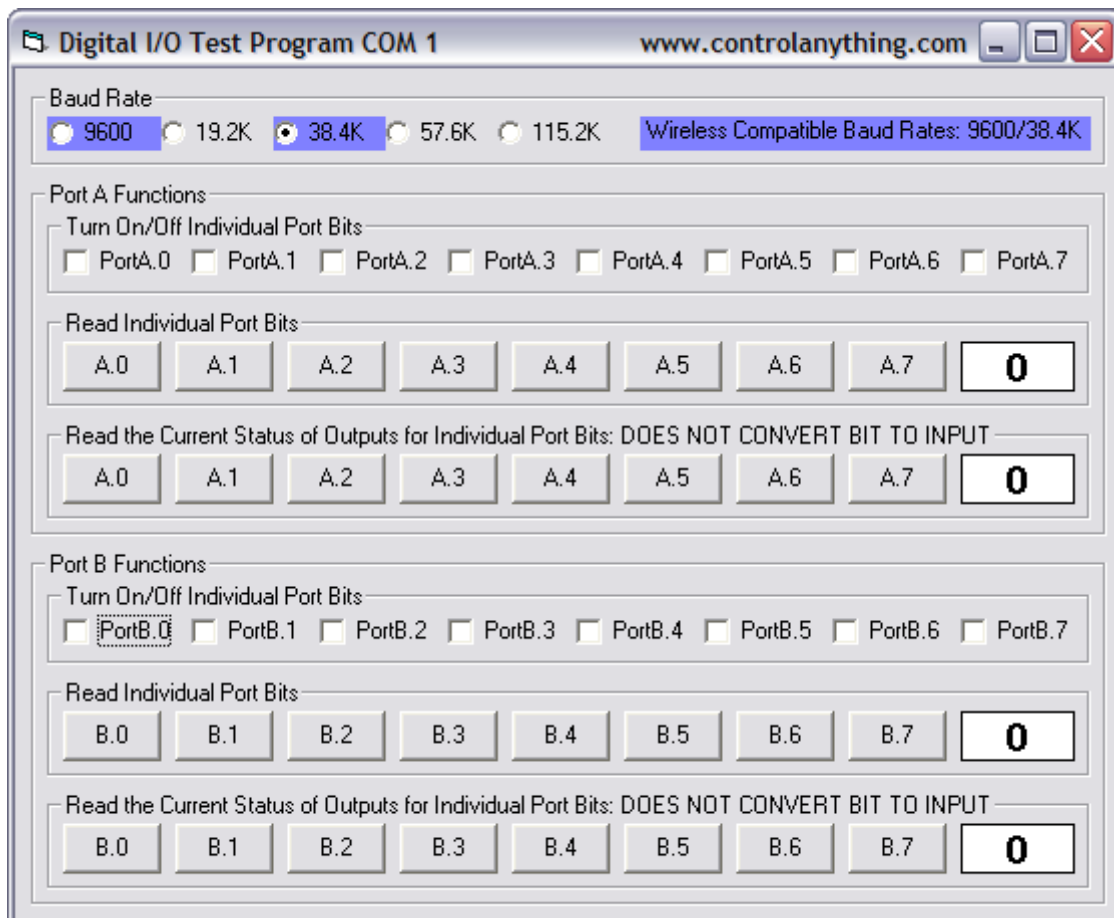
Turning On/Off Outputs on Port A and B.

Reading Inputs on Port A and B.

Reading Output Status on Port A and B.

This program provides a simple GUI that allows you test the bit manipulation functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.



I/O Data Ports: Bit Manipulation Commands

The sample code shown at right comes directly from the Port Bit Functions program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Keep in mind, Ultra Series controllers are 2-way native devices, and will report back ASCII character code 85 to confirm the operation has completed. In the examples shown at right, the GetI function is used to make sure the serial port buffer is empty prior to issuing a command. This will help prevent mis-read data. The GetI function is shown at right. All source code is written in Visual Basic 6.

The following commands will set the data port pins to output mode. Make sure these lines are NEVER connected directly to ground or +5V when using these commands.

254, 7, 0	Sets Port A Bit 0 to Output Zero Volts
254, 7, 1	Sets Port A Bit 1 to Output Zero Volts
254, 7, 2	Sets Port A Bit 2 to Output Zero Volts
254, 7, 3	Sets Port A Bit 3 to Output Zero Volts
254, 7, 4	Sets Port A Bit 4 to Output Zero Volts
254, 7, 5	Sets Port A Bit 5 to Output Zero Volts
254, 7, 6	Sets Port A Bit 6 to Output Zero Volts
254, 7, 7	Sets Port A Bit 7 to Output Zero Volts
254, 7, 8	Sets Port B Bit 0 to Output Zero Volts
254, 7, 9	Sets Port B Bit 1 to Output Zero Volts
254, 7, 10	Sets Port B Bit 2 to Output Zero Volts
254, 7, 11	Sets Port B Bit 3 to Output Zero Volts
254, 7, 12	Sets Port B Bit 4 to Output Zero Volts
254, 7, 13	Sets Port B Bit 5 to Output Zero Volts
254, 7, 14	Sets Port B Bit 6 to Output Zero Volts
254, 7, 15	Sets Port B Bit 7 to Output Zero Volts
254, 7, 16	Sets Port A Bit 0 to Output +5 Volts
254, 7, 17	Sets Port A Bit 1 to Output +5 Volts
254, 7, 18	Sets Port A Bit 2 to Output +5 Volts
254, 7, 19	Sets Port A Bit 3 to Output +5 Volts
254, 7, 20	Sets Port A Bit 4 to Output +5 Volts
254, 7, 21	Sets Port A Bit 5 to Output +5 Volts
254, 7, 22	Sets Port A Bit 6 to Output +5 Volts
254, 7, 23	Sets Port A Bit 7 to Output +5 Volts
254, 7, 24	Sets Port B Bit 0 to Output +5 Volts
254, 7, 25	Sets Port B Bit 1 to Output +5 Volts
254, 7, 26	Sets Port B Bit 2 to Output +5 Volts
254, 7, 27	Sets Port B Bit 3 to Output +5 Volts
254, 7, 28	Sets Port B Bit 4 to Output +5 Volts
254, 7, 29	Sets Port B Bit 5 to Output +5 Volts
254, 7, 30	Sets Port B Bit 6 to Output +5 Volts
254, 7, 31	Sets Port B Bit 7 to Output +5 Volts

Turn Off Port A Bit 0

```
MSCComm1.Output = Chr$(254)      'Enter Command Mode
MSCComm1.Output = Chr$(7)        'Port Bit Functions
MSCComm1.Output = Chr$(0)        'Turn Off Port Bit A.0
GetI                               'Wait for Command to Finish
```

Turn On Port A Bit 0

```
MSCComm1.Output = Chr$(254)      'Enter Command Mode
MSCComm1.Output = Chr$(7)        'Port Bit Functions
MSCComm1.Output = Chr$(16)       'Turn On Port Bit A.0
GetI                               'Wait for Command to Finish
```

Turn Off Port B Bit 7

```
MSCComm1.Output = Chr$(254)      'Enter Command Mode
MSCComm1.Output = Chr$(7)        'Port Bit Functions
MSCComm1.Output = Chr$(23)       'Turn Off Port Bit B.7
GetI                               'Wait for Command to Finish
```

Turn On Port B Bit 7

```
MSCComm1.Output = Chr$(254)      'Enter Command Mode
MSCComm1.Output = Chr$(7)        'Port Bit Functions
MSCComm1.Output = Chr$(31)       'Turn On Port Bit B.7
GetI                               'Wait for Command to Finish
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
        Loop Until MSCComm1.InBufferCount > 0
        GetI = Asc(MSCComm1.Input)
    End Function
```

I/O Data Ports: Bit Manipulation Commands

The sample code shown at right comes directly from the Port Bit Functions program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Keep in mind, Ultra Series controllers are 2-way native devices, and will report back ASCII character code 85 to confirm the operation has completed. All source code is written in Visual Basic 6.

The following command will automatically switch a data line to input mode prior to reading the port pin. Do NOT exceed the 0-5VDC voltage input.

254, 7, 32 Reads Port A Bit 0 Input, Returns 0 or 1
254, 7, 33 Reads Port A Bit 1 Input, Returns 0 or 1
254, 7, 34 Reads Port A Bit 2 Input, Returns 0 or 1
254, 7, 35 Reads Port A Bit 3 Input, Returns 0 or 1
254, 7, 36 Reads Port A Bit 4 Input, Returns 0 or 1
254, 7, 37 Reads Port A Bit 5 Input, Returns 0 or 1
254, 7, 38 Reads Port A Bit 6 Input, Returns 0 or 1
254, 7, 39 Reads Port A Bit 7 Input, Returns 0 or 1

254, 7, 40 Reads Port B Bit 0 Input, Returns 0 or 1
254, 7, 41 Reads Port B Bit 1 Input, Returns 0 or 1
254, 7, 42 Reads Port B Bit 2 Input, Returns 0 or 1
254, 7, 43 Reads Port B Bit 3 Input, Returns 0 or 1
254, 7, 44 Reads Port B Bit 4 Input, Returns 0 or 1
254, 7, 45 Reads Port B Bit 5 Input, Returns 0 or 1
254, 7, 46 Reads Port B Bit 6 Input, Returns 0 or 1
254, 7, 47 Reads Port B Bit 7 Input, Returns 0 or 1

The following commands read the current status of the outputs. These commands do NOT convert a port pin to an Input. They simply tell you weather the output pin is set to on or off, 1 or 0.

254, 7, 48 Read Port A Bit 0 Output, Returns 0 or 1
254, 7, 49 Read Port A Bit 1 Output, Returns 0 or 1
254, 7, 50 Read Port A Bit 2 Output, Returns 0 or 1
254, 7, 51 Read Port A Bit 3 Output, Returns 0 or 1
254, 7, 52 Read Port A Bit 4 Output, Returns 0 or 1
254, 7, 53 Read Port A Bit 5 Output, Returns 0 or 1
254, 7, 54 Read Port A Bit 6 Output, Returns 0 or 1
254, 7, 55 Read Port A Bit 7 Output, Returns 0 or 1

254, 7, 56 Read Port B Bit 0 Output, Returns 0 or 1
254, 7, 57 Read Port B Bit 1 Output, Returns 0 or 1
254, 7, 58 Read Port B Bit 2 Output, Returns 0 or 1
254, 7, 59 Read Port B Bit 3 Output, Returns 0 or 1
254, 7, 60 Read Port B Bit 4 Output, Returns 0 or 1
254, 7, 61 Read Port B Bit 5 Output, Returns 0 or 1
254, 7, 62 Read Port B Bit 6 Output, Returns 0 or 1
254, 7, 63 Read Port B Bit 7 Output, Returns 0 or 1

Reading Port A Input Bit 0

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(32) 'Read Port A Bit 0
GetI 'GetI Contains Input Value
```

Reading Port A Input Bit 5

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(37) 'Read Port A Bit 5
GetI 'GetI Contains Input Value
```

Reading Port B Input Bit 1

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(41) 'Read Port B Bit 1
GetI 'GetI Contains Input Value
```

Reading Port B Input Bit 7

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(47) 'Read Port B Bit 7
GetI 'GetI Contains Input Value
```

Reading Port A Output Status of Bit 0

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(48) 'Read Port A Output Status Bits
GetI 'GetI Contains Output Value
```

Reading Port B Output Status of Bit 5

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(61) 'Read Port B Output Status Bits
GetI 'GetI Contains Output Value
```

Reading Port B Output Status of Bit 7

```
MSComml.Output = Chr$(254) 'Enter Command Mode
MSComml.Output = Chr$(7) 'Port Bit Functions
MSComml.Output = Chr$(63) 'Read Port A Output Status Bits
GetI 'GetI Contains Output Value
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComml.InBufferCount > 0
    GetI = Asc(MSComml.Input)
End Function
```

I/O Data Ports: Introduction to Byte Manipulation Commands

Sometimes it is useful to read a group of inputs at one time, or even set the status of a group of outputs. Byte manipulation commands allow you to read and write data directly to an I/O data port, 8-bits at a time.

Updating 8 Outputs at One Time:

Sending a byte of data directly to a data port is very fast and easy. Using commands for updating data bytes on a data port force the data port to function like a serial to parallel converter. For instance:

Send ASCII Character Codes 254, 1, 0:

This command sets output port A to digital outputs. The 1 is used to send the Port Byte function for port A. The 0 will clear the data port to a value of 0. Once this command has executed, all data lines will be off, %00000000.

Send ASCII Character Codes 254, 1, 255:

The similar command turns on all data lines, %11111111.

Send ASCII Character Codes 254, 1, 85:

The similar command turns on every other data line: %01010101.

Send ASCII Character Codes 254, 1, 170:

The similar command turns on every other data line in the reverse pattern: %10101010.

Binary is always read from the most significant bit to the least significant bit:

%10000000	= 128	IO7 = ON, All Others OFF
%01000000	= 64	IO6 = ON, All Others OFF
%00100000	= 32	IO5 = ON, All Others OFF
%00010000	= 16	IO4 = ON, All Others OFF
%00001000	= 8	IO3 = ON, All Others OFF
%00000100	= 4	IO2 = ON, All Others OFF
%00000010	= 2	IO1 = ON, All Others OFF
%00000001	= 1	IO0 = ON, All Others OFF

Binary is combinational. Adding 2 or more bits together turns on/off different outputs in the equivalent binary pattern:

%00000000	= 0	IO1 = ON, All Others OFF
%00000001	= 1	IO0 = ON, All Others OFF
%00000011	= 3	IO0 and IO1 = ON, All Others OFF
%00000111	= 7	IO0, IO1, IO2 = ON, All Others OFF
%00001111	= 15	IO0-IO3 = ON, All Others OFF
%00011111	= 31	IO0-IO4 = ON, All Others OFF
%00111111	= 63	IO0-IO5 = ON, All Others OFF
%01111111	= 127	IO0-IO6 = ON, IO7 OFF
%11111111	= 255	IO0-IO7 = ON

The binary mechanics of reading data bits are exactly the same as writing data bits. Later, when you ask the controller for the status of a group of inputs. Simply convert the returned value to a binary value and you will immediately "see" which data lines are on and off.

Here are a few Visual Basic tricks that will help you determine if data bits are on or off. I have had a lot of requests for some sample code to test data bits in VB. In this example, we will assume you have 8 door sensors attached to Port A on the controller. Your program needs to quickly read the data port to determine which door sensors show that the door is OPEN (on). If the door is closed, a "0" will be read from the controller.

We will assume you have asked the controller for the current status of the inputs on port A. In this scenario, the controller has sent you back a value of 75. So which doors are open and which doors are closed?

VB Sample Code:

```
Door = 75
If (Door And 1) = 1 Then Debug.Print "Door 1 is Open"
If (Door And 2) = 2 Then Debug.Print "Door 2 is Open"
If (Door And 4) = 4 Then Debug.Print "Door 3 is Open"
If (Door And 8) = 8 Then Debug.Print "Door 4 is Open"
If (Door And 16) = 16 Then Debug.Print "Door 5 is Open"
If (Door And 32) = 32 Then Debug.Print "Door 6 is Open"
If (Door And 64) = 64 Then Debug.Print "Door 7 is Open"
If (Door And 128) = 128 Then Debug.Print "Door 8 is Open"
```

The Debug Window Prints the following:

```
Door 1 is Open
Door 2 is Open
Door 4 is Open
Door 7 is Open
```

In this example, the "And" math function is used to test each of the data bits. If the incoming Door value matches the appropriate bits, alerts will be displayed showing each door that is open. The "And" function is NOT exclusive to Visual Basic. Many other languages support the "And" function, which is a basic math function. Some languages will use different syntax such as "&" in place of the word "And", other languages may use "&&". Consult the math functions section of the manual of your favorite programming language to determine the syntax for using the "And" function.

I/O Data Ports: Digital I/O Test Program

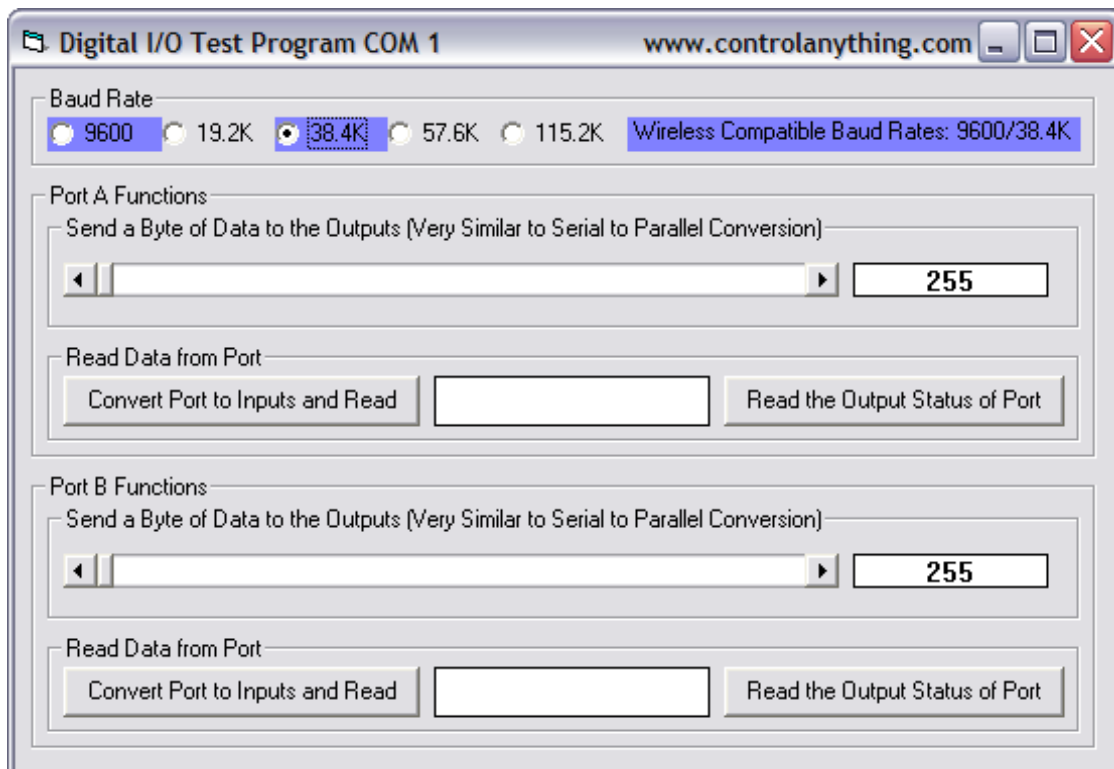
Port Byte Functions are demonstrated using the "Port Byte Functions" program installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

- Writing Bytes of Data Directly to Port A and B.
- Reading the Output Status of Port A and B.
- Reading Input Status on Port A and B.

This program provides a simple GUI that allows you test the port byte functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.



I/O Data Ports: Port Byte Commands

The sample code shown at right comes from the Port Byte Functions program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Keep in mind, Ultra Series controllers are 2-way native devices, and will report back ASCII character code 85 to confirm the operation has completed. All source code is written in Visual Basic 6.

The following commands will set the data port pins to output mode. Make sure these lines are NEVER connected directly to ground or +5V when using these commands.

The Variable "Param" Shown below has a valid range of 0-255.

254, 1, Param Write an Output Byte to Port A
254, 2, Param Write an Output Byte to Port B

The following commands convert the IO Data port to an input and reads a data byte from the data port.

254, 3* Read Port A as Input Byte, Returns 0-255
254, 5* Read Port B as Input Byte, Returns 0-255

The following commands read the output status of an entire data port. The IO lines are NOT converted to inputs, only the current output status is returned.

254, 4* Read Port A Output Byte, Returns 0-255
254, 6* Read Port B Output Byte, Returns 0-255

*This command reports a numeric value back to the computer from 0-255 instead of sending an 85 back to the computer.

Writing a Data byte to Port A

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(1)       'Put Byte A Command  
MSComm1.Output = Chr$(Byte)    'Byte to Write on Port A  
GetI
```

Writing a Data byte to Port B

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(2)       'Put Byte B Command  
MSComm1.Output = Chr$(Byte)    'Byte to Write on Port B  
GetI
```

Read Port A Inputs

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(3)       'Read Port A Inputs  
GetI
```

Read Port B Inputs

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(5)       'Read Port B Inputs  
GetI
```

Read Port A Output Status

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(4)       'Read Port A Output Status  
GetI
```

Read Port B Output Status

```
MSComm1.Output = Chr$(254)     'Enter Command Mode  
MSComm1.Output = Chr$(6)       'Read Port B Output Status  
GetI
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()  
    T = 0  
    Do  
        T = T + 1  
        If T > 10000 then Exit Function  
        DoEvents  
    Loop Until MSComm1.InBufferCount > 0  
    GetI = Asc(MSComm1.Input)  
End Function
```

Analog to Digital Conversion

Analog to Digital Conversion is one of the most powerful features offered by the Ultra series relay controllers. To this point, we have discussed CMOS/TTL logic signals, which are perfect for reading inputs that are 0 or 5VDC. But what about sensors that produce a voltage output?

Analog to digital conversion allows you to read analog sensors. An A/D converter is used to convert an analog voltage to a numeric value. An 8-Bit A/D converter will convert a voltage input of 0-5VDC into a numeric value from 0 to 255. A 10-Bit A/D converter will convert a voltage input from 0-5VDC into a numeric value from 0 to 1023, offering much better resolution.

Volts	8-Bit Value	10-Bit Value
0.00	0	0
0.25	12	51
0.50	25	102
0.75	38	153
1.00	51	204
1.25	63	255
1.50	76	306
1.75	89	358
2.00	102	409
2.25	114	460
2.50	127	511
2.75	140	562
3.00	153	613
3.25	165	664
3.50	178	716
3.75	191	767
4.00	204	818
4.25	216	869
4.50	229	920
4.75	242	971
5.00	255	1023

The Ultra series relay controllers have 8 built in A/D converters. You can take an 8-bit or a 10-bit reading on each channel. An 8-bit reading is very fast. A 10-bit reading takes about twice as long to execute, only because there is more data that must be communicated to your computer. The eight A/D channels are shared with I/O lines of Port A. It is not possible to mix I/O functions with A/D functions on Port A. Port A should be used exclusively for digital I/O or A/D, but not both at the same time. Reading any A/D channel forces all 8 I/O lines on Port A to become analog inputs.

Ultra Series Controllers allow you to read a single A/D channel as an 8-Bit or 10-Bit value. All you have to do is issue the appropriate A/D command. Additionally, you can request the status of all 8 analog inputs as 8-Bit or 10-Bit Values. This function is perfect for high-speed multi-channel A/D conversion, capable of delivering over 500 samples per second over a wireless connection.

Use Port A for A/D Features

Port A Can be used for A/D or I/O, but it is not possible to mix functions on this port. Reading any A/D channel will automatically configure all inputs on Port A to Analog inputs. Using any I/O function will automatically configure all data port lines on Port A for I/O functions.

A/D Header for Multi-Channel Commands Option:

Using the Ultra Series configuration utility, there is a check box labeled "Send Header with A/D Multi Channel". This option only applies to multi-channel A/D requests. Under normal circumstances, you will probably want to leave this option enabled. Turning this option off can give you significantly better performance, but it becomes difficult to track incoming data.

When this option is enabled, a header byte of 254 is used to signify the beginning of a series of data bytes. This header is used to help synchronize your software to incoming data. Here is an example data string that shows what is sent with these different options:

Data Sent from Controller	With Header Option Enabled	With Header Option Disabled
1st Byte Sent	254	A/D Channel 1
2nd Byte Sent	A/D Channel 1	A/D Channel 2
3rd Byte Sent	A/D Channel 2	A/D Channel 3
4th Byte Sent	A/D Channel 3	A/D Channel 4
5th Byte Sent	A/D Channel 4	A/D Channel 5
6th Byte Sent	A/D Channel 5	A/D Channel 6
7th Byte Sent	A/D Channel 6	A/D Channel 7
8th Byte Sent	A/D Channel 7	A/D Channel 8
9th Byte Sent	A/D Channel 8	none

Sending 10-Bit Data via 8-Bit Serial Communications

The reason for the decrease in performance for 10-bit A/D functions versus 8-bit A/D functions is simple to explain. Serial communications is only capable of transferring 8-bits of data at one time. So communicating a 10-bit value requires a second byte of data. Unfortunately, the second byte of data is only needed to communicate 2 bits of data. The other 6 bits of data are wasted. In theory, it should be possible to communicate 9-16 bit A/D values with no further performance penalty.

Since serial communications is restricted to transferring 8 bits of data, two byte must be sent. These two bytes must then be reconstructed by your program to provide a 10-bit value. Here is small section of code that is used to read 2 bytes of data from the controller using the GetI function. These two bytes of data are assigned the variable names Highbyte and Lowbyte. The 10-Bit A/D value is reconstructed using the formula on the last line of the example. As you can see, reconstruction is not very complicated.

```
Highbyte = GetI      'Get High Byte from Controller
Lowbyte = GetI      'Get Low Byte from Controller
ADValue = Lowbyte + (Highbyte * 256)      '10-Bit Value
```

This function is easily simplified to the following line of code:

```
ADValue = (GetI * 256) + GetI      '10-Bit Value
```

Analog to Digital Conversion Test Program

Analog to Digital Conversion Functions are demonstrated using the following programs installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software":

ADTest 8-Bit 1-Channel (Reads one channel at a time)
ADTest 8-Bit (Multi-channel version)
ADTest 10-Bit 1-Channel (Reads one channel at a time)
ADTest 10-Bit (Multi-channel version)

These application covers:

Reading A/D 8-Bit and 10-Bit values using the 1-Channel and Multi-Channel Functions. These programs provide a simple GUI that allows you test the A/D conversion functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.

The screenshot shows the 'A/D Test Program' window for 'COM 1' at 'www.controlanything.com'. The 'Baud Rate' is set to 38.4K. The '8-Bit A/D Test Program' section is active, with 'Requires Header byte' checked. The test results for channels 0 through 7 are as follows:

Channel	Value
0	108 / 255
1	99 / 255
2	98 / 255
3	108 / 255
4	99 / 255
5	98 / 255
6	108 / 255
7	108 / 255

The 'Samples per Second' is 325, and the 'Time of Last Error' is 49459.77. A text box explains: 'Decrease the TimeOut Value to Increase Samples per Second. We get excellent results at 700 (Over 500 Samples per Second on a Wireless Connection), you may be able to get better results with a lower value if you have a fast system! Setting this value too low will increase Communication Errors. The time of the last communication error is shown next to Samples per Second. Set the slider below to the lowest possible value that causes as few communication errors as possible. Compile this program to increase speed.' The 'TimeOut' slider is set to 8775. Performance tips at the bottom suggest increasing the baud rate and decreasing the character delay (CDEL) to the lowest possible value, and setting the TDS value to 1.

Slightly tweaked for performance, the screen shot above was taken using a wireless connection to the Ultra series relay controller, easily achieving 325 samples per second. Removing the "Header Byte" option allows over 500 samples per second, but it become a little difficult for this simple program to track which data byte goes with each channel.

Analog to Digital Conversion Example Usage

The sample code shown at right comes from A/D conversion programs located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Since these commands report data back to the computer, they do NOT send ASCII character code 85 back to the computer. All source code is written in Visual Basic 6.

254, 9, 0	8-Bit Read Channel 0
254, 9, 1	8-Bit Read Channel 1
254, 9, 2	8-Bit Read Channel 2
254, 9, 3	8-Bit Read Channel 3
254, 9, 4	8-Bit Read Channel 4
254, 9, 5	8-Bit Read Channel 5
254, 9, 6	8-Bit Read Channel 6
254, 9, 7	8-Bit Read Channel 7
254, 9, 8	10-Bit Read Channel 0
254, 9, 9	10-Bit Read Channel 1
254, 9, 10	10-Bit Read Channel 2
254, 9, 11	10-Bit Read Channel 3
254, 9, 12	10-Bit Read Channel 4
254, 9, 13	10-Bit Read Channel 5
254, 9, 14	10-Bit Read Channel 6
254, 9, 15	10-Bit Read Channel 7
254, 9, 16	8-Bit Multi-Channel Read
	This command reports back an optional header byte of 254, followed by 8 data bytes.
254, 9, 24	10-Bit Multi-Channel Read
	This command reports back an optional header byte of 254, followed by 6 data bytes.

Reading a Single 8-Bit A/D Channel

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(9) 'AD Commands
MSComm1.Output = Chr$(0) 'Request 8-Bit A/D Channel 0
GetI 'GetI Contains the 8-Bit A/D Value

MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(9) 'AD Commands
MSComm1.Output = Chr$(5) 'Request 8-Bit A/D Channel 5
GetI 'GetI Contains the 8-Bit A/D Value
```

Reading a Single 10-Bit A/D Channel

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(9) 'AD Commands
MSComm1.Output = Chr$(8) 'Request 10-Bit A/D Channel 0
Highbyte = GetI 'Get High Byte from Controller
Lowbyte = GetI 'Get Low Byte from Controller
ADValue = Lowbyte + (Highbyte * 256) 'ADValue Contains 10-Bit Value
```

Reading 8-Channels of 8-Bit A/D

```
Main:
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(9) 'AD Commands
MSComm1.Output = Chr$(16) 'Request 8-Channels 8-Bit
If GetI <> 254 Then GoTo Main 'Wait for Header
For Channel = 0 To 7 'Count Through Returned Data
    Debug.Print Channel;GetI 'Get Data Byte from Controller
Next Channel
```

Reading 8-Channels of 10-Bit A/D

```
Main:
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(9) 'AD Commands
MSComm1.Output = Chr$(16) 'Request 8-Channels 8-Bit
If GetI <> 254 Then GoTo Main 'Wait for Header
For Channel = 0 To 7 'Count Through Returned Data
    Highbyte = GetI 'Get High Byte from Controller
    Lowbyte = GetI 'Get Low Byte from Controller
    ADValue = Lowbyte + (Highbyte * 256) 'ADValue Contains 10-Bit Value
    Debug.Print Channel;ADValue 'Display Data in Debug Window
Next Channel
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Introduction to the LCD Display

An optional 40x2 Character LCD Display can easily be attached to the Ultra Series relay controllers. Connecting the display is as easy as plugging in a cable. The connectors on the display and controller are keyed, it is not possible to connect the display cable backwards. The backlight connections must be soldered to the bottom side of the Ultra Series controller, connections are made at the A and K markings on the back of the controller, and match the A and K markings on the right front side of the display.

The Ultra Series controllers offer two unique commands, not found on most display controllers, allowing you to software adjust 256 levels of backlight brightness and contrast.

These features are demonstrated when the controller is first powered up. The backlight will rise to full brightness. Next, the contrast will fade text into view.

We felt it important to offer software control of these parameters since LCDs are temperature sensitive. This will allow you to software compensate for a display that appears to be sluggish or slightly out contrast in colder temperatures.

By default, the Ultra Series controllers will limit the brightness level of the LED backlight to a value of 40. You can override this setting using the Ultra Configuration utility; however, the backlight protection resistor can get very hot and can eventually damage itself. We trust you will take reasonable care to ensure safe operation of the backlight should you decide to override the level 40 maximum brightness setting.

Here are some useful numbers that will show you maximum safe values. We would prefer that you do not stress the resistor for an extended period of time. But here are a few calculated values that will help determine how safe the brightness level you have chosen is, provided you keep in mind, the backlight resistor is capable of dissipating an absolute maximum of 3 Watts.

Software Controlled Brightness Levels

Brightness Level	Watts Dissipated Through the Backlight Resistor	Notes:
40	< 2 Watts	Normal Safe Levels for Extended Periods of Time
50	2 Watts	Ok to use for a long time, but generates more heat.
82	2.5 Watts	Backlight Resistor Gets Very Hot. Use with Care.
117	3 Watts	Backlight Resistor Gets Very Hot, Running it at Its Limit May Shorten the Life
>117	>3 Watts	Backlight Resistor Gets VERY VERY HOT!!! Recommended for a few seconds at a time only.

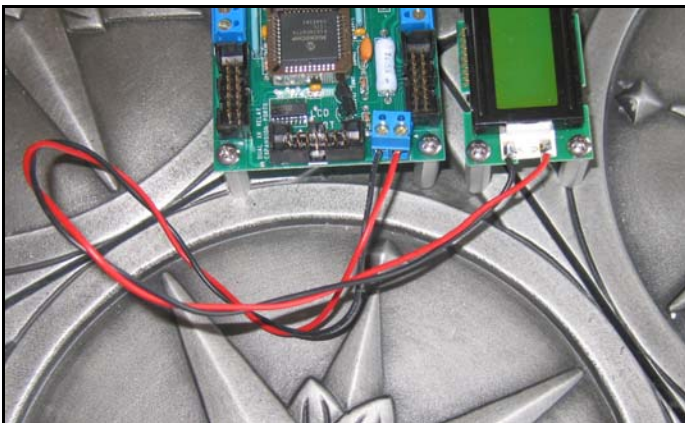
To override the maximum brightness level setting, using the Ultra Configuration utility and uncheck the "Limit LCD Brightness to Safe Levels" setting and store the new setting into the controller. The full brightness scale of 0-255 will then work. NOTE: At full brightness, the LCD backlight will never be damaged.

The Purpose of the Character LCD is primarily for you to use in your own messaging applications. The LCD displays a splash screen on startup, displaying company information, model number, and firmware version. This message can be hidden using the Ultra Configuration utility.

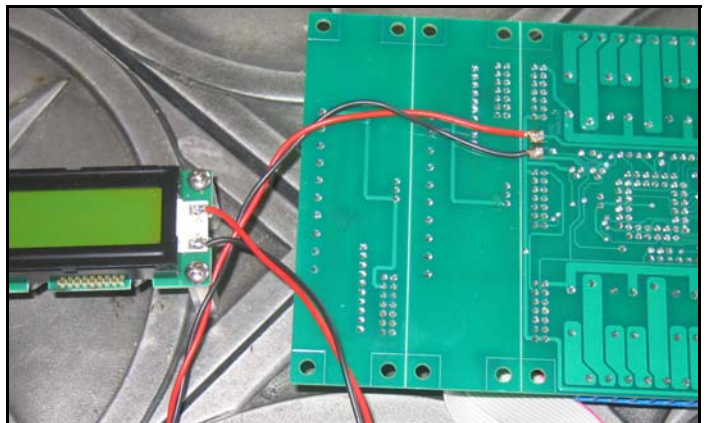
The next screen that appears after the splash screen is the communications configuration screen. This screen is very useful for displaying the communications rate and the E3C or EWC device numbers. This screen cannot be hidden.

After these two screens have finished, you can send commands to the display to display anything you want at any time. The LCD screen can be a very useful messaging tool for displaying errors or important messages to users in an unlimited number of remote locations.

Connecting the LED Backlight



The WIOADXr controller connects to the LED backlight of an LCD display as shown in the photo above.



The WIOADR16x connects to an LED backlight of an LCD display as shown above. Some soldering is required to make this connection.

NOTICE: The Character LCD Display Data Cable Connection is Self Explanatory, there is no way to connect it backwards, the cable and connectors are keyed to prevent misconnection.

LCD Test Program

LCD Display Functions are demonstrated using the "LCD Test Program" installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

- Sending Text to the display
- Software LCD Backlight Brightness and Contrast Adjustment
- Issuing Hardware Commands Directly to the Display

This program provides a simple GUI that allows you test the LCD functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.



LCD Command Examples

The sample code shown at right comes from A/D conversion programs located in the Examples.ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Since these commands report data back to the computer, they do NOT send ASCII character code 85 back to the computer. All source code is written in Visual Basic 6.

Commands 1 and 2 below are used

254, 8, 0	Initialize LCD Display
254, 8, 1,Data	LCD Hardware Data
254, 8, 2,CMD	LCD Hardware Command
CMD = 1	Clear Screen
CMD = 2	Home Cursor
CMD = 8	Blank LCD Display
CMD = 12	Unblank LCD Display
CMD = 14	Set Cursor to “_” Character
CMD = 15	Set Cursor to Blinking Block
CMD = 16	Move Cursor 1 Character to the Left
CMD = 20	Move Cursor 1 Character to the Right
CMD = 24	Scroll Entire Display 1 Character Left
CMD = 28	Scroll Entire Display 1 Character Right
CMD = 128-255	Set Cursor Position
254, 8, 3	Place Cursor on Line One
254, 8, 4	Place Cursor on Line Two
254, 8, 5, Brightness	LCD LED Backlight Brightness
254, 8, 6, Contrast	LCD Contrast Adjustment
254, 8, 7, "Text",255	LCD Text Command

Initializing the LCD Display

This command is used to “Reboot” the display.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(2) 'LCD Hardware Commands
MSComm1.Output = Chr$(0) 'Initialize LCD Command
GetI 'Wait for Command to Finish
```

Sending Hardware Data

Used to communicate data to the data registers of the display. This command is for advanced users who are familiar with the architecture of character LCD displays.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(1) 'LCD Hardware Data
MSComm1.Output = Chr$(DAT) 'DAT Contains Hardware Data for the Display
GetI 'Wait for Command to Finish
```

Sending Hardware Commands

This command is used to issue a command built directly into the character LCD hardware. The display has many commands built in. This is the most frequently used function for controlling all aspects of the display other than text. See the commands at left valid values of CMD shown in the program code below.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(2) 'LCD Hardware Commands
MSComm1.Output = Chr$(CMD) 'Send a Hardware Command
GetI 'Wait for Command to Finish
```

Set Cursor to Line 1

This macro command is used to home the cursor on the far left side of the display on line 1.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(3) 'Set LCD Cursor to Line 1
GetI 'Wait for Command to Finish
```

Set Cursor to Line 2

This macro command is used to home the cursor on the far left side of the display on line 2.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(4) 'Set LCD Cursor to Line 2
GetI 'Wait for Command to Finish
```

Software Controlled Brightness

This command is used to set the brightness of the LED backlight on the character LCD display. While valid values of 0-255 are allowed, the internal limit is set to 40 by default, which can be overridden. Please see notes on this topic in the LCD Introduction section of this manual.

This command can be used to produce some nice fading effects.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(5) 'Set LED Backlight Brightness Command
MSComm1.Output = Chr$(40) 'Set Backlight Brightness Level to 40
GetI 'Wait for Command to Finish
```

Software Controlled Contrast

This command is used to set the LCD Bias (contrast), and is typically set to a high value, around 230-250. This command can be used to produce some nice fading effects.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(6) 'Set LCD Contrast Command
MSComm1.Output = Chr$(250) 'Set Contrast Level to 250
GetI 'Wait for Command to Finish
```

Sending Text to the LCD Display

This command is used to send text to the LCD screen. You should use other commands to position the cursor prior to using this command. You can send a maximum of 60 characters to the LCD screen at one time. This command MUST be terminated with a 255.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(8) 'LCD Commands
MSComm1.Output = Chr$(7) 'LCD Text Command
MSComm1.Output = "Text" 'Send the word "Text"
MSComm1.Output = Chr$(255) 'Terminate Command
GetI 'Wait for Command to Finish
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Introduction to Relay Control

The Ultra Series relay controllers have a very powerful command set capable of controlling up to 512 relays individually or simultaneously. While most Ultra controllers have only 16 relays built into the controller, they also include two XR Relay Expansion Ports, which allow you to add banks of relays to the controller. So later on down the road, if you need a few more relays, a relay bank is easily plugged into the XR port and you are ready to start controlling more relays. XR Relay banks are available with all different kinds of relays, so if you need to switch high current relays along with signal relays, XR relay banks will provide the perfect solution, allowing you build your own relay controller to suite your own particular needs.

The entire architecture for the relay command set is based on Banks. It is absolutely critical to fully understand how banks work BEFORE attempting to use the Ultra Series relay controllers. A bank is simply a group of eight relays. A WIOADR16x controller has two banks of 8 relays, and is capable of controlling a total of 64 banks (2 banks built in plus 62 external banks).

Nearly all commands you send to the Ultra Series controller require that you specify which bank the command will be directed to. You can direct commands to relay banks 1 through 64. So any time you see the variable "Bank" in the upcoming pages, you should understand that a numeric value of 1 to 64 will be required. Most users will only use the on-board relay banks. However, if you need to add relays to your controller, you can save time and money by taking advantage of the built-in XR relay expansion ports.

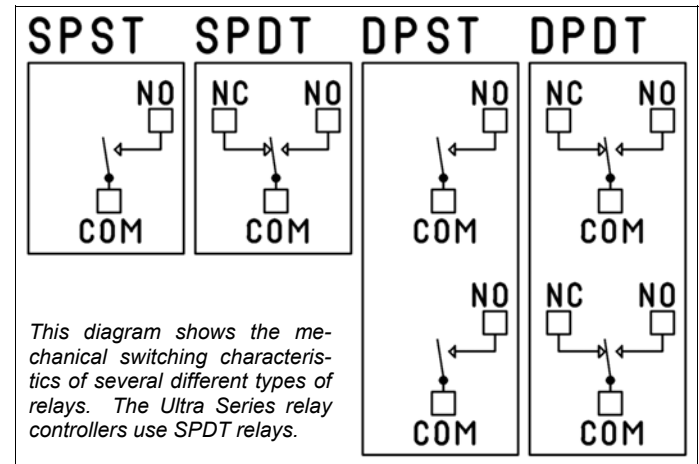
Relays on the Ultra Series controllers are "Latching", meaning that you send a command to turn on a relay. You send a different command to turn it off. The relay can stay on indefinitely, without harm to the relay or controller.

A relay is best described as a switch than can be actuated by an electronic circuit. A relay does NOT provide a voltage output. It only switches whatever voltage you apply to the relay. A relay controller, such as the Ultra Series controllers, allow computer controlled switching, either by wireless or RS-232 communications.

Relays come in many different configurations. The WIOADR16x use SPDT relay. Below is a list of common relay configurations and usages.

Relay Connections

SPDT relays have three connections: Normally Open, Common, and Normally Closed, often labeled NO, COM, and NC. When turning on/off a light bulb, power is usually connected to the Common. When the relay is off, power will go into the Common and directly out the Normally Closed connection on the relay. When the relay is turned on, the Common disconnects from the Normally Closed and reconnects to the Normally Open, cutting off power from the Normally Close lead and powering the Normally Open lead. The Common connection is the part of the relay that mechanically moves inside the relay.



How Long does a Mechanical Relay Last?

It all depends on what you are switching and weather the type of relay you have chosen appropriately matches the application. If you have a 10 Amp motor and you plan to control it with a 10 Amp relay, don't expect the relay to last too long. The reason being, a motor is an inductive load. Every time the motor is turned on, a surge current, upwards of 30+ amps in most cases, can be drawn through the relay, causing an incredible arcing on the contacts. This eventually fuses the contacts closed or permanently shorts them out. The proper choice of relay would be a 30+ amp relay, taking into account the surge current. If however, you are using a 10 Amp relay to control a 10 Amp incandescent lamp, the relay will last it's fully rated life span, which can be over 10 million cycles. Incandescent lamps fall into the category of being a resistive load, which means they do not require a violent surge current to activate the lamp. Today's mechanical relays have been known to outlast a similarly rated solid state relay.

Common Relay Configurations

Configuration	Description	Meaning	Usage
SPST	Single Pole Single Throw	Two Wires can be Connected Together	This type of relay is useful for simply turning a light or a motor on or off.
SPDT	Single Pole Double Throw	A wire can be connected to one of two wires, and is always connected to one of the wires.	This type of relay is very useful for applying power to one of two devices (one of the devices will always have power), or to simply turn a device on or off. These kinds of relays can be wired such that a device is always on, when the relay turns off the device will turn on. These kinds of relays can also be wired so that a device turns on only when the relay turns on.
DPST	Double Pole Single Throw	Two SPST switches inside one relay.	Same as SPST Above x 2, most commonly used in high-current switching or in telecommunications switching applications.
DPDT	Double Pole Double Throw	Two DPDT switched inside one relay.	Same as SPDT Above x 2, most commonly used for audio, signal, or telecommunications switching.

Relay Test Program Example Software

Relay Control Functions are demonstrated using the "Relay Commands" test program installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

Controlling Individual Relays

Controlling Banks of Relays

Controlling up to 512 Relays at One Time

Reading Relay Status and Much More

This program provides a simple GUI that allows you test the relay control functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.

Relay Test Program COM 1 www.controlanything.com

Baud Rate
 9600 19.2K 38.4K 57.6K 115.2K Wireless Compatible Baud Rates: 9600/38.4K

Select a Relay Bank to Communicate To
1

Controlling Individual Relays Cycle Test

Relay 1	Relay 2	Relay 3	Relay 4	Relay 5	Relay 6	Relay 7	Relay 8
Relay Off	Relay Off	Relay Off	Relay Off	Relay Off	Relay Off	Relay Off	Relay Off
Relay On	Relay On	Relay On	Relay On	Relay On	Relay On	Relay On	Relay On

Controlling Multiple Relays

Turn Off All Relays in the Selected Bank	Invert Relays	Set All Relays in Banks to New Value	1
Turn On All Relays in the Selected Bank	Reverse Relay Order		

Controlling Multiple Relay Banks

Disable All Banks	Enable All Banks	Memory Map Editor	Set Relays to Power-Up Default State
-------------------	------------------	-------------------	--------------------------------------

Read the Status of a Selected Relay

1	Read One ->	-
	Read All ->	

Non-Volatile Storage Commands

Store Current Relay Settings (All 64 Banks) as Power-Up Default	
Read the Power-Up Default Status of a Relay Bank (Selected Above)	-

Break Before Make on Selected Relay Bank

1

Make Before Break on Selected Relay Bank

1

Relay Bank Memory Map Editor

The Memory Map Editor can be accessed from the "Relay Test Program". The memory map editor allows you to read, modify, and write relay status information into each relay bank. The 63 shown in the photo below refers to the relay pattern in relay bank 1. The 255 refers to the relay pattern in relay bank 2. The subsequent relay banks correspond to relay banks connected to the main controller via the XR Relay Expansion ports. Using the editor, you can directly modify relay status memory. Once you have completed memory modification, you can refresh all 512 relays at one time. The mechanics of how this is done will be discussed in the following pages. But for now, you should experiment with the different buttons in the Memory Map Editor to see how these commands work.

Control 512 Relays Simultaneously via the XR Relay Expansion Ports!

Address	Value 1	Value 2	Value 3
63:11111100	0:00000000	0:00000000	0:00000000
255:11111111	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000
0:00000000	0:00000000	0:00000000	0:00000000

Select a Relay Bank to Modify. This will Allow you to Easily Update Contents of Memory, then Refresh the Relays with a Simple Command

1 Select a Memory Bank to Modify (One Memory Bank per Relay Bank)

63:11111100 Alter the Memory with New Relay Value

Refresh All Relays with Memory Map Shown Above Store Relay Pattern in Controller Memory Refresh Map

Relay Control Commands Example Usage

The sample code shown at right are demonstrated in the Relay Commands test program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Some of these commands report data back to the computer, they do NOT send ASCII character code 85 back to the computer. All source code is written in Visual Basic 6.

The Bank variable shown below refers to the relay bank to control. Banks values of 1 and 2 refer to the relays built into the WIOADR16x controller. Bank values of 3-64 refer to relay banks that are attached externally to the WIOADR16x controller via the two XR Expansion Relay ports.

```
254, 10,Bank,0 Turn Off Relay 0
254, 10,Bank,1 Turn Off Relay 1
254, 10,Bank,2 Turn Off Relay 2
254, 10,Bank,3 Turn Off Relay 3
254, 10,Bank,4 Turn Off Relay 4
254, 10,Bank,5 Turn Off Relay 5
254, 10,Bank,6 Turn Off Relay 6
254, 10,Bank,7 Turn Off Relay 7
254, 10,Bank,8 Turn On Relay 0
254, 10,Bank,9 Turn On Relay 1
254, 10,Bank,10 Turn On Relay 2
254, 10,Bank,11 Turn On Relay 3
254, 10,Bank,12 Turn On Relay 4
254, 10,Bank,13 Turn On Relay 5
254, 10,Bank,14 Turn On Relay 6
254, 10,Bank,15 Turn On Relay 7
254, 10,Bank,16 Send On/Off Status of Relay 0
254, 10,Bank,17 Send On/Off Status of Relay 1
254, 10,Bank,18 Send On/Off Status of Relay 2
254, 10,Bank,19 Send On/Off Status of Relay 3
254, 10,Bank,20 Send On/Off Status of Relay 4
254, 10,Bank,21 Send On/Off Status of Relay 5
254, 10,Bank,22 Send On/Off Status of Relay 6
254, 10,Bank,23 Send On/Off Status of Relay 7
254, 10,Bank,24 Send On/Off Status of All Relays
254, 10,Bank,25 Stores a Power-Up Default Relay Pattern
254, 10,Bank,26 Recalls a Power-Up Default Relay Pattern
254, 10,Bank,27 Hardware Disable All Relay Banks
Bank Variable Has No Effect
254, 10,Bank,28 Hardware Enable All Relay Banks
Bank Variable Has No Effect
```

Activating Relays at a Specific Time

Want to trigger a relay or some other event at a specific time? This code samples uses the VB Time command to activate an event at a specific time.

```
Do
    'Wait for an event to happen
    DoEvents 'Allow System to Service Other Tasks
    If Time = "10:30:00 AM" Then
        'At 10:30AM
        Debug.Print "EVENT"
        'Execute an Event
    Else
        'Otherwise
        Debug.Print Time
        'Display Current Time
    End If
Loop
    'Continue Waiting
```

Activating Relays on a Specific Date

Want to trigger a relay or some other event on a specific date? This code samples uses the VB Date command to activate an event on specific day.

```
Do
    'Wait for an event to happen
    DoEvents 'Allow System to Service Other Tasks
    If Date = "4/27/2007" Then
        'On 4/27/2007
        Debug.Print "EVENT"
        'Execute an Event
    Else
        'Otherwise
        Debug.Print Date
        'Display Current Date
    End If
Loop
    'Continue Waiting
```

Turning On a Relay Bank 1 Relay 0

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(8) 'Turn On Relay 0
GetI 'GetI Returns 85
```

Turning Off a Relay Bank 1 Relay 0

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(0) 'Turn Off Relay 0
GetI 'GetI Returns 85
```

Turning On a XR Relay Bank 3 Relay 0

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(3) 'Select Relay Bank
MSComml.Output = Chr$(8) 'Turn On Relay 0
GetI 'GetI Returns 85
```

Report Status of Relay Bank 1 Relay 5

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(21) 'Get Relay 5 Status Command
GetI 'GetI Returns 0 or 1 (off or on)
```

Report Status of All Relays on Bank 2

GetI returns 0-255, convert to binary to view the relay pattern.

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(2) 'Select Relay Bank
MSComml.Output = Chr$(24) 'Report Status Command
GetI 'GetI Returns Status of Bank (0-255)
```

Store a Power-Up Default Relay Pattern

The current on/off relay pattern for the selected relay bank will be stored as the power-up default relay pattern. This command only applies to one relay bank. The next time the controller is powered, the relays will automatically update to the stored relay pattern.

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(25) 'Store Relay Pattern
GetI 'GetI Returns 85
```

Recall a Power-Up Default Relay Pattern

The current on/off relay pattern for the selected relay bank will be retrieved from memory and sent back to the user. This command only applies to one relay bank. The returned value of 0-255 can be converted to binary to view the on/off relay pattern.

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(26) 'Recall Relay Pattern
GetI 'GetI Returns 0-255
```

Hardware Disable All Relay Banks

This command kills all relays on all relay banks. This command does not affect the on/off pattern in memory. The Bank variable has no affect on this command, but a variable of 1-64 is required.

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(27) 'Hardware Disable All Relay Banks
GetI 'GetI Returns 85
```

Hardware Enable All Relay Banks

This command restores functionality of all relays on all relay banks. This command does not affect the on/off pattern in memory. Relays that were originally in the on state will be returned to their on state. The Bank variable has no affect on this command, but a variable of 1-64 is required.

```
MSComml.Output = Chr$(254) 'Command Mode
MSComml.Output = Chr$(10) 'Relay Commands
MSComml.Output = Chr$(1) 'Select Relay Bank
MSComml.Output = Chr$(28) 'Hardware Enable All Relay Banks
GetI 'GetI Returns 85
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComml.InBufferCount > 0
    GetI = Asc(MSComml.Input)
End Function
```

Relay Control Commands Example Usage

The sample code shown at right are demonstrated in the Relay Commands test program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Some of these commands report data back to the computer, they do NOT send ASCII character code 85 back to the computer. All source code is written in Visual Basic 6.

The Bank variable shown below refers to the relay bank to control. Banks values of 1 and 2 refer to the relays built into the WIOADR16x controller. Bank values of 3-64 refer to relay banks that are attached externally to the WIOADR16x controller via the two XR Expansion Relay ports.

254, 10,Bank,29	Turn Off All Relays in Relay Bank
254, 10,Bank,30	Turn On All Relays in Relay Bank
254, 10,Bank,31	Invert All Relays in Selected Relay Bank
254, 10,Bank,32	Reverse Current Relay Order
254, 10,Bank,33	Test 2-Way Communications Bank Variable has No Affect Always Reports 85 Back to the User
254, 10,Bank,40,Param	Set Status of All Relays in a Bank

Turn Off All Relays in Bank 1

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(29) 'Turn Of All Relays
GetI 'GetI Returns 85
```

Turn On All Relays in Bank 1

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(30) 'Turn On All Relays
GetI 'GetI Returns 85
```

Invert All Relays in a Selected Relay Bank

This command is used to invert the relay pattern in a selected relay bank. Relays that are currently on will turn off and relays that are off will turn on.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(31) 'Invert Relays in a Bank
GetI 'GetI Returns 85
```

Reverse All Relays in a Selected Relay Bank

This command reverses the current relay pattern in a selected relay bank. This function only alters the current memory.

Relay 1 status is moved to relay 8.

Relay 2 status is moved to relay 7.

Relay 3 status is moved to relay 6.

Relay 4 status is moved to relay 5.

Relay 5 status is moved to relay 4.

Relay 6 status is moved to relay 3.

Relay 7 status is moved to relay 2.

Relay 8 status is moved to relay 1.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(32) 'Reverse Relays in a Bank
GetI 'GetI Returns 85
```

Test 2-Way Communication

This command simply reports back 85 to the user. This command was included to help maintain compatibility with the R8xPRO command set. The Bank variable has no effect on this command, but a valid value of 1-64 must be used.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(33) 'Test 2-Way Communications
GetI 'GetI Returns 85
```

Set Status of All Relays in a Relay Bank

This command is used to set the status off relays in a selected relay bank. This command requires a PARAM value of 0-255. A value of 0 turns off all relays in a selected bank, a value of 255 turns on all relays in a selected bank. Every number in between writes the equivalent binary pattern directly to the selected relay bank.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(40) 'Set Status of All Relays
MSComm1.Output = Chr$(85) 'Relay Pattern 01010101
GetI 'GetI Returns 85
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Relay Control Commands Example Usage

The sample code shown at right are demonstrated in the Relay Commands test program located in the Examples .ZIP archive. The commands shown below refer to the ASCII Character Codes (binary data) that must be sent to issue the command. Some of these commands report data back to the computer, they do NOT send ASCII character code 85 back to the computer. All source code is written in Visual Basic 6.

The Bank variable shown below refers to the relay bank to control. Banks values of 1 and 2 refer to the relays built into the WIOADR16x controller. Bank values of 3-64 refer to relay banks that are attached externally to the WIOADR16x controller via the two XR Expansion Relay ports.

254, 10,Bank,41	Manually Refresh All Relay Banks LCD Display Will Not be Refreshed Bank Variable Has No Effect
254, 10,Bank,42	This Command is Used to Read the Status Information from a Relay Bank Array
254, 10,Bank,43, Param	This Command is Used to Write Relay Status Information into the Relay Bank without Refreshing the Relays
254, 10,Bank,44,Param	Safe Break Before Make, one of eight Activates One Relay at a Time PARAM = 0 to 7
254, 10,Bank,45,Param	Safe Make Before Break, one of eight Deactivates One Relay at a Time PARAM = 0 to 7
254, 10,Bank,46	Sets Relays to Stored Power-Up Default State
254, 11	Refresh All Relay Banks LCD Display Will Display Relay Status Info if configured to Shows Relay Bank Changes

Manually Refresh All Relay Banks

This command will copy relay memory in the controller to all 64 relay banks, causing all 512 relays to update at one time. This command does NOT display any relay status information on the LCD. The relay bank variable has no effect, all relays will be refreshed, but a valid bank value of 1-64 must be used.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(41) 'Refresh All Relays Command
GetI 'GetI Returns 85
```

Read Relay Bank Memory Location

This command sends the current relay status for a selected memory bank location. This command reports back a value of 0-255. Bank values of 1 to 64 are required for this command.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(42) 'Read Relay Memory Bank
GetI 'GetI Returns 0-255
```

Write Relay Bank Memory Location

This command changes relay status memory without refreshing the relays. This command requires a parameter of 0-255. Bank values of 1 to 64 are required for this command.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(43) 'Write Relay Memory Bank
MSComm1.Output = Chr$(170) 'Write Pattern 10101010 to Memory Bank
GetI 'GetI Returns 85
```

Break Before Make

This command turns off all relays, then turns on one of 8 relays. This command is used to safely disconnect an old circuit before reconnecting a new circuit. This command only works with one bank of 8 relays on the selected relay bank, and requires a Param value of 0-7, indicating which new relay should be activated.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(44) 'Break Before Make
MSComm1.Output = Chr$(3) 'Turn Off all relays, Turns on relay 4
GetI 'GetI Returns 85
```

Make Before Break

This command turns on all relays, then turns off one of 8 relays. This command is used to safely connect a new circuit before disconnecting an old circuit. This command only works with one bank of 8 relays on the selected relay bank, and requires a Param value of 0-7, indicating which new relay should be deactivated.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(45) 'Break Before Make
MSComm1.Output = Chr$(3) 'Turn On all relays, Turns off relay 4
GetI 'GetI Returns 85
```

Make Before Break

This command sets all 512 relays to the stored power-up default state. A Bank value of 1-64 is required, but not used by this command.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(10) 'Relay Commands
MSComm1.Output = Chr$(1) 'Select Relay Bank
MSComm1.Output = Chr$(46) 'Break Before Make
GetI 'GetI Returns 85
```

Refresh Relays with LCD

This command refreshes all relays, displaying relay status information on the LCD screen if configured to display Relay Status.

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(11) 'Refresh Relays with LCD
GetI 'GetI Returns 85
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Automatic and Manual Relay Refresh:

In the examples shown in the last two pages, there are many commands that can be used to change the status of the relays. By default, relays are automatically refreshed. You can turn off automatic refreshing, allowing greater simultaneous control of large groups of relays. Automatic refreshing is turned off using the Ultra Configuration Utility, or you can Modify the Controller Configuration Bits manually. When turned off, you can use all of the commands found on the last two pages to alter relay memory contents ONLY, relays will not change state. Relays will not be updated until you issue the Relay Refresh command found on this page.

254, 0, 5 10	'Turns Off Automatic Relay Refresh
254, 0, 5 26	'Turns On Automatic Relay Refresh

E3C Test Program

E3C RS-232 Networking Functions are demonstrated using the “E3C Test Program” installed in the Examples .ZIP archive, which is included with the “Ultra Series Configuration and Test Software”.

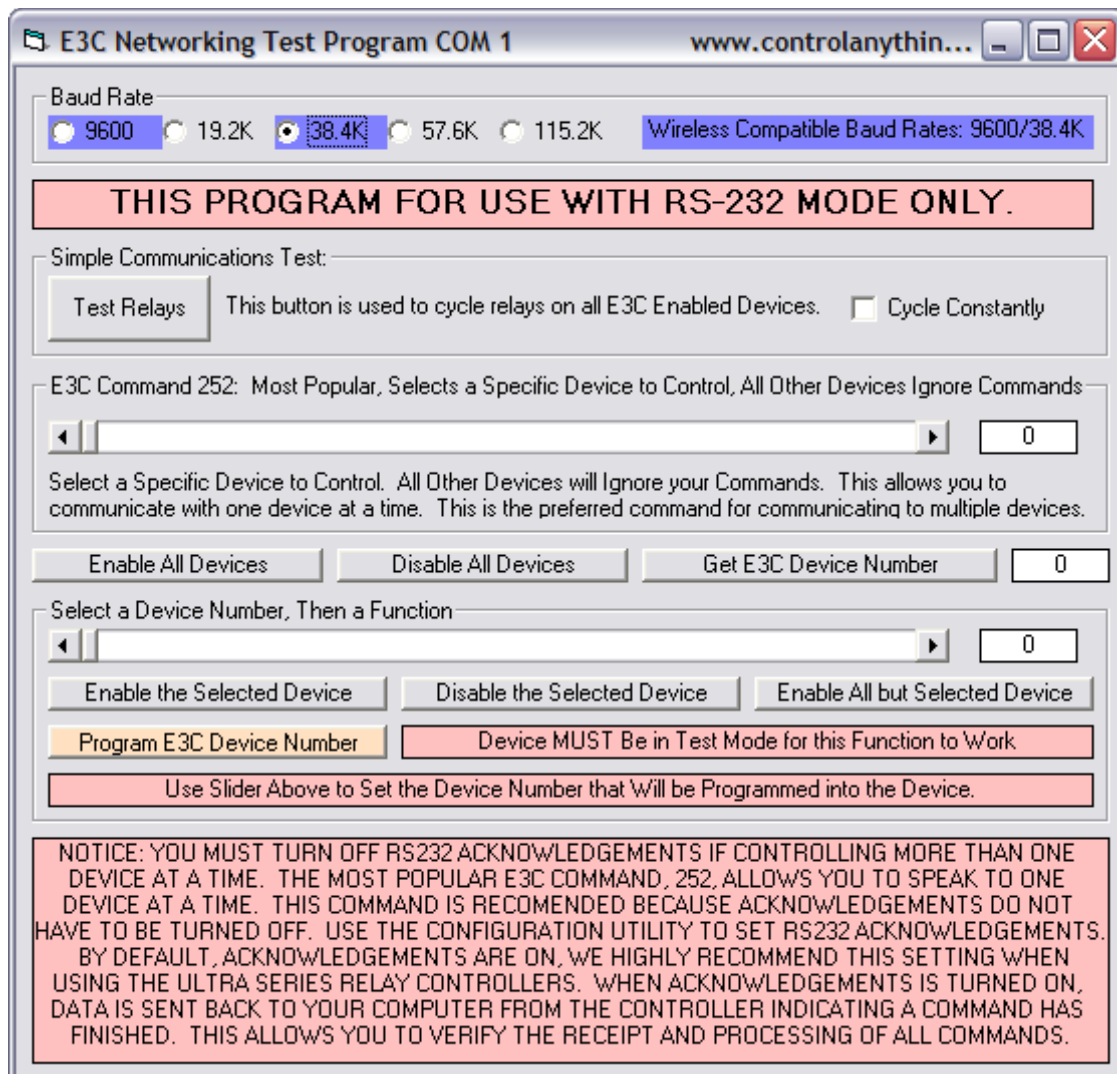
This application covers:

Programming E3C Device Numbers
Enabling and Disabling E3C devices

This program provides a simple GUI that allows you test the E3C functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.

THIS PROGRAM SHOULD ONLY BE USED WITH ULTRA CONTROLLERS THAT ARE DIRECTLY WIRED TO THE RS-232 PORT OF YOUR COMPUTER.



The E3C Command Set: Software Control of Multiple NCD Devices

The E3C command set allows you to control up to 256 NCD devices from a single serial port. It is OK to mix different types of devices, as long as the devices are E3C compliant. The Ultra Series relay controllers support the full set of E3C commands, plus a set of extended commands for storing and recalling the device number.

How does E3C Work?

First of all, each device must be assigned a device number from 0 to 255. The Ultra Series Controllers must be programmed with a device number, which is accomplished using the "Store Device Number" command shown below, or can be done from the Ultra Configuration Utility.

E3C stands for Enabled 3-Wire Communication. Put simply, when you first power up your computer and all the devices attached to the serial port, all devices will respond to your commands.

Using the E3C command set, you can specify which devices will listen and which devices will ignore your commands. Note that E3C commands are never ignored by any device, regardless of the commands you send to the controller.

The number to the left of each command indicates the ASCII character code that must be sent to issue the command. All commands must be preceded with ASCII character code 254 to place the device in command mode. See examples at right.

The E3C Command Set

248 Enable All Devices:

Tells all devices to respond to your commands.

249 Disable All Devices:

Tells all devices to ignore your commands.

250 Enable a Selected Device:

Tells a specific device to listen to your commands.

251 Disable Selected Device:

Tells a specific device to ignore your commands.

252 Enable Selected Device Only:

Tells a specific device to listen to your commands, all other devices will ignore your commands.

253 Disable a Selected Device Only:

Tells a specific device to ignore your commands, all others will listen.

Extended E3C Commands

The Ultra Series controllers support two additional E3C commands which should only be used when a single device is attached to your serial port. Extended commands will report back to the computer. The Ultra controller MUST BE IN TEST MODE for these commands to work.

255 Store Device Number:

Stores the device number into the controller. The device number takes effect immediately. The enabled/disabled status of the device is unchanged.

247 Recall Device Number:

Allows you to read the stored device number from the controller.

E3C Visual Basic Programming Examples

The E3C command set is easily used from any programming language that supports serial communication. The following Visual Basic 6 Example source code demonstrates subroutines that can be used to control which devices will listen and which devices will ignore your commands.

Most commands issued to the Ultra Series controllers are acknowledged by sending ASCII character code 85 back to the host computer (when acknowledgements are turned on). E3C commands are not acknowledged regardless of the reporting mode.

Sample Code: The E3C Command Set

```
Public Sub EnableAllDevices()  
'Enable All E3C Devices  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(248) 'E3C Enable All Device Command  
End Sub  
  
Public Sub DisableAllDevices()  
'Disable All E3C Devices  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(249) 'E3C Disable All Device Command  
End Sub  
  
Public Sub EnableSpecificDevice(Device)  
'Enable A Specific E3C Devices, Other Devices will be unchanged  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(250) 'E3C Disable Specific Device Command  
MSComm1.Output = Chr$(Device) 'Device Number that will be Disabled  
End Sub  
  
Public Sub DisableSpecificDevice(Device)  
'Disable A Specific E3C Devices, Other Devices will be unchanged  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(251) 'E3C Disable Specific Device Command  
MSComm1.Output = Chr$(Device) 'Device Number that will be Disabled  
End Sub  
  
Public Sub DisableAllDevicesExcept(Device)  
'Disable All E3C Devices Except (Device)  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(252) 'E3C Disable All Device Except Command  
MSComm1.Output = Chr$(Device) 'Device Number that will be Active  
End Sub  
  
Public Sub EnableAllDevicesExcept(Device)  
'Enable All E3C Devices Except (Device)  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(253) 'E3C Enable All Device Except Command  
MSComm1.Output = Chr$(Device) 'Device Number that will be Inactive  
End Sub
```

Sample Code: Extended E3C Commands

```
Public Sub StoreDeviceNumber(Device)  
'Store an E3C Device Number into the Controller  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(255) 'E3C Store Device Number Command  
MSComm1.Output = Chr$(Device) 'Device Number that will be Stored  
WaitForReply 'Wait for R16 to Acknowledge Command  
End Sub  
  
Public Function GetDeviceNumber()  
'Read the E3C Device Number from the Controller  
MSComm1.Output = Chr$(254) 'Enter Command Mode  
MSComm1.Output = Chr$(247) 'E3C Get Device Number Command  
Do  
    DoEvents 'Wait for Device to Reply  
Until MSComm1.InBufferCount > 0 'Allow Windows to MultiTask  
GetDeviceNumber = Asc(MSComm1.Input) 'If the Device Replies  
End Sub
```

Using a GUI to Program Device Numbers:

The Ultra Configuration Utility and the E3C Test Program Allow you to Store and Retrieve E3C Device Numbers using a Simple GUI on Any Windows Computer. We highly recommend using the tools we have provided to configure your controller. Ultra Series Controllers Write Protect Memory when NOT in test mode. For this reason, you MUST be in test mode to program E3C device numbers.

Introduction to Wireless Networking: Understanding Keys

Any wireless NCD device that is capable of receiving wireless remote commands has the ability to use "keys". A key is best described as a device number. A complete device number, or key, is made up of three numbers. Each number has a value of 0 to 255.

Our Ultra Series relay controllers, require that you place the remote device in Test Mode before keys are programmed into the remote device. Wireless devices are typically restricted in functionality while in configuration mode. Wireless devices must be in run mode to operate correctly. The Ultra Series Controllers use DIP switches to select modes of operation.

Keys Required Networking Method

When a remote device has the "Keys Required" or "Require Keys" option enabled, keys must be transmitted, along with each command to unlock the command.

Here is an example of how you would turn on a relay using three different devices that have the "Require Keys" option enabled.

The First device is programmed with KEYS 1,1,1
The Second device is programmed with KEYS 2,2,2
The Third device is programmed with KEYS 3,3,3

(Keep in mind, you can use any values for keys, each number must be a value from 0 to 255).

The following code turns on relay 1 on KEY 1,1,1

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(1) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(1) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(1) 'Device Number (Key 3) to Enable
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

The following code turns on relay 1 on KEY 2,2,2

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(2) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(2) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(2) 'Device Number (Key 3) to Enable
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

The following code turns on relay 1 on KEY 3,3,3

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(3) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(3) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(3) 'Device Number (Key 3) to Enable
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

This method is very easy to implement and offers EXCELLENT noise immunity. The drawback being that command execution time will be slightly slower.

This lockout system is used to prevent accidental programming of Keys into wireless devices that should not be programmed. All NCD wireless devices have some hardware method of locking out a device from accidental programming. Some devices are programmed wirelessly, while other devices may require a direct connection to the serial port. You will appreciate our lockout strategy more and more as your wireless network grows to include more devices. You will be able to safely program the device number keys into new controllers without ever being concerned with accidental reprogramming of existing devices.

EWC Networking Method

The EWC Networking Command set is explained in much greater detail on the next page. The following example shows how you would turn on a relay using three different devices using the EWC networking method.

The First device is programmed with KEYS 1,1,1
The Second device is programmed with KEYS 2,2,2
The Third device is programmed with KEYS 3,3,3
The "Require Keys" option is OFF on ALL remote devices.

Enable Device with KEY 1,1,1

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(252) 'Enable a Remote Device with These Keys:
MSComm1.Output = Chr$(1) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(1) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(1) 'Device Number (Key 3) to Enable
```

All subsequent commands will be directed to Key 1,1,1.

Turn On Relay 1

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

Enable Device with KEY 2,2,2

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(252) 'Enable a Remote Device with These Keys:
MSComm1.Output = Chr$(2) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(2) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(2) 'Device Number (Key 3) to Enable
```

All subsequent commands will be directed to Key 2,2,2.

Turn On Relay 1

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

Enable Device with KEY 3,3,3

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(252) 'Enable a Remote Device with These Keys:
MSComm1.Output = Chr$(3) 'Device Number (Key 1) to Enable
MSComm1.Output = Chr$(3) 'Device Number (Key 2) to Enable
MSComm1.Output = Chr$(3) 'Device Number (Key 3) to Enable
```

All subsequent commands will be directed to Key 3,3,3.

Turn On Relay 1

```
MSComm1.Output = Chr$(254) 'Enter Command Mode
MSComm1.Output = Chr$(10) 'Relay Control Commands
MSComm1.Output = Chr$(1) 'Control Relays on Bank 1
MSComm1.Output = Chr$(8) 'Turn on the First Relay.
```

YOU SHOULD ALWAYS CHOOSE A WIRELESS NETWORKING METHOD, NEVER MIX METHODS WITHIN RADIO RANGE OF EACH OTHER. SINCE KEYS CAN BE ANY VALUE, KEYS CAN TRIGGER RELAYS IF NETWORKING METHODS ARE MIXED.

EWC Wireless Networking Test Program

EWC Wireless Networking Functions are demonstrated using the "Wireless Networking" test program installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

Programming EWC Device Numbers (Keys)
Enabling and Disabling EWC devices

This program provides a simple GUI that allows you test the EWC functions of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.

Important Info about Wireless Networking:

Since NCD Wireless ES Series Devices offer 2-way communication, it is NOT possible to reliably communicate with more than one device at a time. For this reason, wireless networking commands should always target only one device, you should never set multiple wireless devices to the same device number.

E3C Networking Test Program COM 1 www.controlanything.com

Baud Rate
9600 19.2K 38.4K 57.6K 115.2K Wireless Compatible Baud Rates: 9600/38.4K

THIS PROGRAM FOR USE WITH WIRELESS MODE ONLY.

Simple Communications Test:
Test Relays This button is used to cycle relays on all E3C Enabled Devices. Cycle Constantly

Set Wireless Device Number
Use Sliders Below to Set the Wireless Device Number that Will be Programmed into the Device.

KEY 1
0 Enable the Selected Device Only the Newly Enabled Controller will Reply to this Command

KEY 2
0 Read Wireless Device Number from Controller (Read Keys) Execute Keychange Sequence

KEY 3
0 Store Wireless Device Number (Store Keys into Controller) Device MUST Be in Test Mode for this Function to Work

NOTICE: Because this is a 2-Way Device, Only One Device may be Spoken to at a Time while in Wireless Mode. You can control multiple NCD Devices at One Time when Device is Directly Wired to an RS232 Serial Port.

The EWC Command Set: Software Control of Multiple Wireless NCD Devices

Using the EWC Command Set, it is possible to individually control any number of remote wireless devices. EWC commands are used to select a specific device to communicate with. Each remote device is identified by a unique Key, which is a set of three numbers that you store into the remote device. Once these three keys have been stored, you can use EWC commands to select which device you would like to control.

NOTE: THE EWC COMMAND SET DOES NOT WORK ON REMOTE DEVICES THAT HAVE THE "KEYS REQUIRED" OPTION ENABLED. IN GENERAL, THE "KEYS REQUIRED" OPTION IS A LITTLE EASIER TO USE, BUT COMMUNICATIONS IS SLOWER.

The EWC command set is virtually identical to the E3C command set found on our directly wired RS-232 devices. There are only a few minor differences. Three keys are used to replace a single device number. The AirControl does not really know anything about EWC commands, it just transmits the command codes to all wireless devices within range. The remote device then determines if it should be enabled or disabled. Since the AirControl is used for communication of EWC commands, the AirControl will report back an 85 after processing, which is another notable difference from standard E3C commands. Since our wireless LR Series controllers are only one way, it is not possible to retrieve the keys or other settings from the remote device. So take note of your keys (device number) settings stored within your remote devices. Lastly, E3C command 253 was eliminated from the EWC command set due to its lack of use.

Networking Comparison

As you probably already know, it is possible to control an unlimited number of devices remotely. There are two ways of doing this, depending on your preference. Devices are identified by a device number, which consists of three numbers, each set to a value between 0 and 255. You will need to program the remote device with a device number when the remote device is in "Test" mode.

Require Keys Method

This is a very convenient protocol, very easy to implement that allows you to control one remote device at a time. Communications is slower, but this method offer superior 72-bit noise rejection. When the "Require Keys" options is enabled on a remote device, keys must be immediately transmitted after the 254 command mode header. Only the device with correct key will respond. This is one our favorite methods of remote control because of it's ease of implementation....just add the three key numbers after the 254 and before the actual command code. It really doesn't get any easier than to use the Required Keys Method.

EWC Networking

EWC cuts down on the number of data bytes sent out the serial port, which greatly increases communication speed. Noise rejection is limited to 42 bits using this method.

Regardless of the networking method you choose, we only recommend communicating with one remote device at a time. Never set remove devices to the same device number.

The EWC Command Set

252 Enable Selected Device Only:

Tells a specific device to listen to your commands, all other devices will ignore your commands. You will need to send 3 keys, identifying the device you would like to enable.

245 Read the Wireless Device Number (Keys)

This command reports back the EWC Device Number of the currently enabled device.

Proper Key Change Sequence for EWC Networks

Because we are dealing with wireless communications, the possibility does exist that command 252 will not be properly executed by all remote devices. In these cases, some device may still be enabled. For this reason, these two commands should always be used symbiotically to ensure you are directing commands to the appropriate device. The proper key change sequence is:

Start Sequence:

254, 252, Key1,Key2,Key3

254, 245, GetKey1,GetKey2,GetKey3

GetKey1 Must = Key1, If Not then Start Sequence

GetKey2 Must = Key2, If Not then Start Sequence

GetKey3 Must = Key3, If Not then Start Sequence

A full example of a proper key change sequence is shown in the source code for Wireless Networking, which uses these two commands to ensure commands are properly directed to the intended remote device.

Sample Code: The EWC Command Set

Enabling a Wireless Device for Communication

This command is used to direct all commands to a specific device number. All subsequent commands will go to this device only.

```
MSCMml.Output = Chr$(254) 'Enter Command Mode
MSCMml.Output = Chr$(252) 'EWC Disable All Device Except Command
MSCMml.Output = Chr$(Key1) 'Device Number (Key 1) to Enable
MSCMml.Output = Chr$(Key2) 'Device Number (Key 2) to Enable
MSCMml.Output = Chr$(Key3) 'Device Number (Key 3) to Enable
GetI 'Enabled Device will Report Back 85
```

Reading the Wireless Device Number

This command reports back the EWC wireless keys for the device that is currently enabled.

```
MSCMml.Output = Chr$(254) 'Enter Command Mode
MSCMml.Output = Chr$(247) 'EWC Disable All Device Except Command
GetI 'Retrieve Key1
GetI 'Retrieve Key2
GetI 'Retrieve Key3
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSCMml.InBufferCount > 0
    GetI = Asc(MSCMml.Input)
End Function
```

Require Keys Wireless Networking Test Program

Wireless Keys Networking Functions are demonstrated using the "Wireless Keys" test program installed in the Examples .ZIP archive, which is included with the "Ultra Series Configuration and Test Software".

This application covers:

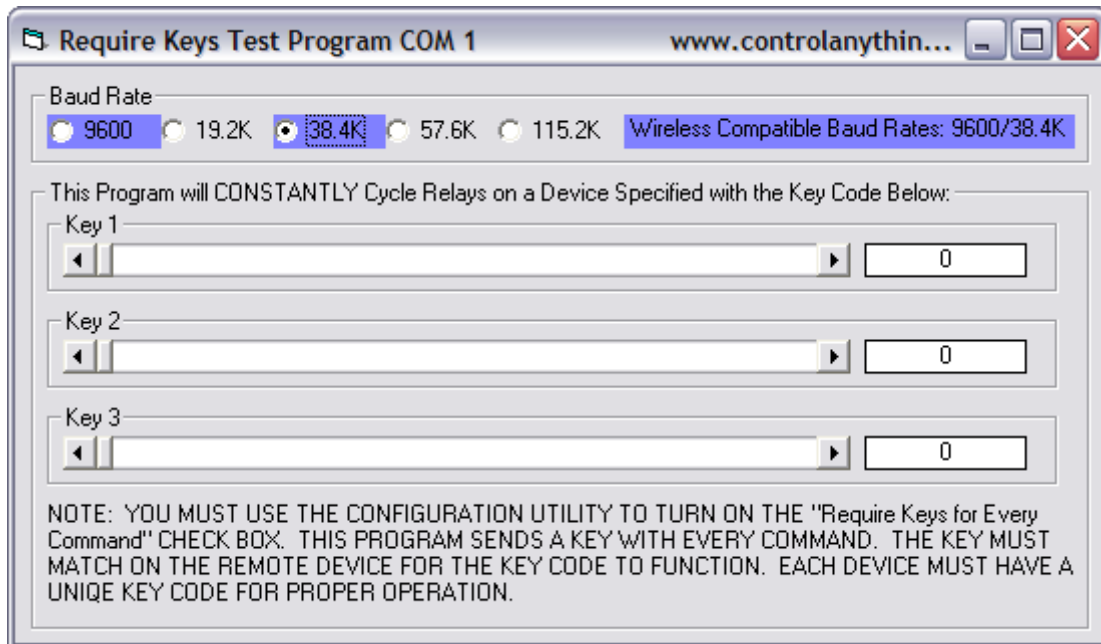
Sending relay control commands to a specific device using the "Require Keys" networking method.

This program provides a simple GUI that allows you test the Require Keys function of the Ultra series controller.

The mechanics of this program will be discussed on the following pages.

Before Using the Require Keys Networking Method:

Set the Ultra Controller to Test Mode and select the "Require Keys for Every Command" option and store these settings into the controller. Set the DIP switches to any communication speed you prefer, except Test Mode, and power cycle the controller. Keys will now be required to execute every command.



Normal Command Structure

The command below are a few typical commands you would send when "Keys Required" is DISABLED. This is the normal command structure as found throughout the manual.

254, 9, 0	8-Bit Read Channel 0
254, 9, 8	10-Bit Read Channel 0
254, 10,Bank,16	Send On/Off Status of Relay 0
254, 10,Bank,23	Send On/Off Status of Relay 7

Keys Required Command Structure

Enabling the "Keys Required" networking method, a "key" is required to "Unlock" a command. Every command must have a Key. ALL COMMANDS IN THIS MANUAL ARE AFFECTED BY THE KEYS REQUIRED NETWORKING METHOD. Examples of new commands are found below.

Key1 = Value 0-255
Key2 = Value 0-255
Key3 = Value 0-255

Keys Sent MUST MATCH the Device Number Keys Stored into the Controller for Commands to Function.

254, Key1, Key2, Key3, 9, 0	8-Bit Read Channel 0
254, Key1, Key2, Key3, 9, 8	10-Bit Read Channel 0
254, Key1, Key2, Key3, 10,Bank,16	Send On/Off Status of Relay 0
254, Key1, Key2, Key3, 10,Bank,23	Send On/Off Status of Relay 7

NOTICE: EWC COMMANDS CANNOT BE USED WHEN THE KEYS REQUIRED NETWORKING METHOD IS ENABLED.

NEVER MIX EWC COMMANDS AND KEYS REQUIRED COMMANDS. CHOOSE A WIRELESS NETWORKING METHOD AND DO NOT MIX METHODS WITHIN RADIO RANGE OF EACH OTHER.

Extra Information about Wireless Communications

The Ultra Series relay controllers were the very first wireless devices we ever developed for commercial applications, though our wireless experience dates back to 1998. We spent over nine months on the Ultra Series project, going through a record 6 prototypes. We believe we have established a very solid footing for the future growth of wireless devices. Every wireless device we create has communication algorithms that are based on the Ultra Series controllers. Though the ES Series operate at 916 MHz, it does not appear to be affected by 900 MHz phones, computers, or any other electronic device. We have subjected the Ultra series controllers to a series of reliability tests and found it to perform exceedingly well. Some of the communications protocols we have developed have more than doubled the actual usable range of the device. We believe the Ultra Series to be our best engineering accomplishment to date.

We hope to release firmware upgrades for the Ultra Series controllers that will allow for additional expansion via the Port A and Port B expansion plugs. We fully intend to release a line of wireless sensors that can be programmed to trigger relays on NCD wireless devices. Regardless of the application, we believe our new Ultra Series controllers to be a truly uniquely powerful device, combining technologies in a way that have never been mixed before.

At the time of writing, the TranSendES is the only compatible wireless communicator for remote operation of the Ultra Series relay controllers.

Did you know?

You can control up to 512 Relays using the XR Relay Expansion Ports found on the Ultra Series Controllers. Port A and Port B expansion modules are planned for late 2005. We are also planning to offer wireless sensors that can activate relays automatically.

Characteristics Data	Minimum	Maximum
Relay Activation Time	>5 ms	<15 ms
Relay Deactivation Time	>5 ms	<20 ms
Power Supply voltage	10 VDC	15 VDC
Current Consumption, Stand By, No Relays Activated, No LED Backlight	70 ma	200 ma
Current Consumption of LED Backlight, Brightness 40	40 ma	50 ma
Current Consumption of LED Backlight, Brightness 255	280 ma	295 ma
Current Consumption per Activated Relay	30 ma	35 ma
Current Consumption All Features Fully Active	800 ma	1 Amp
Wireless Communication Distance from PC using TranSend Device Communicator	N/A	Aprox. 800-1200 Feet
Wireless Frequency	-	916 MHz
Wireless Data Rate	5KBPS	40KBPS
Maximum Allowed Activation Time per Relay (Relay Held in On State)	N/A	Unlimited
Expected Operational Life, Non-DPDT Models	>10,000,000 Cycles	N/A
Expected Operational Life, DPDT Models	>2,000,000 Cycles	N/A
Typical Operational Cycles Per Minute	N/A	1,800
Operational Temperature Range (Based on Data Sheets of Components Used, Excluding LCD Display)	-55°C	+125°C
Storage Temperature Range (Based on Data Sheets of Components Used, Excluding LCD Display)	-65°C	+150°C

Ultra Series Expansion Modules

Designed to Grow with your Needs

One of the most powerful features of the Ultra Series controllers is their ability to grow as your needs grow. Instead of replacing hardware, Ultra Controllers allow you to continuously add new hardware by chaining a large variety of expansion modules to the expansion ports (also known as Ultra Ports). By chaining expansion modules, you can build the control system that best suit your specific application. Our initial release of seven expansion modules, supported by version 2.0 firmware, will handle most control applications you may require. But rest assured, many more expansion modules are planned for release in the coming years as we standardize our products and product line to an expansion oriented architecture.

It All Starts Here: A TranSendES is Connected to a PC, Offering a Wireless Connection to NCD Devices such as the WIOADX.

Ultra Series Controllers Communicate to the TranSendES via Wireless Communications. The WIOADX Shown Below becomes the Central "Hub" for all of the expansion modules. This device can also be used by itself, with 16 I/O Lines and an RS-232 Direct Connection to your PC.



And to think the controller shown above is using only 6.8% of its total expansion capabilities...

With the capacity to control 512 relays, read 4096 contact closure/voltage detection inputs, 96 channels of 12 Bit A/D, the WIOAXR is shown above with only 112 relays attached to its output, 96 voltage detection inputs, 96 contact closure inputs, and 16 Channels 12 Bit A/D...and the network is ready for more anytime you are. But if that still isn't enough, you can communicate with 1.67 Million WIOADX controllers.

Version 2.0 Firmware, Release Date, Aug. 22, 2005

Firmware Version 2.0

Ultra Series Firmware Release Version 2.0 Begins Shipping on August 22, 2005. All products ordered on or after this date will receive version 2.0 firmware. Firmware upgrades for existing controllers are available by contacting ryan@controlanything.com.

Version 2.0 Firmware Identification

Version 2.0 firmware is identified by showing V2.0 on the optional character LCD display. The firmware version is also displayed using the Ultra Configuration utility (Firmware version is shown on the bottom center of the configuration window). Version 2.0 has three components:

V2.0 Product Enhancements:

TranSend Encoder Chip Update

The TranSend Encoder chip was updated to communicate data slightly faster. In addition, new default parameters have been established. SID error checking is not recommended for version 2.0 firmware, but is still usable. The greatest speed benefit and reliability is achieved by keeping SID error checking off. Data transmission routines were updated to improve range. Effective communication distance has been tested at over 1000 feet, up from 500 feet using version 1.0 firmware.

Software Updates

The Configuration and Test software also received a new parameter that allows you to adjust how many times data packets are sent back to the computer. A good value is 2 or 3, but you can choose to transmit return data packets up to 10 times. This parameter greatly increases communication range, but there is a serious speed penalty for every value you choose, so keep the number as low as possible.

In addition, new Version 2.0 test software is included in the archive. Please see below for additional information on test software.

Ultra Series Firmware Controller Updates

The Ultra Series Firmware has been updated for better communication range back to a PC. Effective communication range is now over 1000 feet with improved speed and reliability. Firmware upgrades also allow you to send multiple data packets back to the remote computer, improving range and reliability, but this parameter increase comes at a sacrifice of communication speed.

Version 2.0 Firmware Now supports many new expansion modules, allowing you to read up to 96 12-Bit Analog Channels and up to 4096 Switch Closure and/or Voltage Detector inputs (in any combination). The Ultra Series controllers can now be relied upon for very serious security and monitoring applications. With the new power of highly expandable inputs, Ultra Series controllers can read incoming data just as well as it is able to control large arrays of relays.

The following new expansion devices can now be plugged into expansion ports A or B to significantly increase the input monitoring capabilities of the Ultra Series controllers:

UAD1216	16-Channel 12-Bit A/D Converter. You can chain up to 3 of these on each I/O expansion port, allowing you to read up to 96 12-Bit or 8-Bit Analog Voltages from 0-5VDC.
USCV16x	Ultra Expansion Module ScanVolt 16 Channel. This device is used to detect the presence of voltages. Each input is 100% optoisolated with a wide user-controlled input voltage range. Daisy Chain multiple expansion modules to the same I/O buss. This device counts as 2 Input Banks on the I/O Expansion port. Up to 256 Banks allowed per port.
USCV32x	Same as Above, but with 32 Channels, Counts as 4 Input Banks.
USCV48x	Same as Above, but with 48 Channels, Counts as 6 Input Banks.
USCS16x	Ultra Expansion Module ScanSwitch 16 Channel. This device is used to detect switch closures, useful for motion detection and light switch monitoring. Daisy Chain multiple expansion modules to the same I/O buss. This device counts as 2 Input Banks on the I/O Expansion port. Up to 256 Banks allowed per port.
USCS32x	Same as Above, but with 32 Channels, Counts as 4 Input Banks.
USCS48x	Same as Above, but with 48 Channels, Counts as 6 Input Banks.

Understanding Banks

Understanding the Concept of Banks will become increasingly important as you expand your Ultra Series controller. By this point, you have run across the term "Bank" as it applies to a group of eight relays. Similarly, "Banks" will be used as a generic term for "a grouping of eight". A bank of inputs is a group of eight inputs. Using some of our new expansion modules, a "Bank" parameter will be required. In many cases, you can read inputs from 256 banks (which translates into 2048 individual inputs). Using banks, you will be able to read switch closure inputs using our ScanSwitch expansion modules. Our ScanVolt expansion module is used to detect the presence of a voltage (which is very useful for confirming the actual switch closure of a relay). Complete details will be discussed as we introduce you to our line of Ultra Series expansion modules.

Version 2.0 Firmware is Required for All Programming and Examples and Expansion Modules Shown Beyond this Point

UAD1216: 16-Channel 8-Bit / 12-Bit Analog to Digital Conversion

UAD1216 Introduction

The UAD1216 is an Ultra Series expansion module designed to plug into the Ultra Series Expansion Port A or B. Up to three UAD1216 expansion modules can be connected to a single port, allowing up to six UAD1216 expansions per ultra series controller (for a total of 96 Channels of Analog to Digital conversion).

What is Analog to Digital Conversion?

A/D conversion is the process of converting a voltage to a numeric value. A/D conversion is useful for reading light or sound levels in a room, temperature (using a thermistor), water levels (using only a couple of wires), not to mention all kinds of analog sensors such as pressure sensors, thermo-couples, and much more. Think of the AD1216 as sixteen volt meters, each capable of reading voltages from 0-5 DC volts. Instead of each volt meter having a display, the voltage values are delivered to your PC for manipulation by your own program.

8-Bit Conversion

The user may request an 8-Bit A/D conversion. When the appropriate command is executed, the Ultra Controller will deliver a value from 0-255 indicating voltage. A value of 0 will be returned if no voltage is present on the input. A value of 255 will be returned if the voltage is at 5 volts on the input. A value of 128 will be returned if the voltage is at 2.5 volts. An 8-Bit A/D conversion breaks apart the input into 256 equal steps, and is sensitive to voltage changes as low as .192 volts.

12-Bit Conversion

The user may also request a 12-Bit A/D conversion. When the appropriate command is executed, the Ultra Controller will deliver a value from 0-4095 indicating voltage. A value of 0 will be returned if no voltage is present on the input. A value of 4095 will be returned if the voltage is at 5 volts on the input. A value of 2048 will be returned if the voltage is at 2.5 volts. A 12-Bit A/D conversion breaks apart the input into 4096 equal steps, and is sensitive to voltage changes as low as .00122 volts. A 12-Bit A/D conversion takes just as long as an 8-Bit A/D conversion. However, 12-Bit conversions appear to take longer because data has to be delivered back to the computer using two bytes instead of just one for 8-Bit conversions. These two bytes are then "re-assembled" into a 0-4095 value. More on this later.

Multi-Channel Conversion

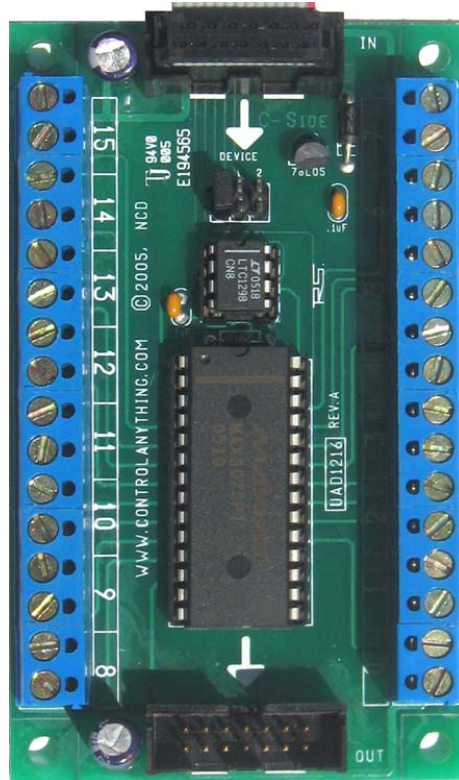
Ultra Series controllers allow you to read a single channel or all 16 inputs using a single request. Requesting all 16 input channels at one time can significantly improve communication speed.

Expansion modules are designed to be chained off the I/O port buss. You can easily mix voltage detectors, switch scanners, and analog to digital converters off the same data port. No power supply is required for this expansion module, the I/O port provides the required power supply.

This Connector Attaches to the Ultra Series Controller

White Arrows Printed on the Circuit Board Should ALWAYS Point AWAY from the Main Controller

Ground
Input Channel 15
Ground
Input Channel 14
Ground
Input Channel 13
Ground
Input Channel 12
Ground
Input Channel 11
Ground
Input Channel 10
Ground
Input Channel 9
Ground
Input Channel 8



Input Channel 7
Ground
Input Channel 6
Ground
Input Channel 5
Ground
Input Channel 4
Ground
Input Channel 3
Ground
Input Channel 2
Ground
Input Channel 1
Ground
Input Channel 0
Ground

This Connector Attaches to More Expansion Modules
(Up to 2 more UAD1216 Expansion Boards per Ultra Port)

The UAD1216 Must Be Connected to the I/O Data Buss (Ultra Port) First. Voltage Detectors and Contact Closure Input Expansions Must be at the End of the Chain.

All Inputs are 0-5VDC. Power to this Device is Derived from the Ultra Expansion Port.

UAD1216: 16-Channel 8-Bit / 12-Bit Analog to Digital Conversion

The UAD1216 Can be Connected to Port A or Port B, Select the Port it is Connected to.

Select a Baud Rate

Up to 3 UAD1216s can Connect to a Single Port, Select the Device you would Like to Communicate with. Make Sure the Device Jumper on the UAD1216 Matches this Setting.

Channel 0 48/255
Channel 1 44/255
Channel 2 25/255
Channel 3 23/255
Channel 4 0/255
Channel 5 0/255
Channel 6 1/255
Channel 7 9/255
Channel 8 19/255
Channel 9 35/255
Channel 10 29/255
Channel 11 27/255
Channel 12 21/255
Channel 13 12/255
Channel 14 0/255
Channel 15 0/255

The value of all 16 channels will be shown in this window. If the input is not tied to anything, the display value will change randomly.

Up to 96 Channels can be monitored using a single Ultra Controller and Six UAD1216 Expansion Modules.

204 Samples per Second **Time of Last Error: 68238.58**

Decrease the TimeOut Value to Increase Samples per Second. Setting this value too low will increase Communication Errors. The time of the last communication error is shown next to Samples per Second. Set the slider below to the lowest possible value that causes as few communication errors as possible. Compile this program to increase speed.

3244

Program Variations in the Examples.ZIP File:

UAD1216 16-Channel One at a Time 8-Bit (COM1-COM6 Supported)
Reads One Channel at a Time, All 16 Channels, 8-Bits per Channel.

UAD1216 16-Channel One at a Time 12-Bit (COM1-COM6 Supported)
Reads One Channel at a Time, All 16 Channels, 12-Bits per Channel.

UAD1216 16-Channel Requests 8-Bit (COM1-COM6 Supported)
Reads All 16 Channels at a Time, 8-Bits per Channel.

UAD1216 16-Channel Requests All 12-Bit (COM1-COM6 Supported)
Reads All 16 Channels at a Time, 12-Bits per Channel.

UAD1216: 16-Channel 8-Bit / 12-Bit Analog to Digital Conversion

Programming Examples

Version 2.0 Firmware includes a new set of commands that allow you to communicate to the UAD1216 on Port A or Port B. You can also request data one channel at a time or all 16 channels at once. You may also request 8-Bit values (0-255), or high resolution 12-Bit values (0-4095). Up to 3 devices are supported per port, for a total of 96 analog inputs per Ultra controller.

254, 12, Device, Channel: 8-Bit Single Channel Port A

This command reads a single 8-Bit channel from the UAD1216 Expansion Module connected to Expansion Port A. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command also requires a Channel parameter, indicating which input channel to read from (0-15). This command will return a single byte of data, from 0 to 255, indicating the analog value on the specified input channel.

254, 13, Device, Channel: 8-Bit Single Channel Port B

This command reads a single 8-Bit channel from the UAD1216 Expansion Module connected to Expansion Port B. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command also requires a Channel parameter, indicating which input channel to read from (0-15). This command will return a single byte of data, from 0 to 255, indicating the analog value on the specified input channel.

254, 14, Device: 8-Bit All Channels Port A

This command reads all 16 8-Bit channels from the UAD1216 Expansion Module connected to Expansion Port A. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command will return sixteen bytes of data, from 0 to 255, indicating the analog values of all input channels, beginning with channel 0.

254, 15, Device: 8-Bit All Channels Port B

This command reads all 16 8-Bit channels from the UAD1216 Expansion Module connected to Expansion Port B. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command will return sixteen bytes of data, from 0 to 255, indicating the analog values of all input channels, beginning with channel 0.

254, 16, Device, Channel: 12-Bit Single Channel Port A

This command reads a single 12-Bit channel from the UAD1216 Expansion Module connected to Expansion Port A. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command also requires a Channel parameter, indicating which input channel to read from (0-15). This command will return two bytes of data (LSB, then MSB), which will be re-assembled by your program to a value of 0-4095 using the equation: $Value = (MSB * 256) + LSB$.

254, 17, Device, Channel: 12-Bit Single Channel Port B

This command reads a single 12-Bit channel from the UAD1216 Expansion Module connected to Expansion Port B. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command also requires a Channel parameter, indicating which input channel to read from (0-15). This command will return two bytes of data (LSB, then MSB), which will be re-assembled by your program to a value of 0-4095 using the equation: $Value = (MSB * 256) + LSB$.

8-Bit Single Channel Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(12) 'UAD1216 Port A 8-Bit Commands
MSComm1.Output = Chr$(0) 'UAD1216 Device 0
MSComm1.Output = Chr$(0) 'Request 8-Bit A/D Channel 0
GetI 'GetI Contains the 8-Bit A/D Value
```

8-Bit Single Channel Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(13) 'UAD1216 Port B 8-Bit Commands
MSComm1.Output = Chr$(2) 'UAD1216 Device 2
MSComm1.Output = Chr$(15) 'Request 8-Bit A/D Channel 15
GetI 'GetI Contains the 8-Bit A/D Value
```

8-Bit All Channels Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(14) 'UAD1216 Port A 8-Bit Commands
MSComm1.Output = Chr$(1) 'UAD1216 Device 1
For Channel = 0 To 15
    Debug.Print Channel;GetI 'Get Data Byte from Controller
Next Channel
```

8-Bit All Channels Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(15) 'UAD1216 Port B 8-Bit Commands
MSComm1.Output = Chr$(1) 'UAD1216 Device 1
For Channel = 0 To 15
    Debug.Print Channel;GetI 'Get Data Byte from Controller
Next Channel
```

12-Bit Single Channel Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(16) 'UAD1216 Port A 12-Bit Commands
MSComm1.Output = Chr$(0) 'UAD1216 Device 0
MSComm1.Output = Chr$(7) 'Request 12-Bit A/D Channel 7
LSB = GetI 'GetI Contains the LSB 12-Bit A/D Value
MSB = GetI 'GetI Contains the MSB 12-Bit A/D Value
Value = (MSB*256)+LSB 'This Equation Converts to 12-Bit Value
Debug.Print Value 'Show 12-Bit A/D Value in the Debug Window
```

12-Bit Single Channel Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(17) 'UAD1216 Port B 12-Bit Commands
MSComm1.Output = Chr$(0) 'UAD1216 Device 0
MSComm1.Output = Chr$(7) 'Request 12-Bit A/D Channel 7
LSB = GetI 'GetI Contains the LSB 12-Bit A/D Value
MSB = GetI 'GetI Contains the MSB 12-Bit A/D Value
Value = (MSB*256)+LSB 'This Equation Converts to 12-Bit Value
Debug.Print Value 'Show 12-Bit A/D Value in the Debug Window
```

Reading Data from the Controller

The *GetI* Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

UAD1216: 16-Channel 8-Bit / 12-Bit Analog to Digital Conversion

254, 18, Device: 12-Bit All Channels Port A

This command reads all 16 8-Bit channels from the UAD1216 Expansion Module connected to Expansion Port A. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command will return 32 bytes of data indicating the 12-Bit analog values of all 16 input channels, beginning with channel 0. The first two bytes of data indicate the analog values for channel 0. The next two bytes of data indicate the analog values for channel 1. The process ends by sending the final two bytes of data indicating the 12-Bit analog value for channel 15. Each pair of data bytes are sent in the order LSB, then MSB, which stands for the Least Significant Byte and Most Significant byte. The 12-Bit value is re-assembled by your program to a value of 0-4095 using the equation: $Value = (MSB * 256) + LSB$. This equation will be used 16 times (for each pair of 32 bytes), during the conversion process.

254, 19, Device: 12-Bit All Channels Port B

This command reads all 16 8-Bit channels from the UAD1216 Expansion Module connected to Expansion Port B. This command requires a Device parameter (0, 1, or 2), which should match the Device jumper setting of the UAD1216 you would like to read from. This command will return 32 bytes of data indicating the 12-Bit analog values of all 16 input channels, beginning with channel 0. The first two bytes of data indicate the analog values for channel 0. The next two bytes of data indicate the analog values for channel 1. The process ends by sending the final two bytes of data indicating the 12-Bit analog value for channel 15. Each pair of data bytes are sent in the order LSB, then MSB, which stands for the Least Significant Byte and Most Significant byte. The 12-Bit value is re-assembled by your program to a value of 0-4095 using the equation: $Value = (MSB * 256) + LSB$. This equation will be used 16 times (for each pair of 32 bytes), during the conversion process.

12-Bit All Channels Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(18) 'UAD1216 Port A 12-Bit Commands
MSComm1.Output = Chr$(0) 'UAD1216 Device 0
For Ch = 0 to 15 'Count Through Channels
  LSB = GetI 'GetI Contains the LSB 12-Bit A/D Value
  MSB = GetI 'GetI Contains the MSB 12-Bit A/D Value
  Value = (MSB*256)+LSB 'This Equation Converts to 12-Bit Value
  Debug.Print Ch;Value 'Show 12-Bit A/D Value in the Debug Window
Next Ch
```

12-Bit All Channels Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(19) 'UAD1216 Port B 12-Bit Commands
MSComm1.Output = Chr$(0) 'UAD1216 Device 0
For Ch = 0 to 15 'Count Through Channels
  LSB = GetI 'GetI Contains the LSB 12-Bit A/D Value
  MSB = GetI 'GetI Contains the MSB 12-Bit A/D Value
  Value = (MSB*256)+LSB 'This Equation Converts to 12-Bit Value
  Debug.Print Ch;Value 'Show 12-Bit A/D Value in the Debug Window
Next Ch
```

Reading Data from the Controller

The *GetI* Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
  T = 0
  Do
    T = T + 1
    If T > 10000 then Exit Function
    DoEvents
  Loop Until MSComm1.InBufferCount > 0
  GetI = Asc(MSComm1.Input)
End Function
```

USCVxxx: 16/32/48-Channel Voltage Detector (ScanVolt)

Is it Really On?

New Expansion Detects Voltages

So... you turned it on, but did it actually come on?

Our new voltage detector expansion modules can help you find up to 4096 potential voltage problems.

ScanVolt USCVxxx Introduction

Ultra ScanVolt 16, 32, and 48 Expansion Modules allow you to detect the presence of a voltage. This is useful for determining if there is ever an electrical failure, or to confirm the actual activation of a relay. ScanVolt inputs are NOT polarity sensitive and are compatible with AC or DC voltages. Each input is rectified with a 400V bridge rectifier AND optoisolated up to 5,000 volts from your Ultra Series controller and computer.

ScanVolt expansion controllers count as 2, 4, or 6 banks on a Port A or Port B expansion buss. Each bank provides you with 8 optically isolated voltage input detectors. You can chain multiple expansion controllers in various combinations to a single port buss. Up to 256 input banks are allowed on either or both data ports, allowing you to detect the presence of 4096 voltages using a single Ultra Controller with ScanVolt expansion modules attached.

Expansion modules are designed to be chained off the I/O port buss. You can easily mix voltage detectors, switch scanners, and analog to digital converters off the same data port. No power supply is required, the I/O port provides the required power supply.

IMPORTANT NOTE:

If you are using this device in combination with the UAD1216 expansion module, the UAD1216 MUST be connected to the Ultra Controller FIRST, you can then chain multiple input expansion boards from the UAD1216.

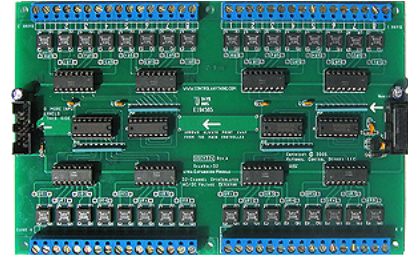
ScanVolt is available with 16, 32, or 48 voltage input channels, and 7 varieties to help you detect many voltage ranges.

BORDER LINE VOLTAGE WARNING:

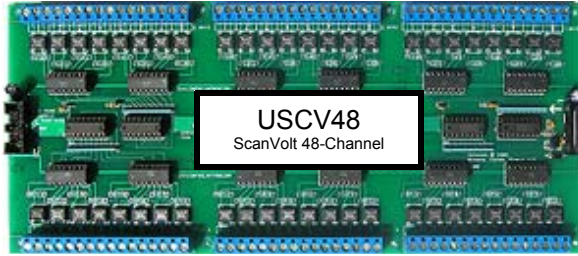
Voltages below the Minimum Detection Voltage ranges shown below may not appear to stay on. Logic circuits may read border line voltages as on or off with a lot of inconsistency. Your software may choose to sample the voltage detect circuit a few times to make sure the voltage is within the detection range. The voltages shown above are the guaranteed ON voltage values.



USCV16
ScanVolt 16-Channel



USCV32
ScanVolt 32-Channel



USCV48
ScanVolt 48-Channel

UD Version

The UD Version of the ScanVolt expansion module does not include optoisolator resistors. This allows you to easily build your own ScanVolt expansion. This is particularly helpful when you need to use one ScanVolt reading a wide variety of voltage ranges.

Using the UD version, you will need to install your own resistors before the ScanVolt will function, the following information will be helpful:

Inputs can be AC or DC and are NOT polarity sensitive.

Inputs are full-bridge rectified prior to optoisolation.

The on-board bridge rectifier drops the incoming voltage by 1.2V.

The on-board bridge rectifier has a 400V input rating.

The Optoisolator has a Minimum ON voltage of 1.0VDC.

The Optoisolator has a Typical ON voltage of 1.2VDC.

The Optoisolator has a Maximum ON voltage of 1.4VDC.

The Optoisolator consumes 50ma at 1.2VDC.

The Quad Optoisolator is www.digikey.com part number 160-1364-5-ND manufactured by Lite-On Inc, Part Number LTV846.

Resistors are installed in **User Resistor** locations **UR1-UR8**, **1 resistor will be required for each input, resistors are not supplied with the UD version of this controller.** The table below shows some common **User Resistor** values and their associated voltage ranges.

Ending Part Number	Description	UR Resistor Value	Minimum Detection Voltage	Maximum Detection Voltage
UD	User Defined: Board Comes Unpopulated with Optoisolation Resistors. Resistors MUST be installed by the user prior to use. This allows the user to specify the detection voltage for each individual channel.	User Installed	User Defined	User Defined
LV	Low Voltage Detection		3V	30V
MV1	Medium Voltage Detection Range 1		25V	50V
MV2	Medium Voltage Detection Range 2		45V	70V
HV1	High Voltage Detection Range 1		65V	90V
HV2	High Voltage Detection Range 2		85V	110V
HV3	High Voltage Detection Range 3		105V	130V

USCSxx: 16/32/48-Channel Switch Closure Input Scanner (ScanSwitch)

Ultra Series Controllers can Now Read up to 4096 Switch Closure Inputs

We developed this device specifically for use at our new design & manufacturing facility...it manages all the light switches, motion detectors, glass breakage detectors, and open window detectors with ease.

USCSxx Introduction

Ultra ScanSwitch 16, 32, and 48 Expansion Modules allow you to detect switch closure inputs, allowing the Ultra controller to easily monitor light switches, motion detectors, door sensors, limit switches, and any other kind of sensor with a switch closure output.

ScanSwitch expansion controllers count as 2, 4, or 6 banks on a Port A or Port B expansion buss. Each bank provides you with 8 switch closure input detectors. You can chain multiple expansion controllers in various combinations to a single port buss. Up to 256 input banks are allowed on either or both data ports, for a total monitoring capability of 4096 switch closure inputs.

Expansion modules are designed to be chained off the I/O port buss. You can easily mix voltage detectors, switch scanners, and analog to digital converters off the same data port. No power supply is required, the I/O port provides the required power supply.

IMPORTANT NOTE:

If you are using this device in combination with the UAD1216 expansion module, the UAD1216 MUST be connected to the Ultra Controller FIRST, you can then chain multiple input expansion boards from the UAD1216.

Connections are the Same for All Models, there is simply more input banks on the other versions. Bank Numbers are Printed on the Circuit Board.

This Connector Attaches to the Ultra Series Controller

White Arrows Printed on the Circuit Board Should ALWAYS Point AWAY from the Main Controller



USCS16
ScanSwitch 16-Channel



USCS32
ScanSwitch 32-Channel

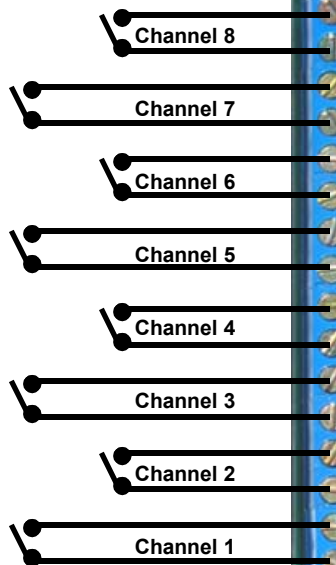


USCS48
ScanSwitch 48-Channel

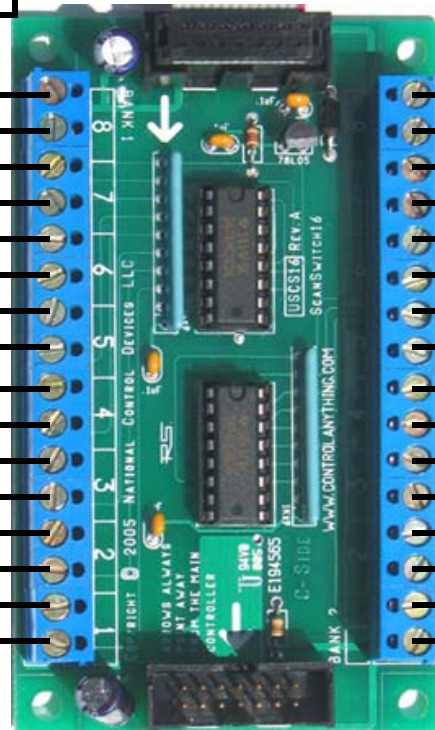
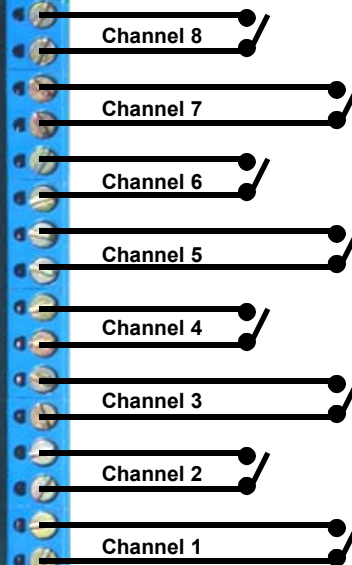
IMPORTANT WARNING:

ScanSwitch controllers should only be used to detect the on/off status of various kinds of switches. ScanSwitch is NOT compatible with any devices that delivers a voltage to the controller (Use our ScanVolt series controller for this application). Applying any kind of voltage to the ScanSwitch inputs can cause severe damage to all attached controllers and computers. Make absolutely certain that ONLY switch closure connections are made to the ScanSwitch expansion module.

Bank 1



Bank 2



All Inputs are 0-5VDC. Power to this Device is Derived from the Ultra Expansion Port.

This Connector Attaches to More Expansion Modules (Up to 256 Banks of 8 per Ultra Port)

Voltage Detectors and Contact Closure Input Expansions Must be at the End of the Ultra Port Chain.

Example Software for Input Scanning Devices

Input Scanning Devices can be Connected to Port A and/or Port B. Make sure this parameter is properly set.

Select a Baud Rate

The SCANx program is capable of displaying 48 channels of inputs at one time (you can mix the ScanSwitch or the ScanVolt expansion modules in any combination). You can control the range of inputs that are scanned using this slider.

When a voltage or switch closure is detected, the corresponding bank turns RED.

In this example, we only had 14 banks attached (totaling 112 inputs) to the Ultra controller when this photo was taken. Unavailable banks appear in RED, and are reported as "all on" by the Ultra Controller. Any programs you write should always know how many banks are available for input scanning.

Up to 4095 Inputs can Be Scanned using a Single Ultra Series Controll + Expansion Modules

178 Samples per Second

Decrease the TimeOut Value to Increase Samples per Second. Setting this value too low will increase Communication Errors. The time of the last communication error is shown next to Samples per Second. Set the slider below to the lowest possible value that causes as few communication errors as possible. Compile this program to increase speed.

7000

Program Variations in the Examples.ZIP File:

SCANx Input Scanning Single Bank (COM1-COM6 Supported)
Reads One Bank at a Time

Expansion Modules: Programming for Input Scanning Devices

Input Module Banks

The ScanSwitch and the ScanVolt expansion modules use the same command set. The logic circuits on these devices are 100% identical, the only difference is the interface circuit. You can use these commands interchangeably for the ScanVolt and the ScanSwitch. Groups of eight inputs are addressed by their "bank". The more devices you have connected to the expansion port, the more banks you have available to read from. Banks are assigned to these devices incrementally. Each time you add a bank of inputs to the expansion port, you increase the potential bank value. The lowest number bank values correspond to the banks that are closest to the input expansion connector. Bank values increase as you get further away from the expansion port on the Ultra controller. You must specify a "Bank" value when using these commands so it is important to understand the numbering sequence of banks. See diagram at right for a visual explanation of banks.

254, 20, Bank: Input Bank Read Port A

This command reads a specified input bank (connected to Port A), and is compatible with ScanVolt and ScanSwitch input expansion modules. This command requires a Bank parameter, with a valid range of 0-255, indicating which input bank you would like to read. This command returns a single byte of data, from 0-255, indicating the on/off status in binary format for all eight inputs (of the selected bank). This byte can then be bit tested to determine if individual inputs are active. See Bit Testing below.

254, 21, Bank: Input Bank Read Port B

This command reads a specified input bank (connected to Port B), and is compatible with ScanVolt and ScanSwitch input expansion modules. This command requires a Bank parameter, with a valid range of 0-255, indicating which input bank you would like to read. This command returns a single byte of data, from 0-255, indicating the on/off status in binary format for all eight inputs (of the selected bank). This byte can then be bit tested to determine if individual inputs are active. See Bit Testing below.

Bit Testing

The programming example below shows you how to "Bit Test" a byte of data to determine if individual inputs are on/off. It is at least 14 times faster to offload this task on your computer, rather than actual device, especially when you are working with large arrays of inputs. Bit Testing uses the mathematical "And" function to determine if bits are active or not. Virtually all programming languages that support math functions support the "And" function, as it is an essential function for most logical operations.

Note that VALUE below is the value reported back to your computer from the Input Expansion controller. You specified the Bank you wanted to read, now the controller is reporting back the VALUE generated by your inputs on the selected bank.

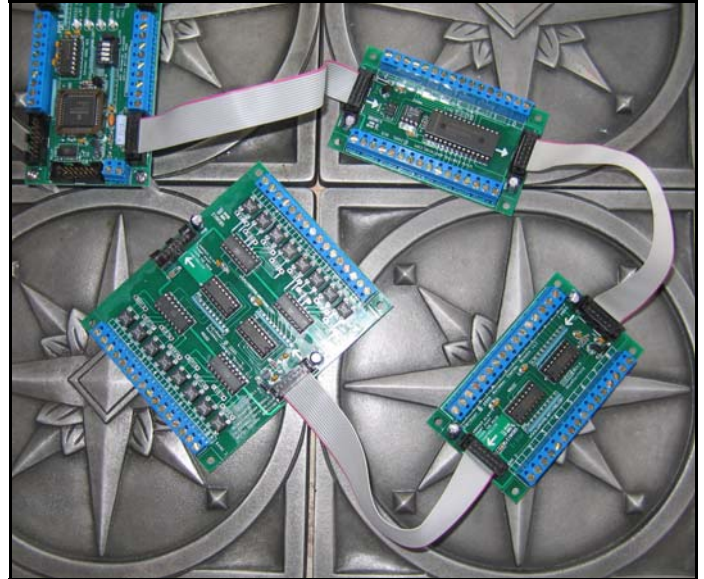
```
VALUE = GetI           'Get a Data Byte from the Controller
If (VALUE AND 1) = 1 then 'If Input Bit 0 is ON Then
                        'Execute these lines of code
Else
                        'Otherwise the Bit is OFF
                        'Execute these lines of code
endif
```

Here is an example that tests all input bits:

```
VALUE = GetI           'Get a Data Byte from the Controller

If (VALUE AND 1) = 1 then 'If Bit 0 (Input 0) is in the ON State
If (VALUE AND 2) = 2 then 'If Bit 1 (Input 1) is in the ON State
If (VALUE AND 4) = 4 then 'If Bit 2 (Input 2) is in the ON State
If (VALUE AND 8) = 8 then 'If Bit 3 (Input 3) is in the ON State
If (VALUE AND 16) = 16 then 'If Bit 4 (Input 4) is in the ON State
If (VALUE AND 32) = 32 then 'If Bit 5 (Input 5) is in the ON State
If (VALUE AND 64) = 64 then 'If Bit 6 (Input 6) is in the ON State
If (VALUE AND 128) = 128 then 'If Bit 7 (Input 7) is in the ON State

If (VALUE AND 1) = 0 then 'If Bit 0 (Input 0) is in the OFF State
If (VALUE AND 2) = 0 then 'If Bit 1 (Input 1) is in the OFF State
If (VALUE AND 4) = 0 then 'If Bit 2 (Input 2) is in the OFF State
If (VALUE AND 8) = 0 then 'If Bit 3 (Input 3) is in the OFF State
If (VALUE AND 16) = 0 then 'If Bit 4 (Input 4) is in the OFF State
If (VALUE AND 32) = 0 then 'If Bit 5 (Input 5) is in the OFF State
If (VALUE AND 64) = 0 then 'If Bit 6 (Input 6) is in the OFF State
If (VALUE AND 128) = 0 then 'If Bit 7 (Input 7) is in the OFF State
```



The photo above shows how expansion modules can be chained to the Ultra Port, adding features to the main controller. Expansion modules simply connect end to end, and are designed to work together. The UAD1216 is shown first in the chain. For proper operation, it is important to keep input scanning devices, such as the ScanSwitch and ScanVolt at the end of the chain. These devices modify data that passes along the data port. The UAD1216 is a pass-through device, it does not modify any data on the port. Pass-through devices should be connected first, then data port modification devices.

Input Bank Read Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(20) 'Read Input on Port A
MSComm1.Output = Chr$(1) 'Read Bank 1
VALUE = GetI 'Get Value from Controller
If (VALUE AND 1) = 1 then 'If Bit 0 (Input 0) is in the ON State
If (VALUE AND 2) = 2 then 'If Bit 1 (Input 1) is in the ON State
If (VALUE AND 4) = 4 then 'If Bit 2 (Input 2) is in the ON State
If (VALUE AND 8) = 8 then 'If Bit 3 (Input 3) is in the ON State
If (VALUE AND 16) = 16 then 'If Bit 4 (Input 4) is in the ON State
If (VALUE AND 32) = 32 then 'If Bit 5 (Input 5) is in the ON State
If (VALUE AND 64) = 64 then 'If Bit 6 (Input 6) is in the ON State
If (VALUE AND 128) = 128 then 'If Bit 7 (Input 7) is in the ON State
```

Input Bank Read Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(21) 'Read Input on Port B
MSComm1.Output = Chr$(0) 'Read Bank 0
VALUE = GetI 'Get Value from Controller
If (VALUE AND 1) = 1 then 'If Bit 0 (Input 0) is in the ON State
If (VALUE AND 2) = 2 then 'If Bit 1 (Input 1) is in the ON State
If (VALUE AND 4) = 4 then 'If Bit 2 (Input 2) is in the ON State
If (VALUE AND 8) = 8 then 'If Bit 3 (Input 3) is in the ON State
If (VALUE AND 16) = 16 then 'If Bit 4 (Input 4) is in the ON State
If (VALUE AND 32) = 32 then 'If Bit 5 (Input 5) is in the ON State
If (VALUE AND 64) = 64 then 'If Bit 6 (Input 6) is in the ON State
If (VALUE AND 128) = 128 then 'If Bit 7 (Input 7) is in the ON State
```

Reading Data from the Controller

The GetI Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
    DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Example Software for Input Scanning Devices

Input Scanning Devices can be Connected to Port A and/or Port B. Make sure this parameter is properly set.

"Input Spanning" allows you to read a group of inputs at one time (up to 32). Spanning is significantly faster when reading large arrays of inputs because there is less communication to the Ultra controller. One request is made for up to 32 channels of data rather than 32 requests for 32 channels. These sliders are used to set the range of inputs you would like to "Span", beginning with the starting bank. The "Span" is set by the second slider, it controls how many banks will be read. Each bank will report back one byte of data. You can span consecutive inputs beginning with any bank.

Select a Baud Rate

When a voltage or switch closure is detected, the corresponding bank turns RED.

In this example, we only had 14 banks attached (totaling 112 inputs) to the Ultra controller when this photo was taken. Unavailable banks appear in RED, and are reported as "all on" by the Ultra Controller. Any programs you write should always know how many banks are available for input scanning.

Note the significant increase in samples per second when requesting inputs in groups rather than individually.

Program Variations in the Examples.ZIP File:

SCANx Input Scanning Multi-Bank (COM1-COM6 Supported)
Reads a Span of Banks at a One Time and Sends Multiple Bytes of Data Back to your Computer

Expansion Modules: Programming for Input Scanning Devices

Input Module Banks

The ScanSwitch and the ScanVolt expansion modules use the same command set. The logic circuits on these devices are 100% identical, the only difference is the interface circuit. You can use these commands interchangeably for the ScanVolt and the ScanSwitch. Groups of eight inputs are addressed by their "bank". The more devices you have connected to the expansion port, the more banks you have available to read from. Banks are assigned to these devices incrementally. Each time you add a bank of inputs to the expansion port, you increase the potential bank value. The lowest number bank values correspond to the banks that are closest to the input expansion connector. Bank values increase as you get further away from the expansion port on the Ultra controller. You must specify a "Bank" value when using these commands so it is important to understand the numbering sequence of banks. See diagram at right for a visual explanation of banks.

254, 22, Start, Span: Input Banks Multi-Read Port A

This command reads a range of inputs (or "Span") on an input module connected to Port A. This command requires two parameters. The first parameter is the Starting bank you would like the controller to read. The Span parameter is used to indicate how many banks you would like the controller to return (up to 32). The Start + Span Values should never exceed 255. The Span value should not exceed 32 on a direct RS-232 connection and 16 on a wireless connection.

Example Usage:

```
254, 22, 10, 5    Returns 5 Bytes, Starting with Bank 10
254, 22, 0, 32   Returns 32 Bytes, Starting with Bank 0
254, 22, 245, 10 Returns 10 Bytes, Starting with Bank 245
```

254, 23, Start, Span: Input Banks Multi-Read Port B

This command reads a range of inputs (or "Span") on an input module connected to Port B. This command requires two parameters. The first parameter is the Starting bank you would like the controller to read. The Span parameter is used to indicate how many banks you would like the controller to return (up to 32). The Start + Span Values should never exceed 255. The Span value should not exceed 32 on a direct RS-232 connection and 16 on a wireless connection.

Bit Testing

The programming example below shows you how to "Bit Test" a byte of data to determine if individual inputs are on/off. It is at least 14 times faster to offload this task on your computer, rather than actual device, especially when you are working with large arrays of inputs. Bit Testing uses the mathematical "And" function to determine if bits are active or not. Virtually all programming languages that support math functions support the "And" function, as it is an essential function for most logical operations.

Note that VALUE below is the value reported back to your computer from the Input Expansion controller. You specified the Bank you wanted to read, now the controller is reporting back the VALUE generated by your inputs on the selected bank.

```
VALUE = GetI           'Get a Data Byte from the Controller
If (VALUE AND 1) = 1 then 'If Input Bit 0 is ON Then
    'Execute these lines of code
Else                    'Otherwise the Bit is OFF
    'Execute these lines of code
endif
```

Here is an example that tests all input bits:

```
VALUE = GetI           'Get a Data Byte from the Controller
If (VALUE AND 1) = 1 then 'If Bit 0 (Input 0) is in the ON State
If (VALUE AND 2) = 2 then 'If Bit 1 (Input 1) is in the ON State
If (VALUE AND 4) = 4 then 'If Bit 2 (Input 2) is in the ON State
If (VALUE AND 8) = 8 then 'If Bit 3 (Input 3) is in the ON State
If (VALUE AND 16) = 16 then 'If Bit 4 (Input 4) is in the ON State
If (VALUE AND 32) = 32 then 'If Bit 5 (Input 5) is in the ON State
If (VALUE AND 64) = 64 then 'If Bit 6 (Input 6) is in the ON State
If (VALUE AND 128) = 128 then 'If Bit 7 (Input 7) is in the ON State

If (VALUE AND 1) = 0 then 'If Bit 0 (Input 0) is in the OFF State
If (VALUE AND 2) = 0 then 'If Bit 1 (Input 1) is in the OFF State
If (VALUE AND 4) = 0 then 'If Bit 2 (Input 2) is in the OFF State
If (VALUE AND 8) = 0 then 'If Bit 3 (Input 3) is in the OFF State
If (VALUE AND 16) = 0 then 'If Bit 4 (Input 4) is in the OFF State
If (VALUE AND 32) = 0 then 'If Bit 5 (Input 5) is in the OFF State
If (VALUE AND 64) = 0 then 'If Bit 6 (Input 6) is in the OFF State
If (VALUE AND 128) = 0 then 'If Bit 7 (Input 7) is in the OFF State
```

Input Banks Multi-Read Port A

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(22)  'Read Multi-Bank Input on Port A
MSComm1.Output = Chr$(0)   'Read Start at Bank 0 (Start Bank)
MSComm1.Output = Chr$(10)  'Read 10 Banks of Inputs (Span)
For Bank = 0 to 9          'Count Through Input Banks
    VALUE = GetI           'Get Value from Controller
    Debug.Print Bank;Value 'Show Bank Data in Debug Window
Next Ch
```

Input Banks Multi-Read Port B

```
MSComm1.Output = Chr$(254) 'Command Mode
MSComm1.Output = Chr$(23)  'Read Multi-Bank Input on Port A
MSComm1.Output = Chr$(5)   'Read Start at Bank 5 (Start Bank)
MSComm1.Output = Chr$(15)  'Read 15 Banks of Inputs (Span)
For Bank = 5 to 20         'Count Through Input Banks
    VALUE = GetI           'Get Value from Controller
    Debug.Print Bank;Value 'Show Bank Data in Debug Window
Next Ch
```

Reading Data from the Controller

The *GetI* Function Below waits for data from the controller. If data is not received before the timeout period, the function will exit, containing no value.

```
Public Function GetI()
    T = 0
    Do
        T = T + 1
        If T > 10000 then Exit Function
        DoEvents
    Loop Until MSComm1.InBufferCount > 0
    GetI = Asc(MSComm1.Input)
End Function
```

Introducing the WIOADX Controller

The WIOADX controller is simply an Ultra Relay controller without the relays. It has the same firmware as the Ultra Series 16-Relay Controllers, and DIP switches operate the same way, wireless communications and programming are all identical. So please reference earlier parts of this manual for detailed instructions for communicating to this device. We stripped out the relays from this design to lower cost for our customers. We also designed this version for our own internal use. In 2005, we built a new office building, consolidating our engineering, manufacturing, design, order processing, and accounting departments into a unified workspace. This controller is the heart of the entire system, fully automating our new facility with computer controlled lighting, light switch monitoring, and motion detection. It has control of our outlets and soldering equipment, making sure everything is off at the end of the day. We believe our customers will find the WIOADX controller to be one of the best devices in its class, offering wireless OR RS-232 communications, dual XR relay expansion ports, dual Ultra Ports, an LCD Display Port, Software Controlled LED Backlight Connector, and Integrated I/O Port with Pull Up/Down Resistors.

LED Status Lights

Heartbeat/Valid Data LED

When in RF Mode, this LED Flashes when Valid Data Packets are Decoded
In RS232 Mode, this LED Flashes all the time, indicating the processor is cycling properly.

Power LED

The Power LED Lights when the Processor has Booted Properly.

Enabled LED

The ENABLED LED Lights when the Device is Network Enabled. By Default, this LED is ON. This LED goes off when the device is disabled using E3C Networking Commands (RS-232) or Key Enabled/Disabled Networking Commands (wireless). If this LED is ON, the device is ready to process your commands. If this LED is off, this device will only process networking commands. The appropriate networking command must enable this device (Lighting the LED) before commands will be processed.

Send Data LED

This LED Lights Anytime Data is Sent Back to the Computer.

Jumpers

RS-232 OUTPUT JUMPER

Set to RS232 When One of these controllers is Connected to the RS232 Port of a Computer. Set to OC232 When 2 or More of these Controllers are Connected to the RS232 Port of a Computer. An RSB Serial Booster is REQUIRED When Connecting Multiple Devices to One Serial Port.

INPUT PULL JUMPERS

These Jumpers Pull All Data Lines on an I/O Port Up or Down with a 10K Resistor. This Helps Keep the Inputs Quiet. More on this later in the manual.

XR Relay Expansion Port

Relay Banks are Connected to this port. This Port ONLY Address ODD Relay banks. For instance, All Relay Banks Connected to this port will be addressed as Banks 1,3,5,7,9, etc.

+12VDC Voltage Input to Power the Controller

Ground Power Supply Ground

RS232 Ground Connects to Ground on the RS232 Port

RS232 INPUT Connects to the RS232 Output Line of your PC

RS232 OUTPUT Connects to the RS232 Input Line of your PC

Ultra Port B

I/O Expansion Port Adds Capabilities to the controller.

Ultra Port B Data Bus

Directly Connect to the I/O Data Port of Ultra Port B

+5VDC
IO7 IO6 IO5 IO4 IO3 IO2 IO1 IO0 Ground

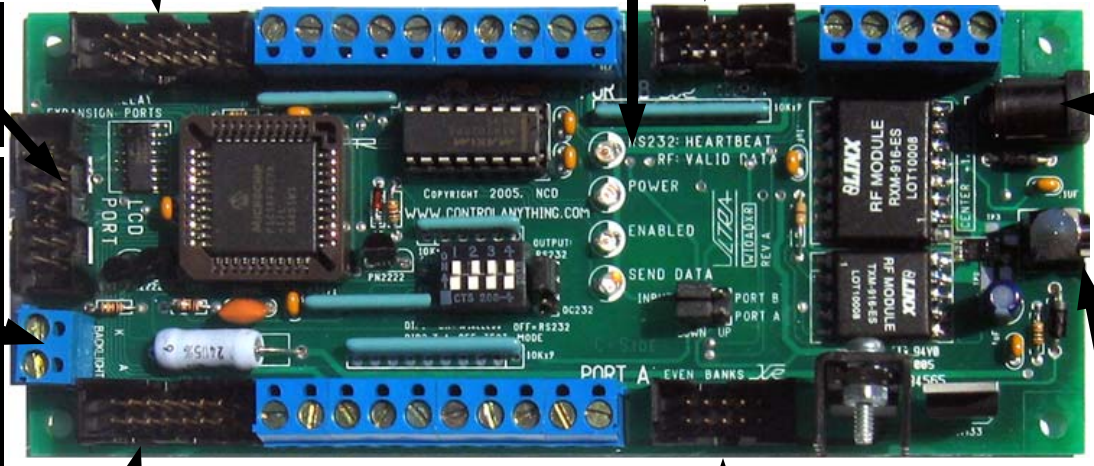
+12 VDC INPUT
GROUND
RS232 GND
RS232 INPUT
RS232 OUTPUT

LCD Port

Add a 40x2 Character LCD Display using this Data Port

LCD LED

Connects to the LED Backlight of a 40x2 LCD and allows the controller to set the brightness of level of the LED.
K = Ground (black wire)
A = V+ LED (red wire)



2.1mm Center Positive +12VDC Power Connector

Ultra Port A

I/O Expansion Port Adds Capabilities to the controller.

+5VDC
IO7 AD7
IO6 AD6
IO5 AD5
IO4 AD4
IO3 AD3
IO2 AD2
IO1 AD1
IO0 AD0
Ground

Ultra Port A Data Bus

Directly Connect to the I/O Data Port of Ultra Port A

This Port Can Also Supports 8 Channels of 8/10-Bit A/D Conversion (Converts 0-5VDC input voltages to numeric values on your computer)

XR Relay Expansion Port

Relay Banks are Connected to this port. This Port ONLY Address EVEN Relay banks. For instance, All Relay Banks Connected to this port will be addressed as Banks 2,4,6,8,10, etc.

RPSMA Antenna Connector for Wireless Communication

DIP Switch Settings

Please see Page 4 for a complete description of the DIP Switch Settings.

+5VDC
OUTPUT ONLY DO NOT APPLY POWER.
This voltage is for you to use, 50ma MAX