Reference Manual &
Programming Guide

WACI NX+   WACI NX Jr.

Revision 2007.01.09a

I.   Quick Contents

II.   Full Contents

III. Introduction

Welcome to the WACI world of powerful, non-proprietary, user-friendly control systems!

All puns aside, Aurora Multimedia has changed the way control systems are expected to behave. The WACI (Web Access Control Interface, pronounced "wacky") is one of the world's first standards-based, non-platform-specific control systems. Not only does it work on just about any IP-based system, but it can be developed on virtually any type of PC as well. The built-in web server, ftp file server, diagnostic tools, and Event Manager allow the WACI to be free of proprietary tools and languages.

In addition to supporting standard languages such as HTML, DHTML, JAVA, FLASH, ASP, and Visual Basic for creating control applications, Aurora Multimedia has provided powerful alternatives for the non-programmer. For one, the built-in Event Manager runs server-side functions from web-based interface without requiring a single line of code to be written. In addition, creating control interfaces with Aurora's Flash-based program, called YIPI (Your IP Interface), is as easy as using a drawing program. Furthermore, since the diagnostics and Event Manager are served up using a standard, non-proprietary web server, any device equipped with a standard web browser, be it PC, Pocket PC, Palm, or Mac, is now enabled to access your system.

The WACI does all this with a powerful 32 bit engine with 32 bit pipeline memory in a small box that can fit just about anywhere.   The power guarantees that small or large files can be processed without delay in a real-time application with network traffic. Systems with less power and less memory bog down quickly on server requests, but this is no problem for the WACI, which has enough power to support the smooth performance of attractive, friendly user interfaces as well.

Aurora Multimedia did not stop at the software user interfaces to increase the serviceability and value of the WACI. Other simple but effective innovations include C Type relays (normally open & closed physically available), which allow creative use such as a switcher or to expand the amount of discrete ports; RS-232 transmit from the IR to allow additional control from alternate ports; IR Input for just about any IR remote to be used with the WACI NX; DSP (Digital Signal Processor) for A/D, D/A, Digital Input, and Digital Output functions from a single port; Compact Flash for wireless and memory functionality; NX-PANSION port to add additional modules for capabilities such as video streaming, storage, touch panel interface and much more; and the built in temperature sensor, which allows monitoring of the physical environment.

With the WACI NX+ and WACI NX Jr., Aurora Multimedia takes the control system industry to a new era of easy-to-use, powerful, and cost-effective tools, paving the way for new possibilities in the way systems are operated.

IV. Box Contents

Box Contents for the WACI NX+

12v 15 Watt Supply

International Supply Kit

WACI NX+ Device   Power Adapter   4 IR Port Emitters

Box Contents for the WACI NX Jr.

12v 15 Watt Supply

International Supply Kit

WACI NX Jr Device   Power Adapter

V.   Hardware Specifications

General Specifications

		WACI NX Jr.	WACI NX+	Notes
Size	5.8" H x 4.9" W x 0.9" D	√	√	
Weight	0.6 lbs.	√	√	
Power Adapter	12V DC	15watt	15watt	
Processor	32-Bit	√	√	750 MIPS Normal Mode 1 BIPS Super Mode
Co-Processing	DSP		√	
	Cipher (Safenet)	√	√	Supports DES, 3DES, AES, ARC-4, SHA-1, MD5 Implements entire IPsec packet processing in hardware – no CPU burden True Random Number Generator (RNG) in hardware Robust security is ideal for most business network environments – meets U.S. government and banking industry requirements
	IR (True Trigger)	√	√	Highly Accurate IR Capture and reproduction
Memory	Total RAM	32M	64M	500DDR 32 bit wide
	Total Flash	16M	32M	
	Available Flash	5M	20M	Available Flash Memory may vary, based on firmware version.

Display	106 x 56 Pixels Backlit	√	√	
Ports	LAN Network Adapter (RJ-45 / Ethernet)	2	2	10/100MBits Auto MDX POE add-on available
	USB OTG (On-The-Go) v1.1	1	1	11MBits Host or Client
	RS-232/422/485 (DB-9)	2	2	15KV Protection, 115KB Max
	Infrared Outputs (IR)	2	4	30KHz – 2MHz RS-232 TX as well
	Infrared Input	1	1	IR input device triggers
	DSP (4 modes of Operation) Analog to Digital Input (A/D) Digital to Analog (D/A) Digital Input Digital Output		4	12-Bit Accuracy 16-Bit Accuracy High Impedance 200ma Sink Open Drain
	Relay (SPDT)		4	C-Type (NO, NC, CP)
Monitoring	32-Bit Internal Clock/Calendar	√	√	
	IR Learner	√	√	30KHz – 2MHz
	Internal Temp Sensor	√	√	
	Beeper	√	√	
Expansion	CF Type II	√	√	Memory and wireless
	NX_PANSION Bus 32bit	√	√	NX add-on modules
Network Access	Web & FTP	√	√	

Specifications subject to change without notice.

Absolute Maximum Ratings for Hardware

Below is a summary of the minimum, maximum, and typical values for the hardware.

These are maximum ratings only. Operation of the device at or above these ratings for extended periods of time may affect reliability.

	Min	Typ	Max	Units	Notes
WACI Input Voltage	+8	+12	+18	V	WACI NX: Min. 15watt supply
Storage Temperature	-20		+55	°C	
Digital I/O Input	-5		5	V	
Digital I/O Output	0		5	V	Can source up to 20mA at 5V
A/D Input	-5		5	V	
Relay	0		2	A	Max 30V DC

Hardware Parts Overview

WACI NX Jr. Parts Overview
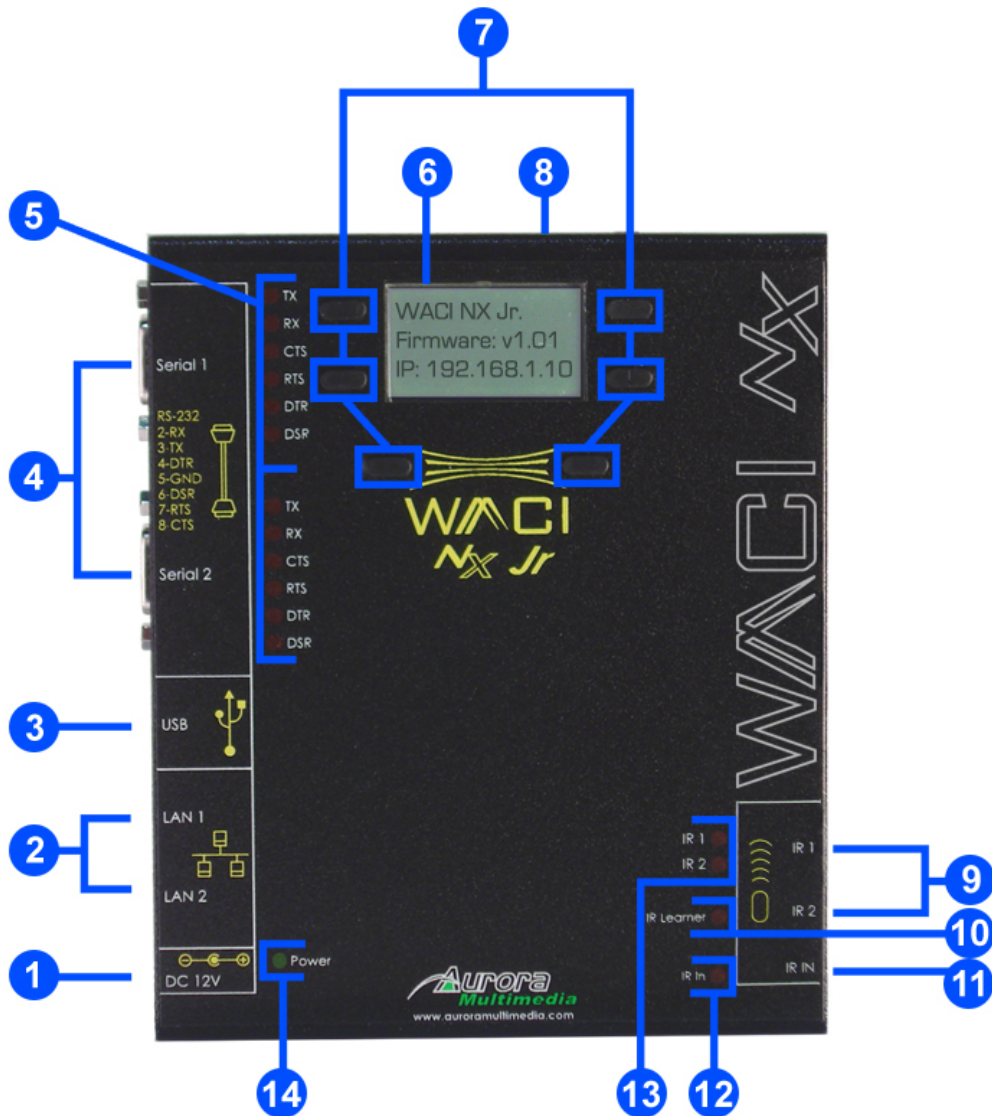
#	Shorts Description	More Info
1	Power Adapter Port	
2	Two 10/100 Auto-MDX LAN Ports	
3	USB 1.1 On-The-Go Port	
4	Two RS-232/422/485 Serial Ports	
5	LED Indicators for Serial Ports 1 & 2	
6	Built-in LCD display for setup and diagnostics	
7	Six button menu navigation	

8	Type 2 Compact Flash expansion port	
9	IR Ports 1-2	
10	IR Learner Port	
11	IR In Port	
12	IR In LED	
13	LED Indicators for IR Ports 1-2	
14	LED Power Status	

WACI NX+ Parts Overview
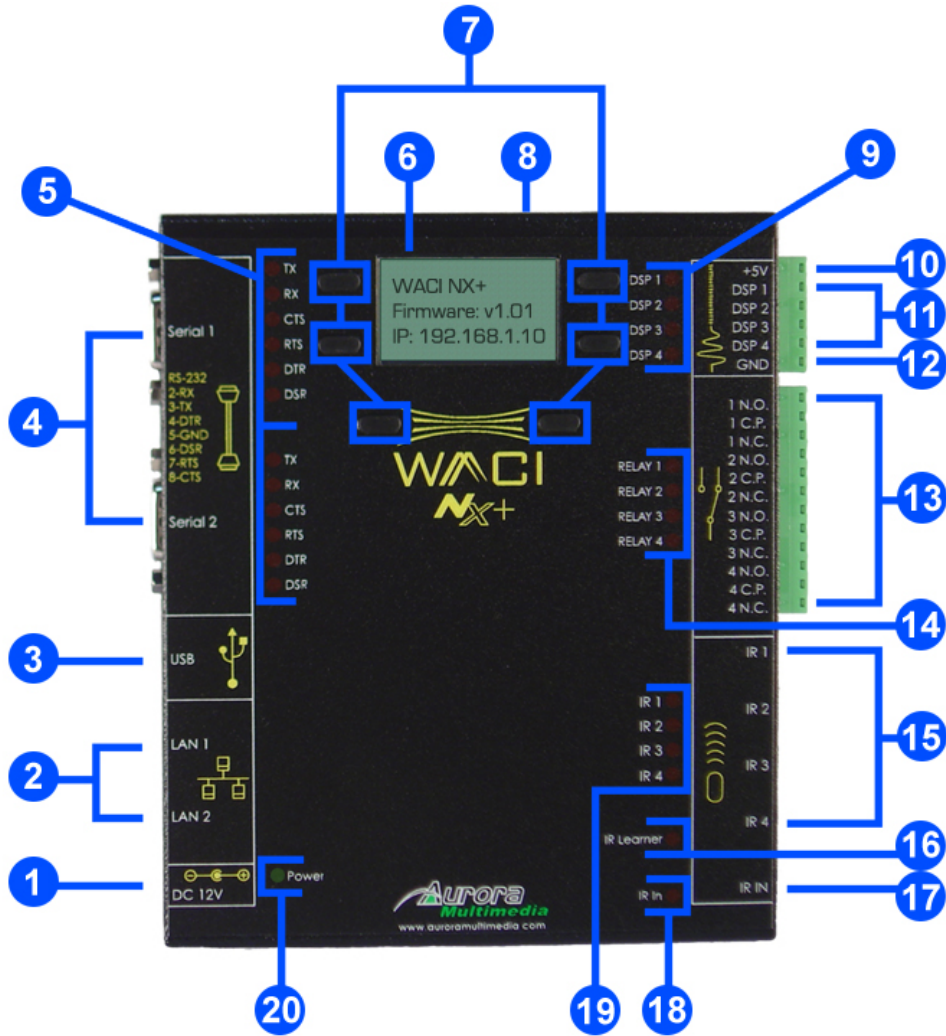
#	Short Description	More Info
1	Power Adapter Port	
2	Two 10/100 Auto-MDX LAN Ports	
3	USB 1.1 On-The-Go Port	
4	Two RS-232/422/485 Serial Ports	
5	LED Indicators for Serial Ports 1 & 2	
6	Built-in LCD display for setup and diagnostics	
7	Six button menu navigation	
8	Type 2 Compact Flash expansion port	
9	LED Indicators for DSP 1-4	
10	DSP +5V Line	
11	DSP Ports 1-4	

12	DSP Ground Line	
13	Relay Ports 1-4 (Normally Open, Center Pin, Normally Closed)	
14	LED Indicators for Relays Ports 1-4	
15	IR Ports 1-4	
16	IR Learner Port	
17	IR In Port	
18	IR In LED	
19	LED Indicators for IR Ports 1-4	
20	LED Power Status	

Detailed Hardware Descriptions

Power Indicator and LCD

A green power indicator is available on both the WACI NX+ and WACI NX Jr. The power indicator is lit whenever power is applied to the WACI. The LED will stay solid if the system is ok. If a problem occurs the advanced internal diagnostics will flash the LED to indicate the type of problem occurring. If the LED is blink the WACI NX may need to be sent in for repair.

Power Adapter

The power adapter is included with the WACI NX. The 15 watts supplied is more then enough to power the WACI NX as well as certain NX-PANSION add-on modules. Check with the NX_PANSION module to make certain a higher wattage supply is not needed.

Power Port

The WACI NX is powered using the included Power Adapter. The smaller end of the Power Adapter is plugged into this port. You should plug the Power Adapter into this port after the network cable has been attached.

Green Power/Status LED

The green LED indicates that the WACI is powered and operating.

LCD Display

The 106 x 56 backlit pixel based display helps with the diagnostic and setup of the WACI NX.

Access Boot Menu

Press and hold the any of the 6 buttons for 5 seconds to bring up the Boot Menu while plugging in the power connector. For more information on the Boot Menu see Boot Menu Reference on page 34. A null RS-232 cable will be needed for the boot menu to a PC running a terminal program.

Serial Ports

Serial port interfaces are available on both the WACI NX+ and WACI NX Jr.

Serial Ports 1 & 2

The WACI NX has two (2) serial ports that enable RS-232 and RS-422/485 communications, supporting a wide range of compatible electronics. More serial ports can be added with the NX-PAND module.

The actual connectors on the WACI are DB-9 male ports.

The device or cable you wish to connect to the WACI NX should be a DB-9 female connector.

Serial Port Pin-Out Diagram
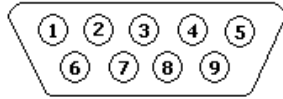
Port	RS-232	RS-422/485
1	NC	RX+
2	RX	
3	TX	
4	DTR	TX-
5	GND	GND
6	DSR	
7	RTS	TX+
8	CTS	
9	NC	RX-

LED Indicators for Serial Ports 1 & 2

There are two (2) sets of LEDs, one set for each serial port. Each set consists of six (6) red LEDs that indicate activity on the pins.

When using RS-232 communications, the six LEDs indicate activity on the (from top to bottom) TX, RX, CTS, RTS, DTR, and DSR pins.

When using RS-422/485 communications, the LEDs indicate activity on the corresponding TX+, TX-, RX+, and RX- pins.

RS-422 Operation:

When using RS-422 operation the RX+ and RX- should have a 120 Ohm resistor across the two pins. Also, RS-422 operation must be selected in the diagnostics page in order for it to properly work.

RS-485 Operation:

When using the RS-485 operation the RX+ and TX+ should be tied together as well as the RX- and TX-. There should be a 120 Ohm termination resistor across the two pins. Also, RS-485 operation must be selected in the diagnostics page in order for it to properly work.

Network Interface

The WACI NX has two 10/100 Auto MDX Ethernet ports. They are also POE (power over Ethernet) capable with the NX-POE add-on module. The Auto MDX allows the use of null or straight through LAN cables to the ports.

LAN Port

The WACI may be connected to a network using a standard RJ-45 Ethernet cable in this port. Both 10Mb and 100Mb connections are supported.

Port	RJ-45
1	TX+
2	TX-
3	RX+
4	NC
5	NC
6	RX-
7	NC
8	NC
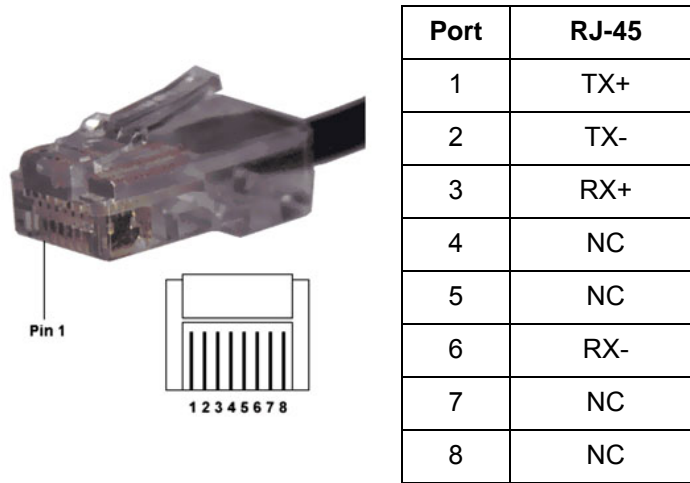
LED Indicators

The green LED on the LAN port indicates that a network has been detected.

The yellow LED on the LAN port indicates that data is being transmitted or received through the port.

Relays (WACI NX+ Only)

The Relays on the WACI NX+ have two connection types: N.O. = Normally Open, and N.C. = Normally Closed. The "Normal" position is the state of the relay when it is not turned on (not energized). You can turn the relays on and off using the relay diagnostic page (see Relay Diagnostics, page 51), using an Event Manager "Relay Action" (see Relay, page 73), or through an RPC method call (see Relay Methods (WACI NX+ Only), page 117). The Relay ports are available on the WACI NX+ only.

Relay Ports 1-4

Each port consists of 3 lines: NO (Normally Open), CP (Center Pole), NC (Normally Closed). Connect the incoming wire to the CP connection, and the outgoing wire to either of the other two ports (NO or NC). The outgoing port you choose depends on which type of connection you wish to consider "Normal".

LED Indicators for Relay Ports 1-4

4 red LEDs indicate activity for each Relay Port.

DSP (WACI NX+ Only)

The WACI NX+ is the first of its kind to have a DSP coprocessor. Each DSP port is capable of being used in one of four different configurations; A/D, D/A, Digital Input, and Digital Output. These modes of operation can be configured in the built in setup pages of the WACI NX or via RPC commands.

A/D (Analog to Digital) Converter

The 12 bit A/D Converter can detect a range of voltage levels and convert them to a numeric value. Therefore, the A/D Converter may be used for applications where states or commands are indicated by variable voltage levels. A voltage of -5v applied to the port will produce a value of 0 when the port is read. A voltage of 5v will generate a value of 4096. Voltage levels between -5v and 5v will produce a proportional numeric value..

You can read the input from an A/D port using the DSP Diagnostics (see DSP Diagnostics, page 52), or using an RPC call (see AD_ReadDigital (Port), page 122).

D/A (Digital to Analog) Converter

The 16 bit D/A Converter can output a range of voltage levels. Therefore, the D/A Converter may be used for applications where varying output voltage ranges of -5v to +5v are required. The port will produce a value of -5v when the port is set to 0 and 5v will with a value of 65536. Voltage levels between -5v and 5v can be sent by using a proportional numeric value.

You can set the output from a D/A port using the DSP Diagnostics (see DSP Diagnostics, page 52), or using an RPC call.

Digital Input

The Digital Input is used to read a digital voltage level. It can read 0v (=0) or 5v (=1). The digital input I perfect for applications requiring simple on/off detection such a contact switch. By default the port is high impedance for the most flexibility. Using a resistor (10k for example) tied from the port to the 5v will create a pull-up configuration. Doing the same but to ground will create a pull-down configuration. If the port is left as is then the device connected to the digital input must supply the high and low voltages.

You can see the state of the Digital Input port using the DIO diagnostic page (see DSP Diagnostics, page 52) or through an RPC method call (see Digital I/O Methods (WACI NX+ Only), page 119).

Digital Output

The Digital Output is used as an open drain and can sink up to 200ma. Setting the port to 0 will leave sink the input to ground. Setting the port as a 1 will put the port into a high impedance state. The digital

output can be used for triggering relays, LEDs, or any device that will not require more then 200ma of current.

You can change the state of the Digital Output port using the DIO diagnostic page (see DSP Diagnostics, page 52) or through an RPC method call (see Digital I/O Methods (WACI NX+ Only), page 119).

DSP +5v Line

Provides a +5V power source. Use the +5V line as a reference, or to power small devices that feed back into the ports.

DSP GROUND Line

To operate properly, the device connected to the input port must be connected to both the GND line of the connector block as well as one of the four input ports.

LED Indicators for DSP Ports (1-4)

The LEDs indicate that the voltage on the port. The brightness of the LED is proportional to the voltage level applied to the port. In the A/D and D/A modes the LEDs will vary their intensity while digital I/O mode will be either on or off.

IR Ports

The four (4) IR ports on the WACI NX+ and (2) IR ports on the WACI NX Jr. are used to send infrared commands to devices that can be controlled using a standard IR remote control. To control a device, connect one of the IR emitters to the IR sensor of the device to be controlled. In addition the IR port can be used a one way RS-232C (TX only) at rates up to 115k.

To program the WACI NX+ to send IR commands, you'll need to download one of the device appropriate .WIR files, or use the IR Learner to learn the remote for the device.

Use the IR Diagnostics page (see Managing Learned Commands, page 49), an IR Action (see IR Port, page 74), or an RPC command (see IR_SendCommand (Port, Group, Command), page 126) to send IR commands out the IR ports.

IR Emitters

The IR Emitters are small cables with an IR Emitter at one end and a small audio style plug at the other. The emitters plug into the IR ports. The emitter end attaches to the device to be controlled. When the WACI is sending a command to the device through the emitter, the LED on the back of the emitter will light up.

IR Ports

There are four IR ports on the WACI NX+ and 2 on the WACI NX Jr. They are located across the Ethernet ports. Plug the IR Emitters into these ports.

LED Indicators for IR Ports

When an IR command is sent out an IR Port, the LED for that port is lit.

IR Learner

The IR Learner is used to learn the IR codes from a device's remote control. It can read many different types of remotes. If it has trouble with your remote, you may need to download from Aurora Multimedia a .WIR file that is specific to the device.

IR Learner Port

Point your remote at the window on the lower right front of the WACI NX to learn an IR command for your remote. You'll need to use the IR diagnostics page to learn the command (see Learn IR Command, page 47).

IR Learner LED

The LED is lit during the learning process when an IR signal is detected on the IR sensor within the WACI NX. When the process is first started the LED will glow in and out slowly until a signal is received. If the remote is too far away it will continue to glow slowly. The correct distance for the strength of the signal is usually about 6" and will turn the LED solid when the proper distance is achieved. If the remote is too close then the LED will blink fast.

VI. Expansion Hardware

The WACI NX control system has the ability to expand (NX-PANSION modules) its capabilities using the docking connector located on the bottom of the unit. This expansion bus is a 32 bit wide high speed connection to pass through small or large amounts of data for advanced applications like video streaming, additional ports, touch panel interfacing, and much more. Up to two expansion modules can be dock to any one WACI NX. Always check with the latest specification of the expansion model to see feature, power and docking requirements. Specifications are subject to change without notice.

One unit docked with a WACI NX+

NX-PAND

• Four IR/Serial (TX only) ports
• Four Relay Ports
• Four DSP Ports
• 5 Serial Ports

NX-STREAM

- Encodes and Decodes MPEG2 and MPEG4 in high and standard definition
- MJPEG at 1280 x 1024 @ 60Hz
- H.263, H.264 Capability
- Composite Video Input and Output, supports NTSC/PAL/SECAM
- S-Video Input and Output, supports NTSC/PAL/SECAM
- YPBPR Input and Output
- HDMI Input and Output
- SPDIF Input and Output
- Standard 40GB hard drive storage with options up to 120GB
- L/R RCA Audio Pair Input and Output Connections
- Controlled via WACI NX Event Manager and Content Manager
- Create play-lists and automate media delivery

NX-HDD

- Can be ordered with up to 120GB of hard drive storage capacity
- Storage of web pages, audio, video, or almost anything that can be served up
- Local data logging storage
- Content handling via WACI NX Event Manager

NX-BAT

• Charged via power supply or the NX-POE (Power Over Ethernet) module
• Battery Gauge via WACI NX diagnostics screen with e-mail warning of power failure
• Provides a true inline redundant power source for the connected WACI NX control unit tasked with a mission critical application in the event of power interruptions (such as brownouts and blackouts)

NX-POE
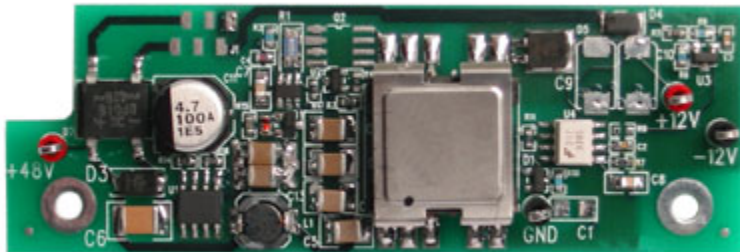
The power over Ethernet module docks inside the WACI NX and not on the expansion bus. This leaves the expansion bus free for additional modules. The NX-POE is docked inside the WACI NX near the LAN connectors on a connector specific for this function.

• Provides 13 watts of power
• Installs seamlessly in the WACI NX as an internal module
• Can be used simultaneously with the WACI NX-BAT and power supply for triple power redundancy
• Optically isolated Power Over Ethernet
• Great when no standard power outlets are available

VII. Factory Default Configurations

Default Network Settings for LAN 1 & 2

DHCP Enabled	Yes
IP Address	From DHCP
Subnet Mask	From DHCP
Gateway	From DHCP
Host Name	WACI
Password	admin

Default Serial Settings

Baud rate	9600 baud
Data bits	8
Parity bits	None
Stop Bits	1
Signal Level	RS232
Handshaking	None

Default DSP Settings (WACI NX+ Only)

DSP Port	Digital Input

VIII. Quick Start

STEP 1: Powering Up

Locate the power adapter that came with your WACI. Plug the small end into the power supply port (labeled "Power") on the lower left side of the WACI, and the large plug into a wall socket or compatible power supply.
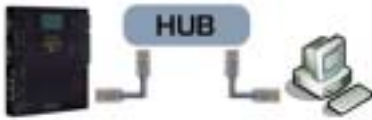
You should observe the following behavior in this order:

- Once power is applied the green Power LED will immediately light as well as the LCD backlight. It will remain lit while the WACI is plugged in. All other LEDs will be off and the LCD will have a message "initializing".

- The WACI will run through its internal hardware diagnostics. If an error occurs, the power LED will flash an error code or display it on the LCD screen (see Error Codes, page 177).

- After approximately 5 seconds (assuming the self-test passes), on the LCD will display firmware version and current IP addresses for the LAN ports. If a LAN port is set to DHCP it may take longer as it will try to get an IP address from the DHCP server. Static IP addresses will allow the WACI NX to boot within 5 sec.

Repeating multiples of fast flashes on the power LED indicate errors (see Error Codes, page 177).

STEP 2: Determining WACI Connection

Decide how you would like to connect to your WACI to configure it:

Connection Type	Why use this connection type?	What do I do next to connect my WACI this way?
Connect the WACI NX to a LAN Network and configure it using a PC on the Network.	This is the typical set up for general WACI NX use.	STEP 3A: Connecting the WACI NX to a LAN
Connect the WACI NX directly to a PC using an Ethernet cable.	You do not have a LAN network, and you need to access all Admin Web Pages for configuration.	SKIP to STEP 3B: Direct Connection if Static IP is Set Ignore step 3A.
Connect the WACI directly to a PC using a serial data transfer cable (AKA null modem cable	You do not have a LAN network, and you only need to access to the basic setup configuration found in the Boot Menu. or You need to set up the manual IP address for a static LAN network	SKIP to STEP 3C: Direct Connection with a Serial Cable Ignore steps 3A & 3B.

STEP 3A: Connecting the WACI NX to a LAN

The WACI can be configured to use either a static IP, or a dynamically allocated one. By default, the WACI is set to DHCP for both LAN 1&2. An IP of 0.0.0.0 will be displayed for a LAN port if a DHCP address is not acquired. (see Default Network Settings, page 24).

If your network is configured to support the default network (IP) settings, you may immediately connect your hub or router to the WACI using an Ethernet cable with an RJ-45 connector.

If your network is setup to use static IP then you'll need to change the WACI's IP address through the Boot Menu (see Boot Menu Reference, page 34).

Some networks use DHCP to automatically issue IP addresses to machines connected to it. If your network is setup to use DHCP, then you'll need to configure the WACI to use DHCP. There are two methods for doing this: 1) press and hold any of the 6 buttons for 5 seconds while plugging in the power for the boot menu 2) go into Setup page of the Admin Web Pages (see Admin Web Pages, page 38).

Once the WACI is configured correctly for your network, connect a standard Ethernet cable from the WACI to either a HUB or router.

STEP 3B: Direct Connection if Static IP is Set

If you wish to connect your computer directly to the WACI, you'll need to use an Ethernet cable (either null or straight through will work). You'll also need to set up your computer to use a static IP address.

Set the PC to an IP address in the same range but different from the WACI. A typical configuration is to manually set the Local Connection on your computer to if the WACI LAN was set to 10.10.10.10:

IP Address	10.10.10.9
Subnet Mask	255.255.255.0
Gateway	10.10.10.1

STEP 3C: Direct Connection with a Serial Cable

You may also configure WACI directly from your computer with a serial connection (see Boot Menu Reference, page 34).

You'll need a Null-Modem cable to connect your computer to the WACI. Run a terminal program on your computer and set the serial communication settings to:

Baud rate	9600 baud
Data bits	8
Parity bits	None
Stop Bits	1
Flow Control	None

Once connected to your computer, press and hold the any of the 6 buttons for 5 seconds while inserting the power connector. Configure your network settings as appropriate.

Once the network settings are configured, connect your WACI as described above in step 3A or 3B.

STEP 4: Access the WACI

Once the WACI is physically connected to the network, and the IP address for the appropriate LAN port is displayed on the front LCD, you may access the WACI through your web browser, FTP client, and other network clients.

The WACI supports the NBNS (NetBIOS Naming Service) for host name resolution. To access the WACI by its host name, make sure your computer has the appropriate protocols loaded. If you are running Windows XP, make sure the NWLink NetBIOS protocol is installed. For Windows 9x, the protocol of interest is NetBEUI. The Apple Macintosh does not ship with NetBIOS support. A third party driver needs to be installed on the Macintosh computer for it to access the WACI via its host name.

You can use the WACI's host name or IP address to:

- Upload User-Defined Web pages to:
  ```
  ftp://[IP Address or Host Name]/wwwpub/
  ```

- View User-Defined Web Pages:
  ```
  http://[IP Address or Host Name]/
  ```

- Access Admin Web Pages (Setup, Diagnostics, etc) at:
  ```
  http://[IP Address or Host Name]/setup/
  ```

Some examples are:

```
http://10.10.10.10/setup/
http://waci/setup/
http://waci/
ftp://user:admin@waci/wwwpub/
```

STEP 5: Configure WACI Using the Admin Web Pages

Once the WACI is connected to the network, you may configure the device through the admin web pages at:

```
http://[IP Address or Host Name]/setup/
```

Click "Setup" icon for network and other basic settings. Details describing these options can be found in the section Admin Web Pages, starting on page 38.

IX. Using the WACI

This section summarizes common administrative tasks for running your WACI and refers you to appropriate sections of this manual for further details.

Testing Hardware

From the Boot Menu (see Boot Menu Reference, page 34), you may:

- Test all the LEDs (see 6: Lamp Test, page 36).

- Check your RAM (see 7: Hardware Test, page 36).

From the Diagnostic web pages (see Diagnostics, page 45), you may check the:

- IR Emitter Ports (see System Information

System information is available for the WACI NX+ and WACI NX Jr. It shows available memory, current network settings, IR Port, Serial Port, Relay Port, and DSP Port settings.

By clicking "System Reset" button, you can remotely restart WACI. It is useful after making changes to some settings or modifying EventManager file.

- Infrared , page 47).

- Serial Ports (see Serial Port Diagnostics, page 49).

- Relay Ports (WACI NX+ only) (see Relay Diagnostics, page 51).

- DSP Ports (WACI NX+ only) (see DSP Diagnostics, page 52).

- Total Memory (see **Error! Reference source not found.**, page **Error! Bookmark not defined.**).

- System Logs (see Log Files, page 52).

Uploading Firmware Upgrades

Your WACI's firmware may be upgraded by uploading a firmware upgrade file provided by Aurora Multimedia. Firmware upgrade files will have an extension of `.AFW` (operating system upgrades) or `.ABT` (bootstrap loader upgrades). These files may be uploaded to the WACI after it is set to firmware upgrade mode. The WACI may be set to firmware upgrade mode through the serial Boot Menu (see 5: Download New Firmware, page 36) or the Admin Web Pages (see Firmware, page 54).

Note: Firmware can be uploaded only one file at a time, and the WACI cannot be used for any other purpose while in the upgrade mode.

Viewing Server Logs

Logs for the RPC, Web, and FTP servers may be monitored in the Logs section (see Log Files, page 52) of the Admin Web Pages:

- RPC Server Info (see RPC Server Info, page 53) tells you the version of the RPC server and the available methods.

- The RPC Server Commands log (see RPC Server Commands, page 53) allows you to see all HTTP Post calls made to the RPC server. It is typically used for debugging custom applications and custom web pages.

- The Event Manager Log logs information about the status and execution of Events and Actions. Errors generated during the execution of the Events and Actions are also written to this log.

- The log for FTP Server Connections and Commands (see FTP Server Connections & Commands, page 53) allows you to monitor activity on the WACI's FTP server. It is typically used to examine user access and determine network needs.

30

- The log for the Web Server (see Web Server, page 53) allows you to monitor activity on the WACI's Web server. It is typically used to examine user access, determine network needs, and debug custom web pages.

Automating Tasks (The Event Manager)

The WACI has built-in, self-monitoring capabilities that will allow you to automate many tasks without developing web pages or applications. The WACI can be configured to watch for specified states, or "Events", such as a time of day or a value on a port. At the time an Event is triggered, the WACI may then perform some specified tasks, or "Actions".

The options for these capabilities may be configured through the Event Manager page (see The Event Manager, page 60), in the Admin Web Pages.

Uploading Custom Web Pages

If you have frequently used tasks or want to facilitate administration of tasks, you may decide to develop your own custom web pages. Typical reasons for custom web page development:

- Streamlining operations, performing tasks efficiently.

- Custom graphical user interface is required.

- Reducing training costs for employees who operate the system controlled by the WACI.

- Easy network access to interfaces is required.

- Facilitation of documentation distribution.

Once your custom web files are created, and your WACI is live on your network, the files may be uploaded with any standard FTP client to:

```
ftp://[IP Address or Host Name]/wwwpub/
```

These custom web pages may be viewed with a web browser at:

```
http://[IP Address or Host Name]/
```

For more information about the FTP server and uploading files, see the Files section (page 55) in the Admin Web Pages documentation.

Using Remote Procedure Calls (RPCs)

In the development of your custom web pages or application, you may wish to get a state from the WACI or command it to perform some task. The WACI supports many programmable functions through remote procedure calls (RPCs). For full details, see (see RPC Server Commands, page 53).

X.   Web Server Features

One of the greatest benefits of the WACI is its ability to serve up custom web pages that you upload to the device.

The WACI has built-in support for the following web features:

Content Type	Notes
Active Server Pages	.ASP File Extension
Hyper Text Markup Language	.HTML File Extension
Flash Programs	.SWF File Extension
Java Script	.JS File Extension
Server-Side Includes	For example, use <!--#include file="MyInclude.txt"--> within your .ASP page or .HTML page.
VB Script	Embedded into the .ASP pages
FTP Sever	Access using an FTP client
RPC Server	Access using HTML post

In addition, the WACI can store virtually any type of file content (see Files, page 55).

XI. Boot Menu Reference

Accessing the Boot Menu with a Serial Connection

Connect the Serial 1 Port of the WACI to your computer with an RS-232 Data Transfer Cable (also known as a Null Modem Cable). Using Hyper Terminal (or similar software), connect to the WACI with the default Serial Settings: Baud Rate: 9600, Data bits: 8, Parity: None, Stop bits: 1, and Flow control: None.

Press & hold any of the 6 buttons while inserting the power connector. The WACI Boot Menu will appear on the terminal screen:

```
WACI Boot Menu
-----------------------------
0 : Display Current Settings
1 : Restore Factory Defaults
2 : Configure IP Settings
3 : Set Password
4 : Set Host Name
5 : Download New Firmware
6 : Lamp Test
7 : Hardware Test
8 : Erase All Files
9 : OEM Functions
x : Exit menu and boot

Selection:
```

The boot menu will prompt you through setup and diagnostics.

Details about the options follow in this section.

0: Display Current Settings

Selecting "0" will display the following as an example:

> Ethernet Port 1 Settings
> MAC address : [00-11-02-02-00-03]
> Host Name   : [WACINX1]
> DHCP Enabled : Yes
> IP address   : [Automatic]
> Subnet Mask : [Automatic]

Gateway   : [Automatic]
Ethernet Port 2 Settings
MAC address : [00-11-02-01-00-03]
Host Name   : [WACINX2]
DHCP Enabled : Yes
IP address   : [Automatic]
Subnet Mask : [Automatic]
Gateway   : [Automatic]
Password   : [admin]
OS Loaded   : Yes
OS Filename : nk.bin

<Press Enter>

1: Restore Factory Defaults

This option restores the WACI's network settings to the factory defaults (see Factory Default Configurations, page 24).

2: Configure IP Settings

Use to enable DHCP or manually set an IP address, subnet mask, and gateway IP address.

Configure which network card (1 or 2)? 1
Enable DHCP (Y/N)?N
Enter new IP address: 192.168.3.165
Enter new Subnet mask: 255.255.0.0
Enter Gateway IP address: 192.168.1.1
Successfully changed IP settings.
<Press Enter>

Configure which network card (1 or 2)? 1
Enable DHCP (Y/N)?Y
Successfully changed IP settings.
<Press Enter>

3: Set Password

The maximum length is 8 characters, numbers and letters only. (If you are prompted for a user name as well as a password, you may use any name or leave it blank.

4: Set Host Name

The maximum length is 16 characters, numbers and letters only. Once set, you will be able to access your WACI at the URL `<http://[Host Name]/>`.

5: Download New Firmware

Select "5" to upgrade the OS (`*.AFW`) or bootstrap loader (`*.ABT`) file.

The Boot Menu display will first indicate some network initialization, and then the WACI will prompt you to use a tftp (trivial ftp) client to upload your upgrade file to the WACI at a specific IP address.

At this point, the blue LED will be OFF, and all the Serial LEDs for both ports will be on. You will now need to upload the OS or boot file using a tfpt client. To use the default Windows tftp client, open a Command Prompt (or DOS shell) window and execute:

```
tftp -i [IP Address] put [/File Path/Upgrade File]
```

Example:

```
tftp -i 10.10.10.10 put c:\temp\waciplus20.afw
```

You may only upgrade one file at a time, and you may not do anything else with the WACI in this mode.

During the file transfer, the serial LEDs will blink one at a time, and you will see the status on the boot menu screen.

The WACI will reset on completion. Continue to use WACI as normal, or press & hold the reset button 5 seconds to re-enter boot menu.

Note: The TFTP client is available on Linux, Windows NT, 2000, and XP. Windows 9x, Windows ME, and the Macintosh O/S's don't come with a TFTP client. You'll need to acquire a third-party client to upgrade your WACI using these operating systems. Contact Aurora Multimedia for a list of clients that will work with your O/S.

6: Lamp Test

Turns all LEDs on, so you can verify that they are working correctly.

7: Hardware Test

Performs a memory scan to validate the integrity of the system RAM.

8: Erase All Files

Deletes all files downloaded to the WACI. These are normally the files that have been downloaded using an FTP client to `<ftp://[Host Name]/>`.
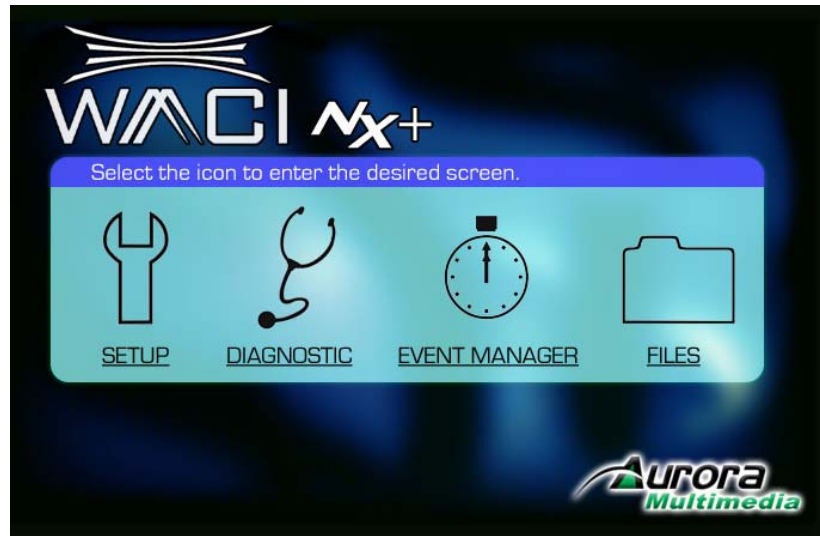
9: OEM Functions

These are factory options that are configured by the manufacturer, and are not applicable for the end-user.

XII. Admin Web Pages

Admin web pages include setup and diagnostic options, and may be accessed at:

```
http://[IP Address or Host Name]/setup/
```


From this web page, you may click on:

- SETUP - for firmware version, date/time setting, network settings, and network security settings.

- DIAGNOSTIC - firmware upgrade instructions, logs, options, and other diagnostic tools for using and/or monitoring WACI hardware.

- EVENT MANAGER - powerful non-programming tool to automate WACI tasks (see The Event Manager, page 60).

- FILES - opens an ftp connection to the WACI.

Setup

This web page allows you to set several administrative options. Many of these options are also available from the Boot Menu.
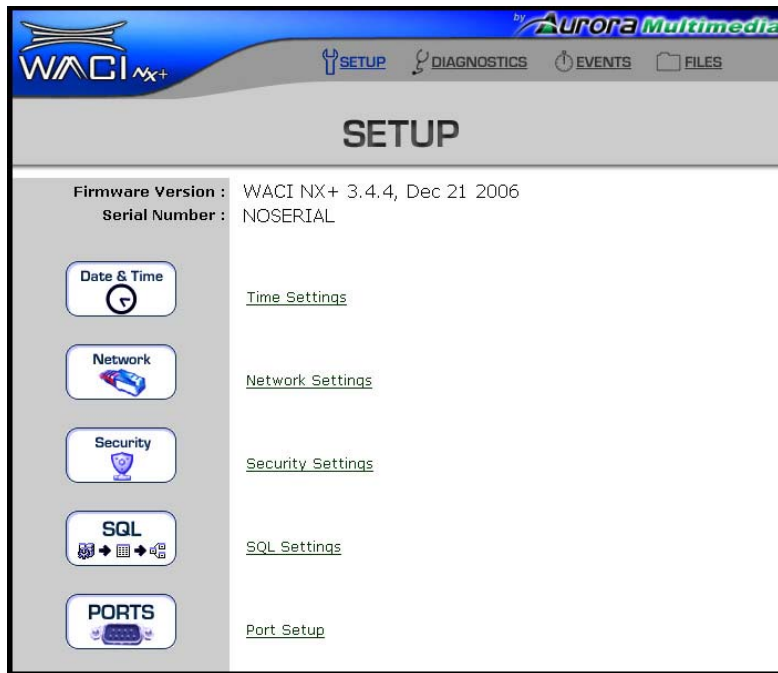
Firmware Version

The firmware version installed on the WACI is displayed.

Date / Time

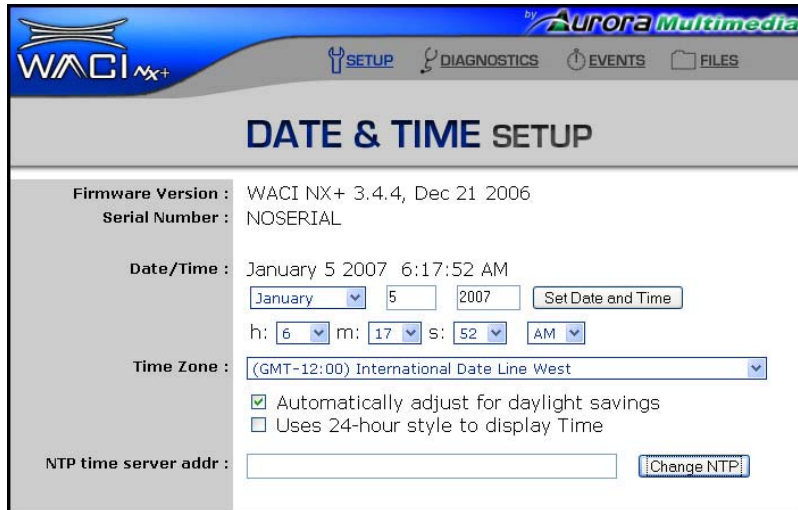The current date and time is displayed. You may also set the date and time using the pull-down menus. Once you have made new selections, use the "Set Date and Time" button next to the pull-downs to save the new date and time.

Time Zone

The time zone setting is used when the WACI communicates with outside computers and servers. Setting the time zone also allows the WACI to appropriately apply corrections for Daylight Savings Time. You can set the active time zone by simply choosing one from the list of time zones.

Check the "Automatically adjust for Daylight Savings" check box to have the clock automatically adjusted when the system enters or leaves Daylight Savings Time.
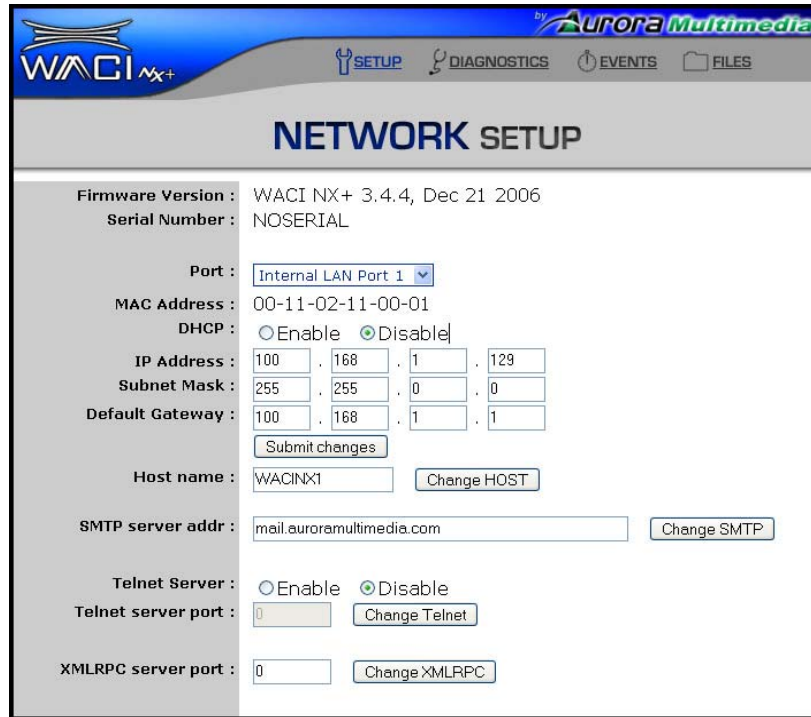
MAC Address

The WACI Ethernet adapter's MAC address is displayed to the right of the MAC Address label. You can use this to validate the IP address assigned in the DHCP server's address table.

Network Settings

If DHCP is enabled for the WACI, the IP Address, Subnet Mask, and Default Gateway are automatically assigned, and the values will be grey and un-editable.

If the DHCP is disabled, the values for IP Address, Subnet Mask, and Default Gateway will be black and editable.

Host Name

By setting the host name, your WACI 's web server may be accessed at

```
http://[Host Name]/
```

Similarly, the FTP server may be reached at

```
ftp://[Host Name]/
```

SMTP Server Address

By setting the SMTP, your WACI 's will be able to send e-mail notifications.

`[SMTP Name]`

System Password

The password allows you to access the Admin Web Pages and FTP server. No password is required to access web pages downloaded into the wwwpub directory.

IP Access Table

When specified, your WACI will only accept network requests from computers with these IP addresses.
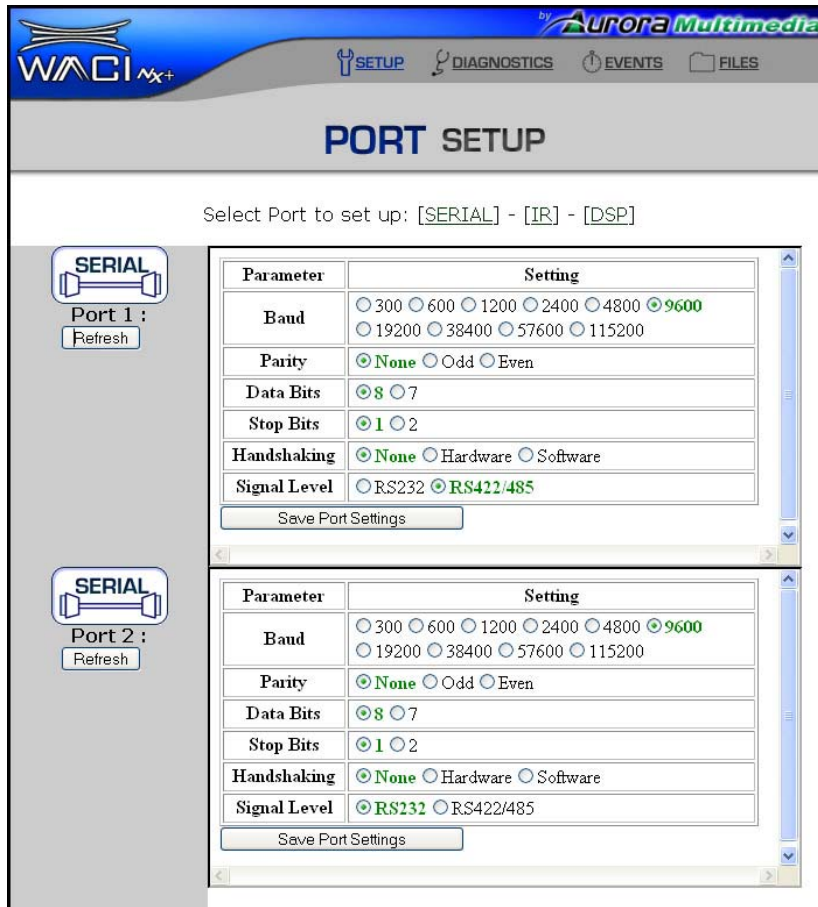
Port Setup

This page allows you to change Serial Port, IR Port, and DSP settings. WACI NX+ has 2 Serial ports, 4 IR ports, and 4 DSP ports, and WACI NX JR has 2 Serial ports, and 2 IR ports.

SQL Setup

Using SQL Setup, users are able to create tables, make queries, and store data in a WACI's database.

Diagnostics

Four of the Diagnostics described in this section (IR, Serial, Memory, Log, and F/W) are available to both the WACI NX Jr. and the WACI NX+. The remaining four Diagnostics (Relay and DSP) are available only for the WACI NX+, which supports the additional hardware.

The Diagnostics web page for the WACI NX Jr. reflects its five supported diagnostics:
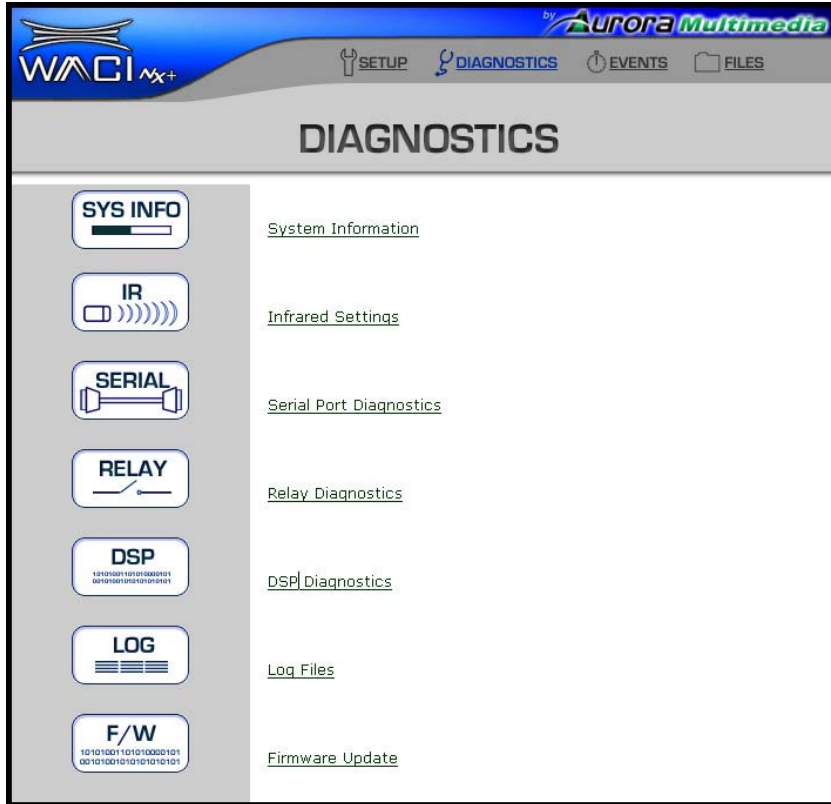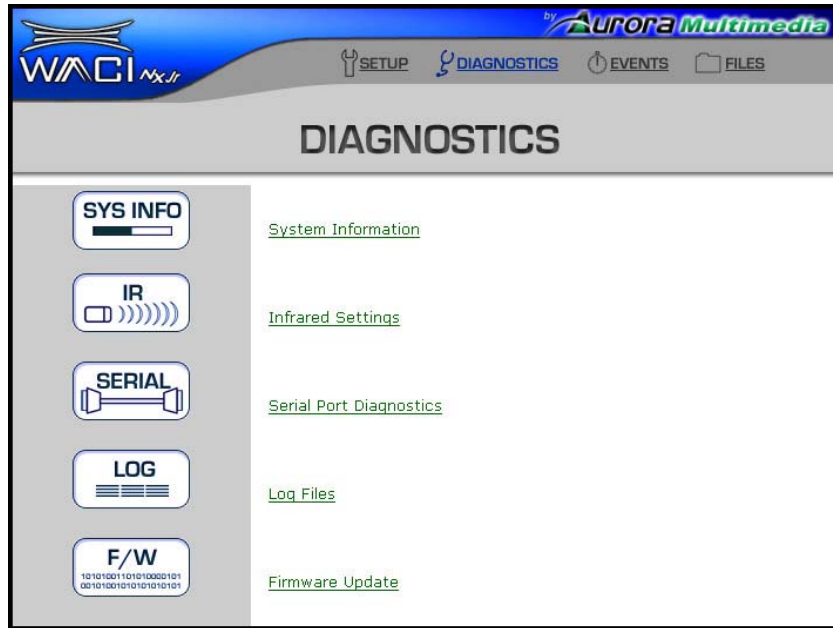
The Diagnostics web page for the WACI NX+ reflects all seven diagnostics:

Clicking on either the button or the link on the web page will take you to the available diagnostic tools.

All the diagnostics are presented in this section in the order found on the WACI NX+ Diagnostics web page.

System Information

System information is available for the WACI NX+ and WACI NX Jr. It shows available memory, current network settings, IR Port, Serial Port, Relay Port, and DSP Port settings.

By clicking "System Reset" button, you can remotely restart WACI. It is useful after making changes to some settings or modifying EventManager file.

Infrared Settings

This page allows you to "teach" your WACI various IR commands for a device (such as a VCR) using the remote control for the device. In addition, a complete list of devices and commands may be reviewed, tested, and/or deleted in the IR Commands section at the bottom of the page.

Learn IR Command

The web page walks you through training the WACI to recognize an IR command:
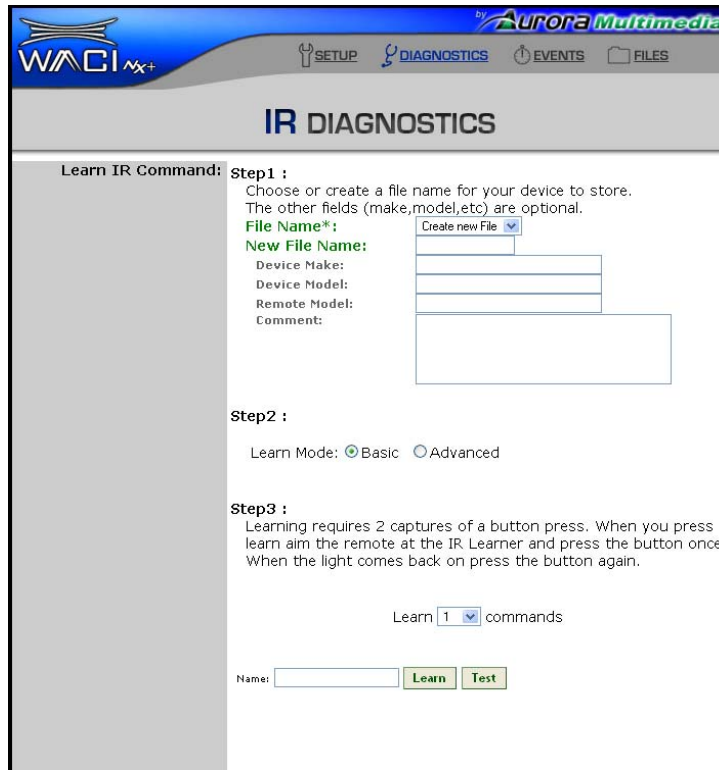
In step 1, choosing a File Name allows you to "group" all of your IR commands by device. For your convenience, you may choose a File Name that indicates which device you have chosen to support (such as "`acmeDVD`" or "`acmeVCR`". You may also enter the Make, Model, Remote Model, and any Comments about your device (this information is optional). The File Name should contain letters and numbers only, no spaces.

In step2 select the learn mode. Basic is a simple sample of the incoming IR. Try this mode first. If it does not work then try advanced mode which will do a more intelligent sampling and filtering.

In step 3, you may choose a name for your Command (such as "`Play`" or "`Eject`"). Use letters and numbers only, no spaces should be included in the name. This Command will be associated with the device / File Name you specified in step 1. There is also a learn x commands where x is the number of commands to be learned. This allows multiple entries of commands to be entered before learning which in turn will speed the process.

Finally look for the IR Learner port located on the front of the WACI. Steadily aim your remote's IR output 6" from the sensors and press the desired remote button you wish the WACI to learn. The IR LED will glow slowly while the WACI is waiting for an IR signal, and turn solid when it detects a signal. If the remote is too close the LED will glow fast. The web display will indicate a successfully learned command or a timeout if nothing was received.

IR Learning Tips

- Hold the remote steady.

- Hold the remote 6" from learner port.

- Do not confuse the learner port with the emitter or IrDA ports.

- Turn off fluorescent lights.

Managing Learned Commands

You may view all learned commands at the bottom of the page:
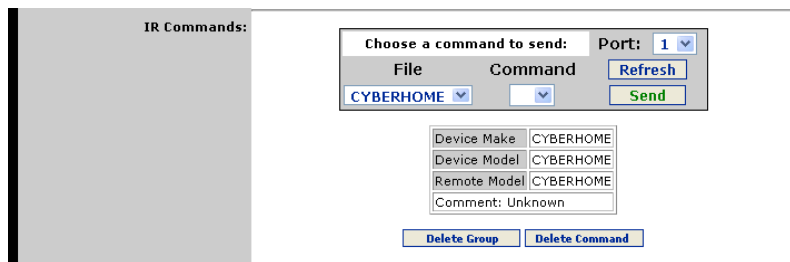
Once a command is selected from the menus, you may send the command out the selected port to the attached IR emitter. When the IR emitter is sending a command, the LED on the back of the emitter will light up.

You may also refresh the file and command lists by selecting "Refresh Lists", delete the selected command by selecting "Delete Command", or delete the selected group (including all its associated commands) by selecting "Delete Group". Depending on your web browser, you may need to scroll down to view all of these buttons.

Serial Port Diagnostics

The serial port diagnostics are available for the WACI NX+ and WACI NX Jr. From this web page, you may access three diagnostics for the serial ports.

Serial Send & Receive Histories

First, you may monitor the send and receive histories from either serial port.
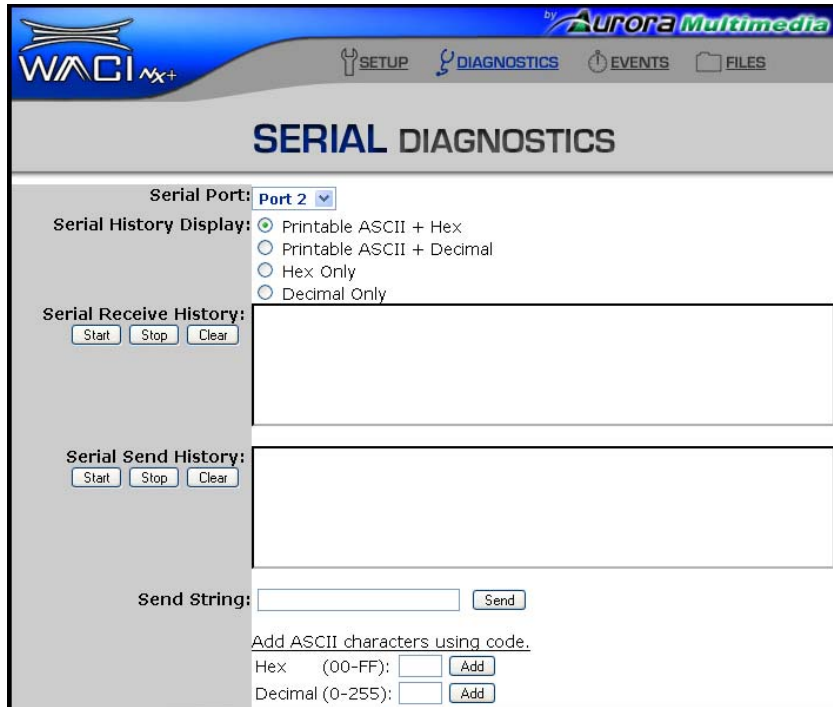
The serial send and receive histories logs the bytes being output or input through the ports. First, select which port to log (1 or 2), and then specify the display. The bytes passing through the port may be displayed as printable ASCII characters, hex values, or decimal values.

If you select ASCII characters, you must also choose how you wish to view non-printable characters. The non-printable characters are denoted by boldface type and "<" and ">" parentheses. For example, an ASCII "space" character, which is non-printable, may be displayed as decimal value **<32>** or hex value **<20>**.

Send String

The second diagnostic allows you to send a byte stream through the port specified at the top of the page. The byte stream you wish to send must be a URL-encoded send string:
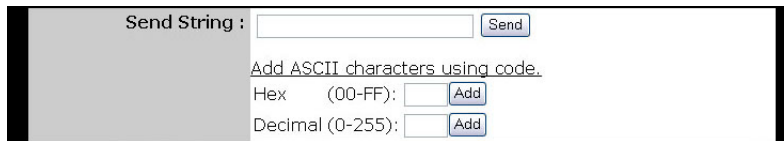
To send a URL-encoded string, follow these three rules for encoding:

1. Enter your byte values as a string of **printable** ASCII characters and hex values. Spaces and returns, for example, are non-printable characters and MUST be converted to hex.

2. Any hex value in the send string must be preceded by a "%" symbol.

3. The URL-encoded string does NOT accept decimal values, so if you only know the decimal

value of a byte, you MUST either convert it to a printable ASCII characters or the hex equivalents.

For example, the decimal byte stream "103 111 111 100 32 100 111 103" is equivalent to the hex byte stream "67 6F 6F 64 20 64 6F 67". It is also equivalent to the ASCII string "`good dog`". However, to send the string to the serial port, you must use a combination of printable ASCII characters and hex values only. Therefore, you may send the string as "`good%20dog`" or "`good%20%64%6F%67`", but not "`good%32dog`" (where "32" is a decimal value). (You do not need to use quotes in the Send String text box.)

For your convenience, if you forget URL-encoding rules 2 and 3 above, you may add hex or decimal values using the hex and decimal fields just below the Send String text box. Adding a hex number to your Send String with the Add tool simply places the "%" prefix in front of the value you enter, with no conversions. Adding a decimal number first converts the decimal number to the hex equivalent and correctly places the "%" prefix in front of it within the send string.

If the Send History log has been started, the send string should appear in that display once the "Send" button is pressed.

Relay Diagnostics

Relay ports are available on the WACI NX+ only. You may test the relays by clicking the "On" radio button for the port you wish to test. The LED for the desired relay port should light, indicating the activation of that port.

DSP Diagnostics

Digital I/O is available on the WACI NX+ only.

Each port may be configured and tested as:

- A/D Analog to Digital (-5v to +5v) 12 bit (0 to 4096)

- D/A Digital to Analog (-5v to +5v) 16 bit (0 to 65536)

- Digital Input (0 to 5v)

- Digital Output (Open Drain sink up to 200ma)

When the state of the port is "High", the LED light will be on otherwise if it is an A/D or D/A the LED will vary intensity based on level.

Log Files

Log files for the WACI servers are available for both the WACI + and the WACI NX Jr. An admin may review or clear (reset) the logs from this page:
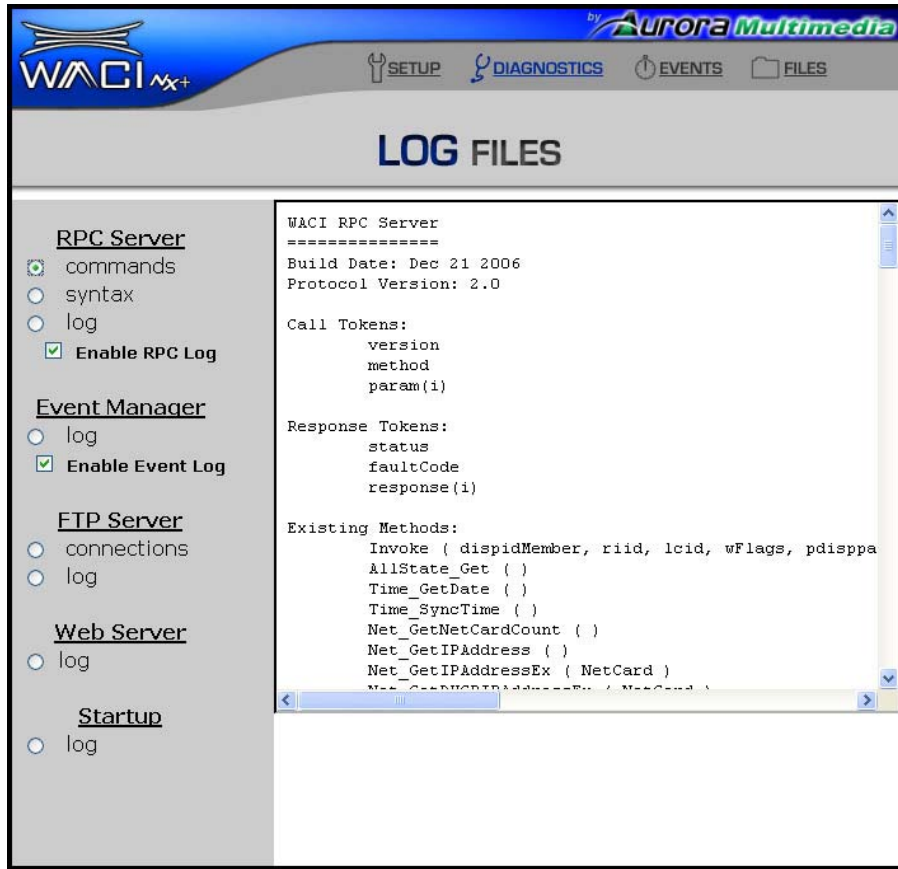
RPC Server Info

Displays the build date and version number of the RPC server. Also displays all tokens and methods available from the server. You may notice that the methods that support the additional ports on the WACI NX+ are not available on the WACI NX Jr.

RPC Server Commands

Logs all HTTP Post calls to the server. Active X calls are not logged. Use this log to monitor calls made to the WACI by your custom web applications. It is quite useful when debugging your application.

Event Manager Log

Logs information about the status and execution of Events and Actions. Errors generated during the execution of the Events and Actions are also written to this log.

FTP Server Connections & Commands

Logs the FTP activity on the WACI.

Web Server

Logs the HTTP activity on the WACI's web server.

Firmware

The Firmware page, available on both the WACI + and the WACI NX Jr., will walk you through a WACI firmware update:

First, you will be prompted for the firmware files. Firmware upgrade files will have either the extension .AFW for operating system upgrades, or .ABT for bootstrap loader upgrades.

Once your upgrade file has been located, the web page will tell you the tftp command line you will need to execute to upload the file into your WACI. It will look something like:

```
tftp -i [IP Address] put [Filename with Full Path]
```

If ActiveX is enabled for your web browser, you may also request that the WACI create a batch file `waciload.bat` that will execute this command. The batch file will be placed at the root of your local drive.

To actually upload the file, click the "Ready to Download Firmware" button. The blue Status LED will turn OFF, and all the Serial LEDs for both ports will be on. You will now need to upload the OS or boot file using a tfpt client.

To use the default Windows tftp client, open a Command Line (or DOS Shell) window and execute the tftp command shown on your web page OR simply execute the batch file. Alternatively, you may double-click on the batch file.

While the file is uploading, the serial LEDs will blink one at a time. Once the file is uploaded, the WACI will reset, and you will be able to access the WACI's web pages again.

Two important notes:

- You will not be able use the WACI for anything else while it is in the firmware upgrade mode.

- You may only update one firmware file at a time. If you wish to update both the boot-strap loader and firmware, you will need to completely upload one first, then repeat the entire process for the other.

Events (See section: The Event Manager)

Details of the Event Manager are described in the section, The Event Manager (page 60).

Files

Selecting the "Files" link or icon will simply take you to the WACI's ftp server:

```
ftp://[IP Address or Host Name]/
```

Typically, selecting this link will display the ftp directory in a browser window, such as Internet Explorer for Windows XP:
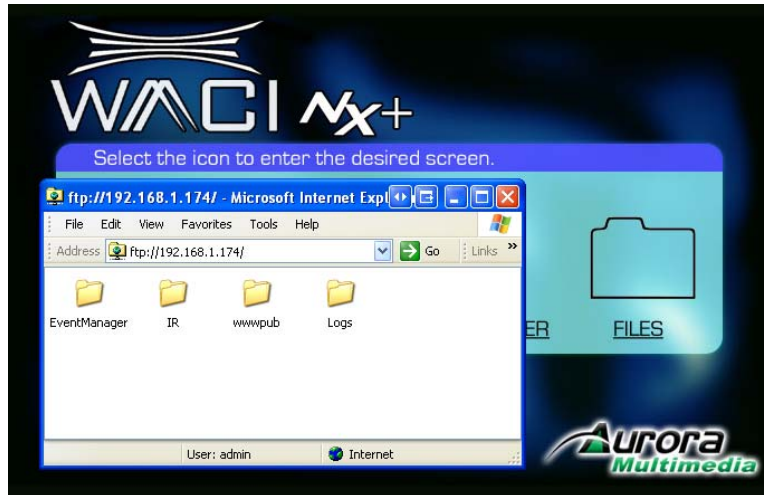
You may be prompted for password, which is the same as the administrative password for the WACI. Any user name, such as "user", is valid as long as the password is correct.
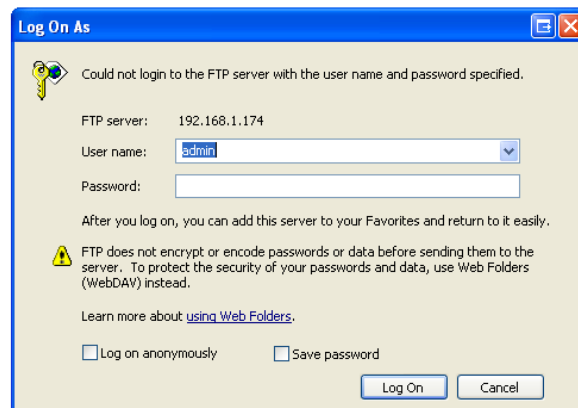
NOTE: While this is a convenient way to access the WACI's FTP directories, any standard FTP client (such WFTP for Windows, Fetch for Macintosh, or "ftp" from a UNIX or DOS command line) may be used to manage files on the WACI's FTP server.

File Structure

The WACI automatically creates three directories at the root level of its FTP server:

- EventManager - holds data for your custom Events, Actions, and Variables.

- IR - holds IR Learning data for any learned commands.

- wwwpub - holds your custom web pages. Most of your ftp activity will happen in this directory.

Note that you may not see the `EventManager` or `IR` directories until you have started using the Event Manager and IR Learning features, as explained in the following subsections.

EventManager

This directory is created when you create a custom Event, Action, or Variable from the Event Manager web page (see The Event Manager Web Page, page 61). All Event Manager data is stored in the file `EventManager.wem` and backed up in the file `EventBackup.wem`. Access to these files are provided for your own convenience, so that you may archive your backups, restore a backup, or create "frequently-used" sets of Event data that may readily be copied to other WACIs.

IR

When the WACI learns one or more IR commands for a device (see Learn IR Command, page 47), all the commands for that device are saved as a group in one file (you choose a name for this Group/File in Step 1 of the IR learning process). These groups of commands are stored as data files in the `IR` directory with the extension `.WIR`.

For example, if you created an `AcmeDVD` group of commands and an `AcmePlasma` group of commands, you should see the files `AcmeDVD.WIR` and `AcmePlasma.WIR` in the `IR` directory.

Access to these files are provided for your own convenience, so that you may backup your IR commands, restore a backup, or easily distribute groups of commands to other WACIs. The `.WIR` files are read by the WACI as it uses them, so moving or updating a file through the FTP directory does not require that you restart the WACI. The `.WIR` file names should contain letters, numbers, and the "_". No spaces should be in the name.

wwwpub

In the event that it is convenient to develop custom web pages to monitor and operate your WACI, the WACI has a built-in web server that will serve up your custom web pages from this directory.

Any custom web pages uploaded to the `wwwpub` directory may be viewed at:

```
http://[IP Address or Host Name]/
```

Allowable File Types

In the `EventManager` directory, the active Event Manager data file is named `EventManager.wem`. This file is read on startup.

Command files for `IR` Learning are stored in the IR directory, and have extension `*.WIR`.

The WACI's built-in web server supports many Internet standards such as HTML, XML, DHTML, Visual Basic, Active Server Pages, Java, JavaScript, server-side includes, and Macromedia Flash. All files supporting these standards (.htm, .html, .gif, .jpg, .asp, .fla, .swf, etc.) may be safely uploaded to the `wwwpub` directory.

In addition, files that are supported by client-side applications (such as Adobe Acrobat Reader, QuickTime, Aladdin Stuffit, etc.) are correctly handled by the WACI's web server. This allows a huge number of file types such as .pdf, .mov, .wav, .sit, and .zip to be easily accessed from custom web pages.

At this time, the WACI does not support PERL, PHP, or built-in databases.

Uploading Files

You may use any FTP client or web browser to manage the files in the WACI's FTP directories.

For example, using Windows XP, you may drag and drop files from the File Explorer to the FTP directory in the Internet Explorer window. First open a connection to the FTP server by selecting "Files" from the Admin Web Page, or by directly navigating to:

```
ftp://[IP Address or Host Name]/
```

You may need to specify a username in the URL for direct navigation. Any username, such as "user", may be used:

```
ftp://user@[IP Address or Host Name]/
```

You will be prompted for password, which is the same as the administrative password for the WACI.
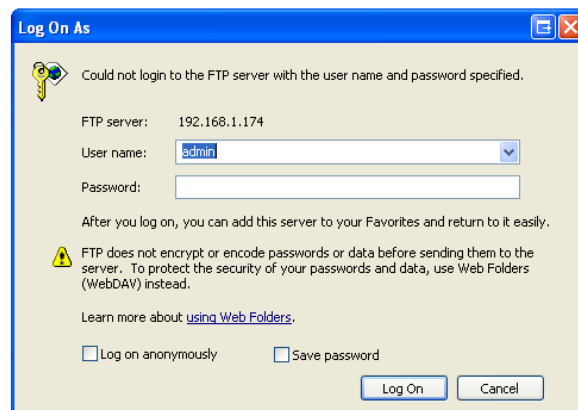
Next, double-click on the directory to which you would like to upload your files.
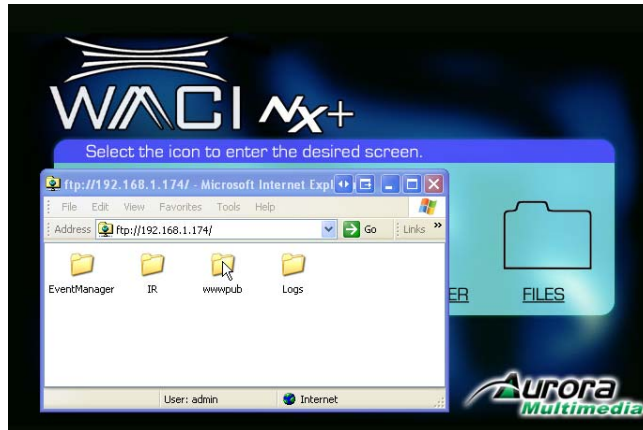
Finally, drag and drop selected files from your File Explorer:
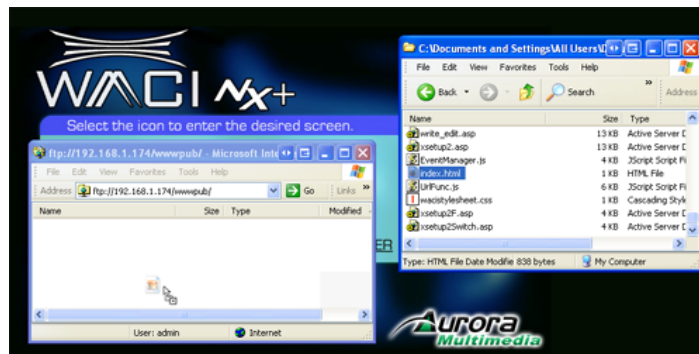
XIII. The Event Manager

The Event Manager is a feature of the WACI that allows you to program the WACI to perform operations autonomously. With the Event Manager, you can associate different hardware or software events with actions to be performed directly on the WACI. There is then no need to continuously control the WACI using some client side software. Client side software can then be used to simply check the WACI's status.

Overview of Events, Actions, & Variables

In the simplest terms, when certain conditions in hardware or variables are met, the WACI may be programmed to automatically perform specified tasks. To put this into the context of EVENTS, ACTIONS, and VARIABLES:

- An EVENT is triggered when conditions in hardware reach a user-defined state, or when a value of a user-defined VARIABLE matches a user-defined constant or expression.

- Once triggered, the EVENT can fire one or more ACTIONS.

- The conditions in hardware are typically read from one of the ports or the internal clock.

- Under many circumstances, user-defined states and constants can be written as expressions.

The Event Manager Web Page

The Event Manager page may be accessed by clicking on "Events" at the top of any page. This page is organized by Events, Actions, and Variables.

Note that Actions are always associated with its Event. Selecting an Event will highlight the Event and display all its associated Actions.

Events, Actions, and Variables may be added, edited, and deleted from this page.

Clicking on the heading at the top of the Event or Variable list will sort the respective list. You can use groups to help organize your Events and Variables.

In addition, you may "Clone" (duplicate) an Event with or without its associated actions.

Disabling the Event Manager renders all triggers inactive without deleting them. This is often useful, sometimes necessary, while:

- Configuring or reconfiguring your WACI with a device.

- Debugging configurations.

- Restoring, copying, or updating an Event Manager file (see EventManager, page 57).

Disable the Event Manager by selecting the Disabled" radio button at the top of the page. Enable the Event Manager to reactivate all triggers when you are finished with your WACI administration.

Events

Creating and Editing Events

Selecting the "Add event" button from the Event Manager page will create a new Event, while selecting the "Edit" button above the Event list will allow you to edit whichever Event is highlighted.

You will be prompted for the name of the Event, the type of Event, and any options associated with the Event, as described in the next two sections.
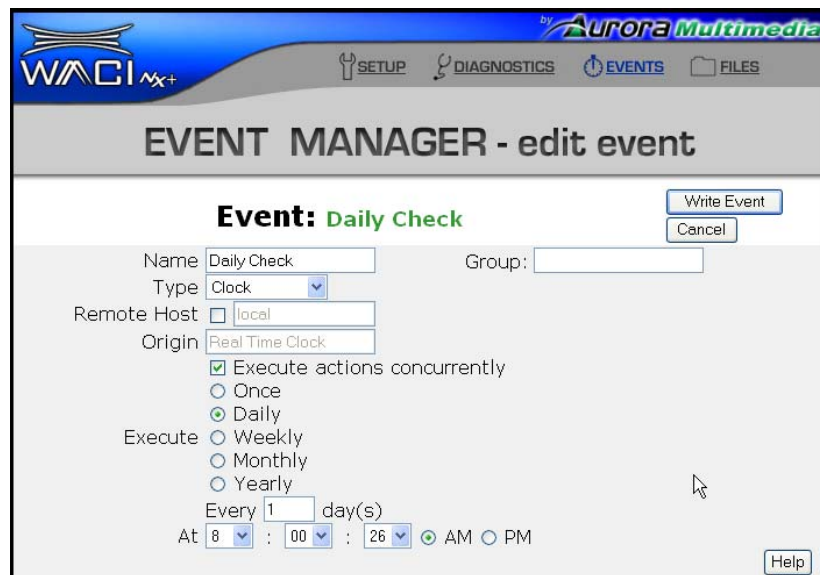
Types of Events

As mentioned earlier, Events are triggered when a condition is met in the hardware, a Variable matches a specific value, or an expression evaluates to a non-zero value. The options for triggering an Event depend on its type. The system recognizes the following types of Events:

Clock

A Clock Event is triggered at a specified time of day on a calendar. Options include the start date and time, as well as the one-time, daily, weekly, monthly, or yearly recurrence.

Timer

The timer operates by counting down from a value specified by hours:minutes:seconds. When the timer "runs out", the Timer Event is triggered, and the timer is restarted at the specified value.

While Clock Events allow you to schedule recurring tasks based on fixed, regular calendar periods (daily, weekly, etc,), Timer Events allow you to schedule recurring tasks based on user-defined time periods. For example, a Timer Event may be triggered every 27 seconds or 36 hours.

Variable

A Variable Event is triggered when the value of a Variable matches a constant or expression. If the "(expression)" check box is checked, then the expression defined by "Trigger value" is evaluated whenever the Variable specified in "Origin" is modified.

Serial

Serial Events are triggered when the INPUT value at a specified port matches a constant or expression. If the "(expression)" check box is checked, use the "\g" escape character (see Escaping Special Characters, page 91) to get access to the incoming serial data. Use a "Trigger value" of "*" (see Wildcard Characters, page 91) to cause any incoming serial data to trigger the Event.

64

DSP DIO (Digital Input/Output)

(Available for the WACI NX+ only.)

A DSP DIO Event is triggered when the INPUT or OUTPUT value at a specified Digital I/O port reaches the desired state, either low or high. If an expression is defined for a DSP DIO Event, then the expression is evaluated whenever the state of DSP DIO port changes.

DSP A/D D/A (Analog/Digital and (Digital/Analog Converters)

(Available for the WACI NX+ only.)

DSP A/D D/A Events are triggered when the INPUT value at a specified A/D or D/A Converter port matches a constant or expression. The trigger value is normally stated as a range, e.g. `0 to 150`.

Startup

Startup Events are triggered while the WACI is booting up (typically from reset or power on). These Events are useful to place the WACI or any devices connected to the WACI into a pre-defined initial state.

Telnet

The Telnet Event connects via TCP to a Telnet server and monitors any incoming data. The connection to the Telnet server will be continuously maintained, and if the connection is dropped, the Event will attempt to reconnect once per minute. The WACI also supports incoming Telnet connections, and will accept those connections if an Event exists having an "Origin" that matches the incoming connection.

Fill the "Origin" field with the IP address or machine name of the Telnet server. Use the "Trigger Value" to define the value to match to cause the Event to be executed. The "Trigger value" used by a Telnet Event is similar in format to a Serial Event's.

Network

The Network Event allows you to trigger when the status of one of the Ethernet Ports changes.

Temperature

The Temperature Event allows you to trigger when the internal temperature of the WACI reaches a certain degree. This is great for monitoring temperature inside a cabinet.

IR Input

The IR Input Event allows you to trigger events based on the IR command received from the external device connected to WACI using IR In port.

Named Event

The Named Event is the Event that can be only triggered manually.

Other Event Options

Group Name

To help organize Events into meaningful groups, a Group field is available. Enter any text into this field. On the main Event Manager page, you can click on the Group label at the top of the Event list's Group column to sort the Events by their Group name.

Execute Actions Concurrently

When the "Execute actions concurrently" check box is CHECKED, all the actions associated with the Event will be executed concurrently at the time the Event is triggered. Use the Action's "Delay By" value to offset the start time of a particular Action.

If this check box is UNCHECKED when the Event is triggered, its actions will be executed sequentially, one at a time, and in the order displayed on the Event Manager web page. The next Action in the list will not start until the previous Action has fully completed. Setting an Action to execute infinitely will prevent any Actions listed afterwards from running.

Expression

When the "(expression)" check box is CHECKED, the value specified in the "Trigger value" text box will be parsed as an expression (page 78). When the expression evaluates to a value not equal to 0, then the Event triggers and executes its Actions.

If the "(expression)" check box is UNCHECKED, the value specified in the "Trigger value" text box will be considered a constant.

Trigger Value

This text box is used for Events that are triggered by matches on a user-defined constant or expression. Like Variables, constants may be strings, numbers, schedules, or ranges.

To trigger an Event using an expression, the result of the expression must be resolved to an integer value not equal to 0. An expression that results in a string value will not trigger unless the string can be converted to an integer value that is not equal to 0, e.g. `Hello` won't trigger, but `3` will.

Remote Events

When connecting two or more WACIs up to a system, one of the WACIs can be used as a controlling WACI. The controlling WACI monitors the state of the remote WACI, and triggers the remote Event on a change in the remote WACI's hardware or a change in one of the remote WACI's Variables. To make a remote Event, check the "Remote" check box and specify the host name or IP address of the remote WACI. To make setup easier, make sure the both the controlling and remote WACIs are connected to the network and powered up.

When a remote Event is created, a copy of the Event is place on the remote WACI. This remote copy cannot be changed, but can be seen if you open the Event Manager web page on the remote WACI. The copy will have the IP address of the local WACI appended to the end of the Event's name.

Actions

Creating and Editing Actions

Actions are the tasks performed when an Event is triggered, so they are always associated and displayed with their Events.

Actions may be created from the main Event Manager web page by first selecting an Event, highlighting it, and then selecting the "Add action" button.

Similarly, to edit an Action, first select an Event, and then select the associated Action. When the desired Event and Action are both highlighted, the "Edit" button above the list of Actions may be selected. From the "edit action" page, you will have several options based on the type of Action.

Types of Actions

Variable

A Variable Action allows you to change the value of a Variable (see Variables, page 76) once an Event is triggered. Options that must be defined for the Variable Action are the Output Variable and the Output Value.

The "Output Value" may be set to a constant, such as `Hello` or `5`, or an expression, such as `Counter+1`. If the "(expression)" check box is UNCHECKED, the value will be interpreted as a constant of the same type as the Variable.

If the "(expression)" check box is CHECKED, the "Output to" Variable will be assigned the value calculated for the expression in the "Output Value" text box. The Variable's type is not changed by the assignment. Prior to assigning the result of the expression to the Variable, the result is converted to the Variable's type, e.g. a number. The interpretation and calculation of expressions are detailed in the Expressions section.

Serial

A Serial Action allows you to send a string to a specified port. If the "(expression)" check box is UNCHECKED, the value in the "Output Value" text box will be sent as a string to the specified port.

If the "(expression)" check box is CHECKED, the expression in the "Output Value" text box will first be interpreted as described in the Expressions section and the resulting string value will then be sent to the specified port. If the result of the expression is a number, then it will be converted to a string prior to being sent out the serial port.

Event

An Event can fire Actions that trigger or cancel other Events. An Event Action allows you to specify the Events to "execute" or "cancel". Executing an Event means that if the Action's Event is not already running at the time the Event Action is called, the Action's Event will then be triggered, and any associated Actions will be fired. Canceling an Event will stop the running of any Actions associated with the Event, then un-trigger the Event.

70

NOTE: When an Event is triggered, it remains triggered until all Actions have been completed. Once the Actions are complete, the Event's "triggered" flag is reset.

E-Mail

The E-mail Action sends an e-mail to a set of recipients. There are fields for the To, Cc, and Bcc addresses, as well as, a field for the message body. The message body can be either a simple text string, or a complex expression. The subject line of the e-mail is created from the name of the Action.

There are no repeat values specified for E-Mail Actions. The Action will execute only once per Event trigger.

Use the "Timeout in" value to have the Action abort, if it doesn't complete before the time-out period expires.

HTTP Post

An HTTP Post Action can be used to post data to a web server. The format of the posted data is dependent on the capabilities of the server receiving the post. The data returned by the server is discarded. HTTP Post Actions are useful for commanding a server to perform a particular operation.

There are no repeat values for HTTP Post Actions. The Action will execute only once each time the owning Event is triggered.

Use the "Timeout in" value to have the Action abort, if it doesn't complete before the time-out period expires.

The HTTP Post Action can be used to make RPC calls to other WACIs. Set the "Output to" field to the `/rpc` directory of the other WACI, e.g. `192.168.0.120/rpc`. The "Output value" should be the RPC call (see Syntax for HTTP Post).
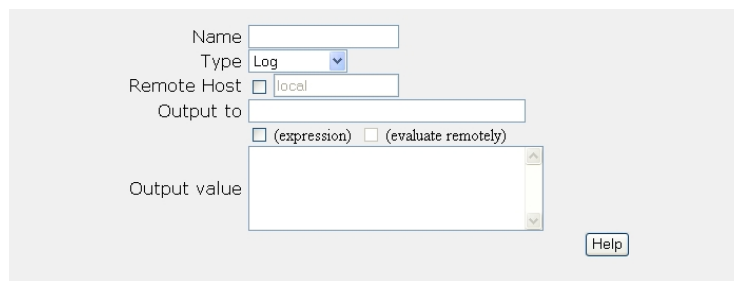
Log

Use the Log Action to write information about the status of the system, or about a specific Event to a text log file. Specify the name of the log file using the "Output to" field. The content of the "Output value" field is written to the log file. The "Output value" can contain either an expression or a simple piece of text.

Log files are created in a `/logs` directory on the WACI. To get access to the file, you can use FTP, or the `Log_ReadFile` RPC function.

There are no repeat values for Log Actions. The Action will execute only once each time the owning Event is triggered.

Telnet

A Telnet Action can be used to send data to a Telnet enabled network server device. The format of the telnet data is dependent on the capabilities of the server receiving the data. The data returned by the server can be obtained if a Telnet Event is created for the same server. A Telnet Action will create a temporary connection to a Telnet server, if no Telnet Event to the same server already exists.

There are no repeat values for Telnet Actions. The Action will execute only once each time the owning Event is triggered.

Use the "Timeout in" value to have the Action abort, if it doesn't complete before the time-out period expires.

DSP DIO

(Available for the WACI NX+ only.)

A DSP DIO Action sends an Output Value of "High" or "Low" to the specified port. The state of the DIO port will change only if the port is set as an output. You can check whether a port is set as an input or output using the DSP DIO diagnostics (see DSP Diagnostics, page 52).

DSP D/A

(Available for the WACI NX+ only.)

The DSP D/A action outputs a voltage of -5v to +5v to the specified port. The voltage level of the DSP D/A port will change only if the port is set as a D/A. You can check how a port is set using the DSP diagnostics (see DSP Diagnostics, page 52).

Relay

(Available for the WACI NX+ only.)

A Relay Action turns a relay on or off. Set the "Output value" to the state you want the Action to set the relay to.

IR Port

IR Port Actions send learned IR Commands (see Learn IR Command, page 47) to the specified port. The IR Command to execute must be specified by its Group name and Command name.

Buzzer

Buzzer Action enables and disables an internal buzzer. It can be used to let users know about anything important, when they do not have an easy access to the device.

Action Timing Options

Actions may be:

- Infinite - execute forever, or until another Action cancels the Event. Two notes: first, checking the "Infinite" checkbox will override the "Execute" number of times field. Second, the associated Event will remain triggered unless another Action cancels the Event.

- Executed a particular number of times. Actions may be specified to execute as many as 2^{31} times. This field is not used if the "Infinite" check box is CHECKED.

- Delayed by as many as 2^{28} seconds. The value can be a real number, e.g. 2.5 for 2500 milliseconds. If the Action is repeated any number of times, this delay only applies to the **first** run.

- Repeated every *N* seconds, where *N* can be a maximum of 2^{28} seconds. The value can be a real number, as opposed to an integer, e.g. 2.5 for 2500 milliseconds. *N* applies to the time period before the **second** and all subsequent runs. For remote Actions, this value doubles as a network timeout value as well.

- Aborted if they don't complete within a defined time-out period. This applies only to HTTP, Telnet, and E-mail Actions.

Remote Actions

When connecting two or more WACIs up to a system, one of the WACIs can be used to control the other WACI. The controlling (or local) WACI would then manage the state of the hardware on the remote (slave) WACI. To set an Action to be performed on the remote WACI, check the "Remote" check box and specify the host name or IP address of the remote WACI. To make setup easier, make sure that both the controlling WACI and remote WACI are connected to the network and powered up.

Network Timeouts

By default, Actions that communicate over a network (E-mail, Telnet, HTTP Post, and remote Actions) timeout after 30 seconds if no connection can be made to the remote server or remote WACI. You can control the timeout value by entering a non-zero value into the "Repeat every" edit box for remote Actions or into the "Timeout in" edit box for E-mail, Telnet, and HTTP Post Actions. Setting the "Execute N times" value to 1 and the "Repeat every" value to 3 for a remote Action would cause the Action to perform its operation once, but fail if it couldn't complete it within 3 seconds.

Setting proper timeout values helps reduce the overhead of the Event Manager's network functions. Remote Actions running on another WACI on the Local network could use a low time-out value, e.g. 2 seconds; whereas, Actions run on a remote WACI connected through a public network might have a higher timeout value, e.g. 10 seconds.

Variables

Variables are used to store information that can be retrieved or acted upon by the Event Manager, or by calls to RPC functions (see Variable Methods, page 168). You can create Variables that hold temporary data, or configure them to keep their value even when the WACI is reset. You can also use Variables to store data to be used by your own Web interface or Flash application stored on the WACI. Use the RPC calls to access the Variables within Flash, or VB Script to access them within an HTML page.

Variables are created independently of Events or Actions, and any Variable may be used in any Event or Action; however, a Variable **may not be deleted** until it is cleared from all the Events or Actions that use it.

Creating and Editing Variables

Variables may be created from the main Event Manager web page (see The Event Manager Web Page, page 61) by selecting the "Add variable" button. To edit an existing Variable, first select it so that it is highlighted, then select the "Edit" button above the list of Variables. The "edit variable" screen allows you to specify options and values for a Variable.
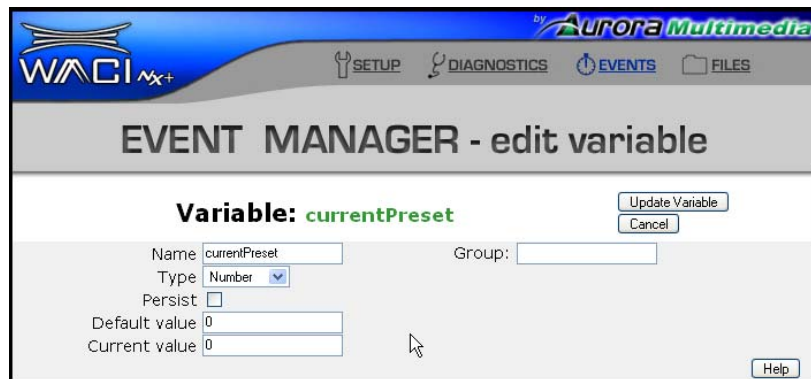
Some options for Variables will differ by the type of Variable. The options common to all Variables are Persistence, Default Value, and Current Value.

Group Name

Use the Group field to help organize Variables into meaningful groups. Enter any text into this field. On the main Event Manager page, click on the Group label at the top of the Variable list to sort the Variables by their Group name.

Persistence

When the persistence flag ("Persist" check box) is set, the "Current value" of the Variable will be stored to permanent storage whenever it is changed. This value will persist through a system reset or power failure, so the Default Value is not used when the "Persist" check box is CHECKED.

76

The storage of the Current Value does take some time, since it is stored into flash. Therefore, if a Variable is changed often, such as a counter, the WACI will perform more efficiently if you set a Default Value, and UNCHECK the "Persist" check box.

Default Value

The "Default value" is copied to the "Current value" whenever the system starts up. The start-up typically occurs from a system reset or a power failure. If a Variable has the persistent flag set, the "Default value" is not used.

Current Value

The "Current value" is the working value of the Variable. It is the value accessed in all Event and Action expressions. In addition, Variable Actions (see Variable, page 69) send their outputs to the "Current value".

The "Current value" is set to the "Default value" during system start-up, if the Persistent flag is not set.

Types of Variables

Number

A Number Variable can have an integer value between -2^{31} and $+2^{31}-1$.

Schedule

A Schedule Variable can have a time value, specified by `[hour]:[minute]:[second] [AM/PM]` fields.

String

A String Variable can be a Unicode string up to 128 characters in length.

77

Range

A Range Variable specifies a range of values between a minimum and maximum value. Each value can be a number between -2^{31} and $+2^{31}-1$. The maximum value should be greater than the minimum.

Range Variables have one other parameter, which is calculated from the minimum and maximum values. The de-bounce value, displayed after the maximum value, is used to provide a small measure of hysteresis to prevent flutter.

Functions

Note: ALL FUNCTIONS ARE CASE SENSITIVE!!!

***** Highlight items ARE NOT currently in WACI Help File *****

Note: You can utilize *decimal, hex, binary*, *and octal* integers for functions as exampled here: CHAR(66) = CHAR(0x42) = CHAR(0b1000010) = CHAR(0o102) = 'B'

The following functions are available for use when the expression check box is checked in one of the supported actions. The following types of actions support expressions: Variable, Serial, DSP D/A, IR Serial, HTTP Post, Email, Log, and Telnet interpreting Values as Expressions

- **ASCII (Char)** Returns the ASCII numeric value for a character passed in. Example: ASCII("A"), ASCII(var_name). Example: varstring = CHAR(66)
- **atof(string)** Returns floating-point number
Converts a numeric string with floating-point to a floating-point number. Example: atof("test")

- **atol(string) or atoi(string)** Returns integer
  Converts a numeric string to a number. Range of input number: 0 to 2147483647. Example:
  atoll("test") or atoi("test")

- **BinToSum(binary Stream)** - adds all of the bytes of a binary data stream and returns the sum
  ???

- **bincpy(binary_data, Start, Count)** - returns 'count' bytes of binary data starting at 'start'. ???

- **binlen(binary_data)** - - returns the number of bytes of binary data

- **CHAR(integer)** Returns string
  Converts a integer to a ASCII character. Range of input number: 0 to 255. Example:
  varstring = CHAR(65)
  varstring = 'A'

- **ftoa(float)** Returns string
  Converts an floating-point number to a string representation of the number. Example:
  ftoa(3.14)

- **GetAudioVolume()** Returns integer (NX-DAP)
  Gets the currently set system volume level in a range from 0 - 31 (31 being the loudest and 0
  being almost silent).

- **GetTrackLength()** Returns integer (NX-DAP)
  Gets the total track duration in milliseconds.

- **GetTrackPosition()** Returns Integer (NX-DAP)
  Gets the current track position in milliseconds.

- **GetSystemTime()** Returns the current time
  If assigned to a Number, then returns the number of milliseconds since the start of the day.
  Assigning the return value to a String gives the date and time in text form. Example:
  GetSystemTime()

- **HEX (Char)** Returns a HEX string that represents the character value passed in. Example:
  HEX("A"), HEX(var_name).

- **LStr(string, integer)** Returns string
  Return integer number of characters from the left of string. Example:
  LStr("abcdef",3) returns "abc"

- **ltoa(integer) or itoa(integer)** Returns string
  Converts an integer to a string representation of the integer. Domain of return number:
  -2147483648 to 2147483647. Example:
  ltoa(31) or itoa(31)

- **Ltrim(string)** Returns string
  Remove leading spaces from string. Example:
  Ltrim("   test") returns "test"

- **Mod(interger,divisor)** Returns remainder of division. Ex: mod(10,3) = 1

- **Padl(string, Length)** Returns string
  Pads a string with spaces on the left to the length specified. Returns a string. If Length
  is less than the length of String, then String is returned. Example:
  Padl(" test", 4)

- **Padr(string, Length)** Returns string
  Pads a string with spaces on the right to the length specified. Returns a string. If Length
  is less than the length of String, then String is returned. Example:
  Padr("test ", 6)

- **rand(min, max)** Returns integer
  Generates a random number between 0 and 32,767 seeded by the number of milliseconds since
  the device boot.

- **replace(String, S1, S2)** Search string for findstr and replace with newstr. If the string S1
  cannot be found inside of String, just return the original string. Example:
  replace("abcdef","xyz","abc") returns "abcdef".

  If the string to replace occurs more than once, replace each instance. Example:
  replace("abcabcabc","bc","Z") returns "aZaZaZ".

- **RStr(string, integer)** Returns string
  Return integer number of characters from the right of string. Example:
  string="abcdef", RStr(string,3)="def"

- **rstrstr(string, string2find)** Returns integer
  Searching Right to Left, finds string2find inside of string and returns the beginning location
  of string2find.
  Failure to find returns -1. Example:
  string="abcabc", string2find="bc", returns 4

- **Rtrim(string)** Returns string
  Removes trailing spaces from string. Example:
  Rtrim(" test") returns "test"

- **strlen(string)** Returns Integer
  Returns the number of characters in the string. Example:
  strlen("test") returns "4"

- **strlwr(string)** Returns string
  Returns the lower case version of a string. Example:
  strlwr("TEST") returns "test"

- **strstr(string, string2find)** Returns integer
  Searching Left to Right, finds string2find inside of string and returns the beginning
  location of string2find.
  Failure to find returns -1. Example:
  string="abcdef" string2find="bc" = returns 1

- **StrToBin(string)** - converts a string to a binary data stream, and returns the binary data stream
  ???

- **strupr(string)** Returns string
  Returns the upper case version of a string. Example:
  strupr("test") returns "TEST"

- **substr(String, StartChar, NumOfChars)** Returns string. Example:
  substr("abcdef",2) or substr("abcdef",2,0) returns "cdef"

- **Trim(string)** Returns string
  Remove leading and trailing spaces from string. Example:
  Trim(" test ") returns "test"

Expressions

Trigger Values for Events and Output Values for Actions may be written as expressions (instead of constant values) by checking the "(expression)" flag. Evaluating expressions is described in this section.

Interpreting Values as Expressions

If the "(expression)" option is UNCHECKED, the trigger and output values are treated as simple strings, and no quotation marks are needed. (NOTE: Variable Events and Actions automatically interpret the value based on its output or input Variable's type (see Types of Variables, page 77).

On the other hand, if the "(expression)" option is CHECKED, the WACI will treat the values as an expression and evaluate them before triggering or sending an output.

In an expression, any string values must now be in quotes; otherwise, the token will treated as an identifier. For example:

- "Hello" (in quotes) is a string value in the expression.

- Hello (without quotes) is an identifier for a Variable.

Special characters inside string values (inside quotes) must be escaped (see Escaping Special Characters, page 91) or URL encoded (see Send String, page 50).

Evaluation of Expressions

Expressions are evaluated strictly left-to-right. Order of operations is NOT supported; however, parentheses may be used to prioritize the execution of the expression.

In addition, the type of operation (string, integer, etc) and the final value of the expression are determined by the first token. If a string cannot be resolved to an integer during an integer operation, it takes on the value of 0 (zero).

Some examples:

This expression...	... resolves to this value.	Notes
3+4*5	35	No order of operations.

This expression...	... resolves to this value.	Notes
3+(4*5)	23	Priority is indicated by parentheses.
"Hello"*3	"HelloHelloHello"	* is a string operator here.
"Hello"+3	"Hello3"	The number 3 becomes part of the "Hello" string because the string comes first.
Hello+3	7	Assuming Hello is a number Variable holding a value of 4.
3+"6"	9	The string "6" is valued as 6, a number, and 3 is a number.
3+"Hello"	3	The string is treated as 0 because "Hello" has no integer interpretation.

Operators

The operators that are supported are:

- Arithmetic operations: +, -, *, /

- Logical comparison operators: ==, != , <, <=, >, >=

- Logical operators: &&, ||, !

- Bitwise operators: &, | , ~,^

- Range operator: in

- Assignments: =, +=, -=, *=, /=, &=, |=

Addition/Plus (+) Operator

The '+' operator is used to append one string to another, or to add two integer values together.

Adding a number to a string will produce a string with the number (as text) appended to the end. Adding a string to a number results in just the number (unless the string contains a value that can be converted to a number).

For example:

This expression...	... resolves to this value.
3+4	7
"Hello"+" "+"There"	"Hello There"
"Hello"+3	"Hello3"
3+"Hello"	3
3+"6"	9

Subtraction/Minus (-) Operator

The subtraction operator works with both numerical values and strings values. Use this operator with strings to remove a sub-string from a source string, or use this operator to subtract two integer values.

For example:

This expression...	... resolves to this value.
6-2	4
"Hello"-"ll"	"Heo"
10-"3"	7

Multiplication/Times (*) Operator

This operator multiplies two numbers, or can be used for creating a string with a repeating value. The numbers and strings can be either Variables or literals.

For example:

This expression...	... resolves to this value.
4*5	20
"Hello"*3	"HelloHelloHello"
3*"Hello"	0

Division/Divide (/) Operator

The divide operator is valid only for numerical values and Variables. Use this operator to divide one numerical value by another. The resultant type is an integer value. If the denominator is 0, then the operation evaluates to 0.

For example:

This expression...	... resolves to this value.

10/3	3
10/2	5
10/0	0

Equal Comparison (==) Operator

The equal comparison operator will compare two tokens, which can be strings or integers. The expression will return 1 if the two tokens are equivalent, and 0 if they are not.

String values can be compared against wildcard values, e.g. "Hello*".

For example, assuming MyNumber has an integer value of 4, and MyName has a string value of "John":

This expression...	... resolves to this value.
MyNumber==4	1
MyNumber=="4"	1
MyNumber==5	0
MyName=="John"	1
MyName=="J*"	1
0=="John"	1

Not-Equal Comparison (!=) Operator

The not-equal comparison operator will compare two tokens, which can be strings or integers. Exactly opposite to the equal comparison operator, the expression will return 0 if the two tokens are equivalent, and 1 if they are NOT equal.

String values can be compared against wildcard values, e.g. "Hello*".

For example, assuming MyNumber has an integer value of 4, and MyName has a string value of "John":

This expression...	... resolves to this value.
MyNumber!=4	0
MyNumber!="4"	0
MyNumber!=5	1
MyName!="John"	0
MyName!="??ul"	1

This expression...	... resolves to this value.
`0!="John"`	0

Greater-Than Comparison (>) Operator

The greater-than comparison operator will compare two tokens, which can be integers or strings (assuming the string represents an integer value). The expression will return 1 if the first token has a larger integer value than the second, and 0 if the second value is larger or equal.

For example, assuming `MyNumber` has an integer value of 4:

This expression...	... resolves to this value.
`MyNumber>5`	0
`MyNumber>4`	0
`MyNumber>3`	1

Greater-Than or Equal Comparison (>=) Operator

The greater-than or equal comparison operator will compare two tokens, which can be integers or strings (assuming the string represents an integer value). The expression will return 1 if the first token has an integer value larger than or equal to the second, and 0 if the second value is larger.

For example, assuming `MyNumber` has an integer value of 4:

This expression...	... resolves to this value.
`MyNumber>=5`	0
`MyNumber>=4`	1
`MyNumber>=3`	1

Less Than Comparison (<) Operator

The less-than comparison operator will compare two tokens, which can be integers or strings (assuming the string represents an integer value). The expression will return 1 if the first token has a smaller value than the second, and 0 if the second value is smaller or equal.

For example, assuming `MyNumber` has an integer value of 4:

This expression...	... resolves to this value.
`MyNumber<5`	1

This expression...	... resolves to this value.
MyNumber<4	0
MyNumber<3	0

Less Than or Equal Comparison (<=) Operator

The less-than or equal comparison operator will compare two tokens, which can be integers or strings (assuming the string represents an integer value). The expression will return 1 if the first token has an integer value smaller than or equal to the second and 0 if the second value is smaller.

For example, assuming MyNumber has an integer value of 4:

This expression...	... resolves to this value.
MyNumber<=5	1
MyNumber<=4	1
MyNumber<=3	0

Logical-And (&&) Operator

Assume you have two Boolean (true/false) expressions, Expression1 and Expression2, each returning true (1) or false (0). The logical-and operator determines the true or false state of the complex expression Expression1&&Expression2 using the rules applying to a traditional logical AND statement:

If Expression1 is...	and Expression2 is...	Expression1&&Expression2 resolves to...
True (1)	True (1)	True (1)
True (1)	False (0)	False (0)
False (0)	True (1)	False (0)
False (0)	False (0)	False (0)

In other words:

This expression...	... resolves to this value.
(3==3)&&(4==4)	1
1&&(4==5)	0
0&&(4==4)	0

This expression...	... resolves to this value.
(3==6)&&0	0

Logical-Or (||) Operator

Assume you have two Boolean expressions, Expression1 and Expression2, each returning true (1) or false (0). The logical-or operator determines the true or false state of the complex expression Expression1||Expression2 using the rules applying to a traditional logical OR statement:

If Expression1 is...	and Expression2 is...	Expression1&&Expression2 resolves to...
True (1)	True (1)	True (1)
True (1)	False (0)	True (1)
False (0)	True (1)	True (1)
False (0)	False (0)	False (0)

In other words:

This expression...	... resolves to this value.
(3==3)||(4==4)	1
1||(4==5)	1
0||(4==4)	1
(3==5)||0	0

Logical-Not (!) Operator

The logical-not is a prefix operator that "negates" the Boolean expression that follows it. In other words, if Expression1 returns true (1), !Expression1 returns false (0).

For example:

This expression...	... resolves to this value.
!0	1
!(4==3)	1
!("Hello"=="Hello")	0

Bitwise-And (&) Operator

The bitwise-and operator performs the logical AND operation bit by bit between two integers. This is better understood by looking at the example of `12&10`. The binary value of `12` is `1100`, and the binary value of `10` is `1010`. `12&10` is computed by performing an AND between each of the corresponding four bits:

```
12:          1    1    0    0

10:          1    0    1    0

12&10:       1    0    0    0
```

Therefore: `12&10` = (Binary) `1000` = (Decimal) `8`

Bitwise-Or (|) Operator

The bitwise-or operator performs the logical OR operation bit by bit between two integers. This is better understood by looking at the example of `12|10`. The binary value of `12` is `1100`, and the binary value of `10` is `1010`. `12I10` is computed by performing an OR between each of the corresponding four bits:

```
12:          1    1    0    0

10:          1    0    1    0

12|10:       1    1    1    0
```

Therefore: `12|10` = (Binary) `1110` = (Decimal) `14`

One's Complement (~) Operator

The one's complement operator is a bitwise-not prefix operator that performs a logical NOT to each bit of an integer. This is better understood by looking at the example of `~5`. The binary value of `5` is `101`. `~5` is computed by performing a NOT on each of the three bits:

```
5:           1    0    1

~5:          0    1    0
```

Therefore: `~5` = (Binary) `010` = (Decimal) `2`

Realize that the bitwise not is performed on a 32-bit value, e.g. `~0` converts to a 32-bit value with all bits in the integer set to 1. To get just the bits you want, you should AND the result with the bits of interest, e.g. `~5 & 7`.

Range (in) Operator

The 'in' operator compares a numerical value or Variable against a range Variable. The resultant type is an integer value of either 0 or 1. If the value is within the range, 1 is returned; otherwise, 0 is returned.

For example, assume Count has a value of 15, Range10 has a value of 1 to 10, and Range20 has a value of 1 to 20

This expression...	... resolves to this value.
`Count in Range10`	0
`Count in Range20`	1

Assignment (=) Operator

The '=' operator stores a value (R-Value) into a Variable (L-Value). The value on the left of the assignment operator is the L-Value. The result of the assignment is an R-value, and cannot be used in another assignment, though it can be used as the result of an expression.

For example:

This expression...	... resolves to this value.
`3=4`	Illegal (L-Value not a Variable)
`MyCounter=1`	1 (also stores 1 into the MyCounter Variable)
`5==(MyCounter=1)`	0 (stores 1 into MyCounter as well)

Addition Assignment (+=) Operator

The '+=' operator is used to append one string to another, or to add two integer values together.

Adding a number to a string will produce a string with the number (as text) appended to the end. Adding a string to a number results in just the number (unless the string contains a value that can be converted to a number).

For example, assuming `MyNumber` has an integer value of 4, and `MyString` has a value of "`Hello`":

This expression...	... resolves to this value.
`MyNumber+=4`	7 (MyNumber also contains 7)
`MyString+=3`	"Hello3" (MyString also contains "Hello3")
`3+="6"`	Illegal (L-Value not a Variable)

Subtraction Assignment (-=) Operator

The subtraction assignment operator works with both numerical values and strings values. Use this operator with strings to remove a sub-string from a string Variable, or use this operator to subtract an integer value from a numeric Variable.

For example, assuming `MyNumber` has an integer value of 4, and `MyString` has a value of "`Hello`":

This expression...	... resolves to this value.
MyNumber-=1	3 (MyNumber also contains 3)
MyString-="ll"	"Heo" (MyString also contains "Heo")
10-="3"	Illegal (L-Value not a Variable)

Multiplication Assignment (*=) Operator

This operator multiplies two numbers, or can be used for creating a string with a repeating value. The numbers and strings can be either Variables or literals.

For example, assuming `MyNumber` has an integer value of 4, and `MyString` has a value of "`Hello`":

This expression...	... resolves to this value.
MyNumber*=5	20 (MyNumber set to 20)
MyString*=3	"HelloHelloHello" (MyString same as result)
3*=5	Illegal (L-Value not a Variable)

Division Assignment (/=) Operator

The divide operator is valid only for numerical values and Variables. Use this operator to divide one numerical value by another. The resultant type is an integer value. Division by 0 results in a return value of 0.

For example, assuming `MyNumber` has an integer value of 4:

This expression...	... resolves to this value.
MyNumber/=3	1 (MyNumber is also 1)
MyNumber/=0	0 (MyNumber is also 0)

Bitwise-And Assignment (&=) Operator

The bitwise-and assignment operator performs the logical AND operation bit by bit between an integer Variable and another integer. The result of the operation is stored in the L-Value of the operator. Refer to the Bitwise-And operator for an example of the bitwise AND.

Bitwise-Or Assignment (|=) Operator

The bitwise-or assignment operator performs the logical OR operation bit by bit between an integer Variable and another integer. The result of the operation is stored in the L-Value of the operator. Refer to the Bitwise-Or operator for an example of the bitwise OR.

Wildcard Characters

There are two special characters used to match one or more characters in a comparison string:

> '*'   Matches one or more characters

> '?'   Matches any single character

Either character can be used anywhere within a string constant. When used with a comparison operator, "he*", "*llo", "he*o", and "he??o" will all match the string "hello".

When used to compare "\g" data within a serial or Telnet stream, the use of '?' is more efficient, since the total number of comparison characters can be known. Use of the '*' could require buffering large amounts of the incoming data.

Wildcard characters are not limited to expressions only. They can be used within simple text values (values where the expression check-box is not checked).

Escaping Special Characters

Escape characters can be placed anywhere in an expression, though the expression will fail to evaluate if the escape character causes an invalid expression. The escape character should be placed within quotes when defining an expression, e.g. "\m". Quotes are not needed if the "(expression)" check box is UNCHECKED for the Event or Action. Valid escape codes are:

Code	Name	Description
\a	bell alert	Character code: 0x07
\b	Backspace	Character code: 0x08
\d	Date	Represents the date as a string: MM/DD/YYYY
\f	Form feed	Character code: 0x0C
\g	Trigger data	Data that triggered the current Event

Code	Name	Description
\m	Time	Time that the current Event was triggered at: HH:MM:SS in 24hr format
\n	New line or linefeed	Character code: 0x0D
\o	Origin that triggered Event	Port number, Variable name, etc. The value depends on the type of Event.
\r	Carriage return	Character code: 0x0A
\t	Horizontal tab	Character code: 0x09
\v	Vertical tab	Character code: 0x0B
\'	Single quote	Character code: 0x27
\"	Double quote	Character code: 0x22
\\	Backslash	Character code: 0x5C
\?	Question mark	this is needed, because wild cards will be identified by * and ? within the string.
*	Asterisk	Character code: 0x2A

XIV. Remote Procedure Calls

Remote procedure calls (RPCs) are used to control a WACI using another WACI, or using a client computer. You can turn on and off the relays, send e-mail, add Events and Actions, etc. Anything the WACI can do can be configured or controlled through the RPC interface.

The WACI supports RPCs for most web interfaces, including HTML, Flash (.fla), and Active Server Pages (.asp). Methods are typically called using the HTTP Post syntax for HTML and Flash, and Visual Basic scripting for Active Server Pages.

Calls to the RPC server using HTTP post are done through the /rpc virtual directory, e.g. http://waci/rpc. Use this directory as part of the URL when creating the address for the HTTP post transaction.

This RPC reference covers:

- RPC Server Logs
- Syntax for HTTP Post
- Fault Codes
- Flash Example
- Visual Basic Scripting
- Note on Error Checking
- RPC Quick Reference to Methods
- Detailed List of All Methods

RPC Server Logs

As noted in Diagnostics web pages section, key information regarding the RPC server can be retrieved from the Log Files web page. This information includes the RPC server version and a list of available methods. The Log Files web page also allows you to enable logging for the RPC commands.

Syntax for HTTP Post

Call Tokens

Methods may be called using HTTP Post using the following call tokens:

- **version** - The RPC server version may be obtained through the Log pages in the Admin Web Pages. This token is entirely optional and may be safely omitted from the call.
- **encoding** - Number of times that the parameters (param1..paramN values) have been URL encoded. Some web applications URL encode the parameters more than one time. Use this optional parameter to define the number of times that the RPC server should decode the parameters. By default, this value is set to 1.
- **method** - Name of the method to call.
- **param**[*i*] - Value for parameter #*i*, where *i* is an integer.

General format for a call using HTTP Post:

```
[version=[ RPC Server Version ]&]method=[ Method to Call ][&encoding=[
Count ]][&param1=[ Value for Parameter 1 ]&param2=[ Value for Parameter 2
]&param3=[ Value for Parameter 3, etc...]]
```

See the detailed list of methods at the end of this reference for the **method** name and the values for **param1**, **param2**, etc. These parameter values are denoted by the text "[in]" in the Syntax and Parameters sections.

Response Tokens

The response tokens are:

- **status** - 0 for Failure, 1 for Success

- **faultCode** - If **status** is 0 (failure). **faultCode** will be an integer representing a specific error (see Fault Codes). This token is NOT returned if status is 1 (success).

- **response**[*i*] - Returned value #*i*, where *i* is an integer. If **status** is 0 (fail), the value for **response1** will be a string describing the failure, and the value for **response2** will be a debug string giving more information about the failure.

Returned values are strings, integers, or Booleans, depending on the method. See the detailed list of methods at the end of this reference for details about the returned values for any given method. These returned values are denoted by the text "[out, retval]" in the Syntax and Parameters sections.

The response to the HTTP Post will be a string in the format:

```
status=[ 0 or 1 ][&faultCode=[Fault Code]] [&response1=[ Value for
Response 1 ] &response2=[ Value for Response 2 ] &response3=[ Value for
Response 3, etc...]]
```

EXAMPLE #1: Serial_GetSettings

Let us look at the method Serial_GetSettings (defined on page 114). From the Syntax and Parameter sections, we can tell that param1 is a long integer that specifies either serial port #1 or #2. We can also see that response1 (retval, or return value) is a string describing the settings for the given port.

Using the HTTP Post format, the call to obtain settings from serial port #2 is:

```
version=2.0&method=Serial_GetSettings&param1=2
```

Since the version token is optional, this call may be simplified:

```
method=Serial_GetSettings&param1=2
```

A sample response for this call:

```
status=1&response1=9600,8,ODD,2,1,SOFTWARE
```

Responses are not usually used in its raw string format. A web programmer will typically parse the response before displaying or otherwise making use of the information received.

94

EXAMPLE #2: Net_GetSubnetMask

The call for the Net_GetSubnetMask method:

```
method=Net_GetSubnetMask
```

Sample response:

```
status=1&response1=255.255.255.0
```

Fault Codes

Fault Code	Fault Description
1	Function parameter out of range / item specified does not exist
2	Timed out waiting for a mutex or shared resource
3	Memory allocation failure
4	Configuration does not support action (i.e. setting a Digital I/O output when it is configured as an input)
5	Failed to find entry in database (registry or FRAM)
6	Failed to store persistent database (registry or FRAM), settings might not be saved
7	Some internal error occurred that does not fit another category. Must use extended error information to diagnose.
8	An expected file format was invalid
9	IR signal too strong, receiver is overloaded
10	IR signal too weak
11	Other IR capture error

Using Macromedia Flash

You may access all WACP RPCs using HTTP post. In Flash, you may accomplish this using LoadVars{} and sendAndLoad(). An example for making these calls follows.

STEP 1: Pass WACI IP Address to Flash

To begin, you must pass the IP address of WACI to your Flash file. There is more than one way to do this, but below, we describe one sample method using JavaScript. NOTE: In this script, the .html and .swf files must reside on the WACI to determine the IP address.

First, at the top of the .html file in which you are embedding your flash file (here, the name of the file is "MyWACIShockWaveFlashFile.swf"), insert this bit of script:

```
<script type="text/javascript" language="JavaScript">
      thispage = location.href;
      URLarr = thispage.split("/");
      theIP = URLarr[2];
      theFILE = "MyWACIShockWaveFlashFile.swf";
</script>
```

Next, use this bit of script to embed the flash file in the desired area of the web page:

```
<script type="text/javascript" language="JavaScript">
      document.write('<object id="WACI" classid="clsid:D27CDB6E-AE6D-11cf-
96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.
cab#version=6,0,29,0" width="100%" height="100%">');
      document.write('<param name="movie" value="'+theFILE+'" />');
      document.write('<param name="quality" value="high" />');
      document.write('<param name="FlashVars" value="callerID='+theIP+'"
/>');
      document.write('<embed name="flashMovie" src="'+theFILE+'"
FlashVars="callerID='+theIP+'" quality="high"
pluginspage="http://www.macromedia.com/go/getflashplayer"
type="application/x-shockwave-flash" width="100%"
height="100%"></embed>');
      document.write('</object>');
</script>
```

The IP Address may now be accessed from your Flash action scripts in the Variable "callerID".

STEP 2: Call RPC with HTTP Post

Once you have the IP address, you are able to access the RPC server. Load the method and it's parameters into a LoadVars() object as shown below, then make the call with sendAndLoad().

```
// Get WACI IP
RPCServerIP = [callerID, or whatever you used to get the IP address of the
WACI];

// Initialize Call & Response Tokens
WACI_Call = new LoadVars()
WACI_Response = new LoadVars()

// Set up RPC method and parameters you desire to call.
// Note that these are all URL-encoded strings.
// You may assign values based on buttons, text fields, constants, or
whatever else is appropriate for your page.
WACI_Call.method = "[Desired RPC Method]";
WACI_Call.param1 = "[Parameter 1 for the Method]";
WACI_Call.param2 = "[Parameter 2 for the Method]";
WACI_Call.param3 = "[Parameter 3 for the Method]";
[etc.]

// Call the RPC using HTTP Post
WACI_Call.sendAndLoad("http://"+RPCServerIP+"/RPC", WACI_Response, POST);
```

STEP 3: Response Tokens

At this point, if you used the script template above, you may now examine, use, and/or display the contents of:

```
WACI_Response.status
WACI_Response.faultCode
WACI_Response.response1
WACI_Response.response2
WACI_Response.response3
[etc.]
```

The response tokens, like methods and call tokens are all in the form of URL-encoded strings.

Visual Basic Scripting

To access an RPC using Visual Basic, first create a WACI object of type "WACI.UserAPI.1". Then, call the method with the object, specifying any parameters in parentheses. For example, this script in an Active Server Page retrieves the IP address of the WACI and the settings for serial ports 1 and 2.

```
Dim waciUser
Dim strIP, strErr
Dim strPort1, strPort2

set waciUser = CreateObject("WACI.UserAPI.1")

strIP = waciUser.Net_GetIPAddress
if(strIP = "") then
      strErr = waciUser.GetLastErrorString
end if
strPort1 = waciUser.Serial_GetSettings(1)
strPort2 = waciUser.Serial_GetSettings(2)
```

Some notes:

- Parameters and returned values are strings, integers, or Booleans, depending on the method.

- The above example includes error checking for Net_GetIPAddress. Methods will report errors based on the type of its return value. Strings will return the null string, integers will return -1, and Booleans will return FALSE.

- See the detailed list of methods at the end of this reference for details about the parameters and return values for any given method. Parameter values are denoted by the text "[in]" in the Syntax and Parameters sections. Similarly, the return value is denoted by the text "[out, retval]".

- String values should be passed in as URL encoded values. Returned strings are also URL encoded.

Note on Error Checking

The HRESULT, referred to in the Syntax of each method, ALWAYS returns "S_OK", so it is not useful in error checking.

Instead, use the error checking values described in the HTTP Post and Visual Basic above, and the Error Information Methods below (see Error Information Methods, page 101) to perform meaningful error checks.

Errors are based on the type of a function's out/retval value. On error, a string retval is returned as a NULL string, an integer retval returns -1, and Booleans will return FALSE.

RPC Quick Reference

Error Information Methods

GetLastErrorCode ()

Returns the fault code from the last failed method call.

Syntax

```
HRESULT GetLastErrorCode(
  [out, retval] long* Code
);
```

Parameters

Code   [out, retval] Integer representing fault code.

Remarks

The value of `Code` corresponds to `faultCode` of an HTTP Post response when `status` is 0 (failure).

See {see **Error! Reference source not found.**, page **Error! Bookmark not defined.**} for a list of defined codes.

GetLastErrorString ()

Returns the description of the error from the last failed method call.

Syntax

```
HRESULT GetLastErrorString(
  [out, retval] BSTR* ErrorString
);
```

Parameters

ErrorString   [out, retval] String describing an error.

Remarks

ErrorString corresponds to `response1` of an HTTP Post response when `status` is 0 (failure).

GetLastExtendedErrorString ()

Returns an extended description of the error from the last failed method call.

Syntax

```
HRESULT GetLastExtendedErrorString(
  [out, retval] BSTR* ExtErrorString
);
```

Parameters

ExtErrorString   [out, retval] String giving more information about an error.

Remarks

Corresponds to `response2` of an HTTP Post response when `status` is 0 (failure).

General Information Methods

AllState_Get ( )

Returns the entire hardware state

Syntax

```
HRESULT AllState_Get(
  [out, retval] VARIANT* Array
);
```

Parameters

Array   [out, retval] Array representing entire hardware state.

Remarks

The array is filled as follows. Each port value is separated from the next port value by a comma.

Element #	Field	Type	Example
0	Date	DATE	
1	Serial Rcv Buffer Count	BSTR	"0,1092"
(Elements 2-7 pertain to the WACI NX+ only.)			
2	Relays	BSTR	"1,0,0,1"
3	DIO IsOutput	BSTR	"0,1,0,0"
4	DIO State	BSTR	"1,1,1,1"
5	DIO Is Pulled Up	BSTR	"1,0,0,0"
6	AD Digital Readings	BSTR	"128,54,2096,8194"
7	AD Voltages	BSTR	"0.2,0,1.2,2.4"

GetMachineType ( )

Returns the type of WACI, either Plus or Junior

Syntax

```
HRESULT GetMachineType(
  [out, retval] long* Type
);
```

Parameters

Type   [out, retval] The parameter returns 1 for a WACI Plus and 2 if the device is a WACI NX Jr.

GetFirmwareVersion ( )

Returns a string containing the version information about the firmware loaded into the WACI.

```
HRESULT GetFirmwareVersion(
  [out, retval] BSTR* Version
);
```

Parameters

Version   [out, retval] A returned string of the format "DEVICE #.# DATE", where "DEVICE" is the name of the device, #.# represents the version number, and DATE represents the build date for the firmware image in MMM DD YYYY format.

ValidatePassword (Password)

Checks a password against the administrator password.

Syntax

```
HRESULT ValidatePassword(
  [in] BSTR Password,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

Password   [in] The password to check

Success   [out, retval] TRUE if the password matches the Administrator password, and FALSE if password does not match.

Remarks

If this function is called with an invalid password more than 25 times, then ValidatePassword will return FALSE for the next 15 minutes, regardless of whether the password passed into subsequent calls is valid or not.

Time_GetDate ( )

Returns current time and date on the device.

Syntax

```
HRESULT Time_GetDate(
  [out, retval] DATE* Date
);
```

Parameters

Date   [out, retval] DATE object.

Remarks

If the returned DATE object represents a date before 1900, the returned value will be negative, signifying an error.

The format is

```
[year],[month],[day],[hour],[minute],[second],[millisecond]"
```
February 10, 2003, 7:15PM would be represented as "2003,2,10,19,15,0,0".

Precision is provided to the second, so the millisecond value is ALWAYS "0".

Time_Sleep (Milliseconds)

Generates a wait period using the specified number of milliseconds.

Syntax

```
HRESULT Time_Sleep(
  [in] long Milliseconds,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Milliseconds   [in] The number of milliseconds to wait.

Success   [out, retval] TRUE if the Milliseconds value is greater than or equal to 0 and less than or equal to 999999, and FALSE if value is outside this range.

Remarks

The call to this function will not return until after the wait time has expired.

Network Methods

Net_GetNetCardCount ()

Returns the number of Ethernet ports on the device.

Syntax

```
HRESULT Net_GetNetCardCount(
  [out, retval] long* Count
);
```

Parameters

Count   [out, retval] The response send back from the server.

Net_GetIPAddress ()

Returns the current IP address of the device.

Syntax

```
HRESULT Net_GetIPAddress(
  [out, retval] BSTR* IP
);
```

Parameters

IP   [out, retval] String containing the current IP address of the device, such as "10.0.1.3".

Net_GetIPAddressEx (NetCard)

Returns the current IP address of the device.

Syntax

```
HRESULT Net_GetIPAddress(
  [in] long NetCard
  [out, retval] BSTR* IP
);
```

Parameters

NetCard [in] The ID of the netcard to get the IP address from

IP   [out, retval] String containing the current IP address of the device, such as "10.0.1.3".

Net_GetSubnetMask ()

Returns the current subnet mask of the device.

Syntax

```
HRESULT Net_GetSubnetMask(
  [out, retval] BSTR* SubnetMask
);
```

Parameters

> SubnetMask   [out, retval] String containing the current subnet mask of the device, such as "255.255.255.0".

Net_GetSubnetMaskEx (NetCard)

> Returns the current subnet mask of the network interface specified.

Syntax

```
HRESULT Net_GetSubnetMask(
  [in] long NetCard
  [out, retval] BSTR* SubnetMask
);
```

Parameters

> NetCard [in] The ID of the netcard to get the subnet mask from

> SubnetMask   [out, retval] String containing the current subnet mask of the device, such as "255.255.255.0".

Net_PostHtmlData (Url, PostData)

> Sends data to a web server on port 80 via the HTML post mechanism.

Syntax

```
HRESULT Net_PostHtmlData(
  [in] BSTR Url,
  [in] BSTR PostData,
  [out, retval] BSTR* RetData
);
```

Parameters

> Url   [in] The server address to post to, e.g. `http://waci/rpc`.

> PostData   [in] A string that represents the data to send to the server.

> RetData   [out, retval] The response send back from the server.

Net_PostHtmlDataEx (NetCard, Url, IsExpression, PostData, Timeout)

> Sends data to a web server. This function is similar to Net_PostHtmlData, but includes two additional parameters. `Net_PostHtmlDataEx` supports expressions and a timeout value.

Syntax

```
HRESULT Net_PostHtmlDataEx(
  [in] long NetCard
  [in] BSTR Url,
  [in] long IsExpression,
  [in] BSTR PostData,
  [in] long Timeout,
  [out, retval] BSTR* RetData
);
```

Parameters

> NetCard [in] The ID of the netcard to get the subnet mask from

Url	[in] The server address to post to, e.g. `http://192.168.0.100/rpc`.
IsExpression	[in] Flag indicating whether `PostData` contains simple text or an expression. Set to 1, if `PostData` is an expression; otherwise, set to 0.
PostData	[in] A string or expression that represents the data to send to the server.
Timeout	[in] The number of milliseconds to wait for a response from the web server before the call aborts.
RetData	[out, retval] The response send back from the server.

Net_SendMail(SenderId, ToIds, CcIds, BccIds, Subject, MessageBody)

Sends an email to the specified recipients.

Syntax

```
HRESULT Net_SendMail(
  [in] BSTR SenderId,
  [in] BSTR ToIds,
  [in] BSTR CcIds,
  [in] BSTR BccIds,
  [in] BSTR Subject,
  [in] BSTR MessageBody,
  [out, retval] long *Result
);
```

Parameters

SenderId	[in] Sender's email address, e.g. joe@mycompany.com
ToIds	[in] Email addresses to be included in the To: line of the email. Separate each email address with a semicolon.
CcIds	[in] Email addresses to include on the Cc: line of the email.
BccIds	[in] Email addresses placed in the Bcc: line of the email.
Subject	[in] Subject line for the email
MessageBody	[in] Body text of the email
Result	[out, retval] Returned as 0 if no error occurred; otherwise the return value is the error returned from the SMTP server.

Remarks

The WACI does not retry sending when it fails to send the email. A return code of 250 normally indicates that the email was sent successfully. Other error codes normally indicate a failure.

Net_SendMailEx (NetCard, SenderId, ToIds, CcIds, BccIds, ReturnPathId, ReturnRcpId, MsgComment, Subject, IsExpression, MessageBody)

Sends an email to the specified recipients. This function is an expanded form of Net_SendMail.

```
HRESULT Net_SendMailEx(
  [in] long NetCard,
  [in] BSTR SenderId,
  [in] BSTR ToIds,
  [in] BSTR CcIds,
  [in] BSTR BccIds,
  [in] BSTR ReturnPathId,
  [in] BSTR ReturnRcpId,
  [in] BSTR MsgComment,
  [in] BSTR Subject,
  [in] long IsExpression,
  [in] BSTR MessageBody,
  [out, retval] long *Result
);
```

Parameters

NetCard	[in] The ID of the Network port to send the mail through
SenderId	[in] Sender's email address, e.g. joe@mycompany.com
ToIds	[in] Email addresses to be included in the To: line of the email. Separate each email address with a semicolon.
CcIds	[in] Email addresses to include on the Cc: line of the email.
BccIds	[in] Email addresses placed in the Bcc: line of the email.
ReturnPathId	[in] Email address to send server errors and responses to. This parameter is normally set to the same address as the SenderId.
ReturnRcpId	[in] Email address to send return receipt requests to. Pass an empty string if a return receipt is not needed.
MsgComment	[in] Sets the comment field for the email.
Subject	[in] Subject line for the email.
IsExpression	[in] Pass a 1 if the MessageBody parameter contains an expression, or 0 if the body is a simple string.
MessageBody	[in] Body text of the email. Can be an expression.
Result	[out, retval] Returned as 0 if no error occurred; otherwise the return value is the error returned from the SMTP server.

Remarks

The WACI does not retry sending when it fails to send the email. A return code of 250 normally indicates that the email was sent successfully. Other error codes normally indicate a failure.

Telnet Methods

Telnet_Send (NetCard, Msg, MaxWaitMS)

Writes a string to the log file identified by `FileName`.

Syntax

```
HRESULT Telnet_Send(
  [in] long NetCard,
  [in] BSTR Msg,
  [in] long MaxWaitMS
  [out, retval] long* NotSent
);
```

Parameters

Netcard	[in] ID of network port to use.
Msg	[in] Message to send.
MaxWaitMS	[in] Timeout Value.
NotSent	[out, retval] Returns 1 if Telnet Send Failed.

Telnet_SendExpression (NetCard, Expression, MaxWaitMS)

Writes a string to the log file identified by `FileName`.

Syntax

```
HRESULT Telnet_SendExpression(
  [in] long NetCard,
  [in] BSTR Expression,
  [in] long MaxWaitMS
  [out, retval] long* NotSent
);
```

Parameters

Netcard	[in] ID of network port to use.
Expression	[in] Expression to send.
MaxWaitMS	[in] Timeout Value.
NotSent	[out, retval] Returns 1 if Telnet Send Failed.

Telnet_Read (NetCard)

Reads the data in the telnet buffer of the specified network port.

Syntax

```
HRESULT Telnet_Read(
  [in] long NetCard,
  [out, retval] BSTR* Read
);
```

Parameters

Netcard   [in] ID of network port to use.

Read   [out, retval] Returns the data in the buffer.

Telnet_ClearReadBuffer (NetCard)

Clears the telnet buffer of the specified network port.

Syntax

```
HRESULT Telnet_ClearReadBuffer(
  [in] long NetCard,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Netcard   [in] ID of network port to use.

Success   [out, retval] Returns TRUE if buffer was cleared successfully.

Telnet_ReadBufferCount (NetCard)

Counts the number of characters in the buffer.

Syntax

```
HRESULT Telnet_ReadBufferCount (
  [in] long NetCard,
  [out, retval] long* Count
);
```

Parameters

Netcard   [in] ID of network port to use.

Count   [out, retval] Number of characters in the buffer.

Buzzer Methods

Buzzer_On ( )

Emits a beeping noise from inside the WACI.

Syntax

```
HRESULT Buzzer_On (
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Success   [out, retval] Returns TRUE if the buzzer was turned on.

Buzzer_Off ( )

Emits a beeping noise from inside the WACI.

```
HRESULT Buzzer_Off (
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

    Success   [out, retval] Returns TRUE if the buzzer was turned off.

Buzzer_GetState ()

Emits a beeping noise from inside the WACI.

Syntax

```
HRESULT Buzzer_Off (
  [out, retval] long* State
);
```

Parameters

    State   [out, retval] Returns 0 if buzzer is off 1 if buzzer is on.

Logging Methods

Log_Write (FileName, IsExpression, Buffer, MaxSize)

Writes a string to the log file identified by `FileName`.

Syntax

```
HRESULT Log_Write(
  [in] BSTR FileName,
  [in] long IsExpression,
  [in] BSTR Buffer,
  [in] long MaxSize
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

FileName	[in] Name of the log file.
IsExpression	[in] Set to 0 if `Buffer` is a simple string, and 1 if `Buffer` contains an expression.
Buffer	[in] String to write to the log file.
MaxSize	[in] The maximum size the log file can grow to. Once the file reaches this size, then the oldest data is discarded. to make room for the new data.
Success	[out, retval] TRUE if the buffer was cleared, FALSE if buffer failed to clear.

Remarks

The Msg parameter text should be passed in as a URL encoded string. To have a Nul character sent out the serial port, pass a "%00" as part of the Msg string.

Log_ReadFile (FileName, MaxLength)

Writes a string to the specified log file.

Syntax

```
HRESULT Log_ReadFile(
  [in]BSTR FileName,
  [in]long MaxLength,
  [out, retval]BSTR* RetData
);
```

Parameters

FileName   [in] Name of the log file.

MaxLength   [in] Number of bytes to return in `RetData`.

RetData   [out, retval] The buffer to fill with the data from the log file.

Remarks

The number of bytes returned in the `RetData` buffer could be less than the requested `MaxLength` value, depending on the number of bytes contained in the log file.

Log_ClearFile (FileName)

Clears the contents of the named log file.

Syntax

```
HRESULT Log_ClearFile(
  [in]BSTR FileName,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

FileName   [in] Name of the log file.

Success   [out, retval] TRUE if the log file was cleared, FALSE on failure.

Log_FirstFile (Wildcard)

Read the name of the first file in the Logs directory.

Syntax

```
HRESULT Log_ClearFile(
  [in]BSTR Wildcard,
  [out, retval]BSTR* Filename
);
```

Parameters

Wildcard   [in] Filters the list of log names returned. Leave blank for all.

Success   [out, retval] TRUE if the log file was cleared, FALSE on failure.

Log_NextFile (Wildcard)

Read the name of the last file in the Logs directory. Use this to loop through all filenames, and use wildcards to filter results.

Syntax

```
HRESULT Log_ClearFile(
  [in]BSTR Wildcard,
  [out, retval] BSTR* FileName
);
```

Parameters

Wildcard   [in] Filter the list of log names returned. Leave blank for all.

Filename [out, retval] The name of the next file in the directory.

Serial Methods

Serial_GetSettings (Port)

Returns the port settings for the specified serial port. Use the WACI DIP switches or Web interface to set these parameters.

Syntax

```
HRESULT Serial_GetSettings(
  [in] long Port,
  [out, retval] BSTR* Settings
);
```

Parameters

Port   [in] Port number for the serial port. Valid values are 1 and 2.

Settings   [out, retval] Comma separated string representing the current port settings. On error, this parameter is returned as NULL or ““. The format of the string is: baud-rate, parity, stop-bits, RS422-flag, flow-control, e.g. “9600,8,NONE,1,0,HARDWARE”.

Remarks

Valid values for the different settings are:

Baud: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200

Parity: ODD, EVEN, NONE

Stop Bits: 1, 2

Flow Control: HARDWARE, SOFTWARE, NONE

Serial_Send (Port, Msg, MaxWaitMS)

Sends a string to the specified port.

Syntax

```
HRESULT Serial_Send(
  [in] long Port,
  [in] BSTR Msg,
  [in] long MaxWaitMS,
  [out, retval] long* NotSent
);
```

Parameters

Port   [in] Port number for the serial port. Valid values are 1 and 2.

Msg   [in] URL encoded string to be sent to the port.

MaxWaitMS   [in] The number of milliseconds allowed to complete the sending of the serial data before timing out.

NotSent   [out, retval] If an error occurred or the timeout expired, then this is the number of bytes not sent; otherwise, this value is set to 0.

The Msg parameter text should be passed in as a URL encoded string. To have a Nul character sent out the serial port, pass a "%00" as part of the Msg string.

Serial_SendExpression (Port, Expression, Data)

Sends the result of an expression to a serial port.

Syntax

```
HRESULT Serial_SendExpression(
  [in] long Port,
  [in] BSTR Expression,
  [in] long MaxWaitMS,
  [out, retval] long* NotSent
);
```

Parameters

Port	[in] Port number for the serial port. Valid values are 1 and 2.
Expression	[in] Expression to be evaluated. The result of the expression is sent out the serial port.
MaxWaitMS	[in] The number of milliseconds allowed to complete the sending of the serial data before timing out.
NotSent	[out, retval] If an error occurred or the timeout expired, then this is the number of bytes not sent; otherwise, this value is set to 0.

Remarks

The rules applied for `Expression` are the same as those for any expression defined for a Serial Action.

Serial_Read (Port)

Returns the contents of the serial read buffer for the specified serial port.

Syntax

```
HRESULT Serial_Read(
  [in] long Port,
  [out, retval] BSTR* Read
);
```

Parameters

Port	[in] Port number for the serial port. Valid values are 1 and 2.
Read	[out, retval] URL encoded string read from the specified port.

Remarks

The string returned by this method is in URL encoded format. Once the data is read from the buffer, the contents of the serial read buffer are cleared.

Serial_ClearReadBuffer (Port)

Clears the contents of the read buffer for the specified port.

```
HRESULT Serial_ClearReadBuffer(
  [in] long Port,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number for the serial port. Valid values are 1 and 2.

Success   [out, retval] TRUE if the buffer was cleared, FALSE if buffer failed to clear.

Remarks

If you wish to read a specific response to a sent command, then make a call to this function just prior to calling Serial_Send.

Serial_ReadBufferCount (Port)

Returns the number of characters in the read buffer for the specified port. The maximum number of characters stored by the circular buffer is 4K.

Syntax

```
HRESULT Serial_ReadBufferCount(
  [in] long Port,
  [out, retval] long* Count
);
```

Parameters

Port   [in] Port number for the serial port. Valid values are 1 and 2.

Count   [out, retval] Number of characters currently in the receive buffer.

Serial_GetPortCount ()

Returns the number of serial ports supported by the WACI hardware.

Syntax

```
HRESULT Serial_GetPortCount(
  [out, retval] long* NumPorts
);
```

Parameters

NumPorts   [out, retval] Number of serial ports supported by the hardware.

Relay Methods (WACI NX+ Only)

Relay_On (Port)

Switches on the relay at the specified port.

Syntax

```
HRESULT Relay_On(
  [in] long Port,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number for the relay. Valid values are 1, 2, 3, and 4.

Success   [out, retval] TRUE if the relay was switched on, FALSE if the relay failed to switch on.

Remarks

This function energizes the specified relay. The switch "sense" depends on whether a device has been wired to the port's "Normally Open" or "Normally Closed" connector on the connector block.

Relay_Off (Port)

Switches off the relay at the specified port.

Syntax

```
HRESULT Relay_Off(
  [in] long Port,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number for the relay. Valid values are 1, 2, 3, and 4.

Success   [out, retval] TRUE if the relay was switched off, FALSE if the relay failed to switch off.

Remarks

This function de-energizes the specified relay. The switch "sense" depends on whether a device has been wired to the port's "Normally Open" or "Normally Closed" connector on the connector block.

Relay_GetState (Port)

Reports state of relay (on or off) at the specified port.

Syntax

```
HRESULT Relay_GetState(
  [in] long Port,
  [out, retval] long* State
);
```

Parameters

Port   [in] Port number for the relay. Valid values are 1, 2, 3, and 4.

State   [out, retval] 1 if the relay is on, 0 if is off.

Relay_GetPortCount ( )

Returns the number of relays supported by the WACI hardware.

Syntax

```
HRESULT Relay_GetPortCount(
  [out, retval] long* NumPorts
);
```

Parameters

NumPorts   [out, retval] Number of relays supported by the hardware.

Digital I/O Methods (WACI NX+ Only)

Available for the WACI + Only

DIO_Read (Port)

Reads the digital I/O at the specified port.

Syntax

```
HRESULT DIO_Read(
  [in] long Port,
  [out, retval] long* State
);
```

Parameters

Port   [in] Port number for the digital I/O. Valid values are 1, 2, 3, and 4.

State   [out, retval] The state of the DIO port. A voltage of less than 2v on the port returns a 0. A
                  voltage greater than 2v returns a 1.

Remarks

When set as an output, each DIO port can source 1.25mA at 9v.

DIO_OutputMode_Close (Port)

Sets the Digital I/O port to an active low state (approximately 0.3v).

Syntax

```
HRESULT DIO_OutputMode_Close(
  [in] long Port,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number for the digital I/O. Valid values are 1, 2, 3, and 4.

Success   [out, retval] TRUE if the digital I/O port was successfully set low, and FALSE if the port was
                  not set.

Remarks

The voltage is a reference from the GND line of the DIO connector block.

DIO_OutputMode_Open (Port)

Sets the Digital I/O port to an active high state (approximately 9v).

Syntax

```
HRESULT DIO_OutputMode_Open(
  [in] long Port,
  [out, retval] VARIANT_BOOL* Success
);
```

Port   [in] Port number for the digital I/O. Valid values are 1, 2, 3, and 4.

Success   [out, retval] TRUE if the digital I/O port was successfully set high, and FALSE if the port was not set.

Remarks

The voltage is a reference from the GND line of the DIO connector block.

DIO_IsOutput (Port)

Returns whether the Digital I/O line is set for output or input.

Syntax

```
HRESULT DIO_IsOutput(
  [in] long Port,
  [out, retval] long* Result
);
```

Parameters

Port   [in] Port number for the digital I/O. Valid values are 1, 2, 3, and 4.

Result   [out, retval] Returns 1 if the DIO line is set as an output, and 0 if set as an input.

DIO_IsPulledUp (Port)

Used to determine whether a pull-up or pull-down resistor is connected to the input of the specified DIO line.

Syntax

```
HRESULT DIO_IsPulledUp(
  [in] long Port,
  [out, retval] long* Result
);
```

Parameters

Port   [in] Port number for the digital I/O. Valid values are 1, 2, 3, and 4.

Result   [out, retval] If the returned value is 1, then the input port has an internal pull-up resistor connected to it. A 0 indicates that the input line is connected to a pull-down resistor.

Remarks

The value returned is only valid if the DIO port is set to be an input. The pull-up and pull-down resistors are used to prevent the input from floating when no driving output is physically connected to the port.

DIO_GetPortCount ( )

Returns the number of digital I/O ports supported by the WACI hardware.

Syntax

```
HRESULT DIO_GetPortCount(
  [out, retval] long* NumPorts
);
```

Parameters

NumPorts   [out, retval] Number of digital I/O ports supported by the hardware.

A/D Converter Methods (WACI NX+ Only)

Available for WACI NX+ ONLY.

AD_ReadVoltage (Port)

Reads the analog voltage across the specified A/D converter port.

Syntax

```
HRESULT AD_ReadVoltage(
  [in] long Port,
  [out, retval] double* Val
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Val   [out, retval] Voltage across the specified A/D port.

Remarks

The value returned by this call represents the raw voltage applied to the specified port. It will fall between 0 and the value returned by AD_MaxVoltage. By default, the maximum voltage value is 10.0.

AD_ReadDigital (Port)

Returns the integer value (0 to 1023) that is proportional to the voltage applied to the specified port.

Syntax

```
HRESULT AD_ReadDigital(
  [in] long Port,
  [out, retval] long* Val
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Val   [out, retval] An integer value between 0 and 1023

Remarks

The Analog to Digital ports convert voltage values to integer values. The integer value is proportional to the applied voltage. For example, if the port accepts voltages between 0v and 10v, and the digital values range between 0 and 1023, then a voltage of 4v would return a digital value of 409. A value of 5v would return a digital value of 511.

AD_DigitalToVoltage (Port, Digital)

Converts a digital value to an analog voltage value.

```
HRESULT AD_DigitalToVoltage(
  [in] long Port,
  [in] long Digital,
  [out, retval] double* Volts
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Digital   [in] A value between 0 and 1023 (default) to convert to a voltage value.

Volts   [out, retval] A real number (default: 0 to 10.0) representing a voltage level.

Remarks

The value returned by this function depends on the values of the maximum voltage and maximum digital value.
Use `AD_MaxVoltage` and `AD_MaxDigital` to get the true ranges for the digital and voltage values.

AD_MaxVoltage (Port)

Gets the voltage corresponding to the maximum A/D digital value.

Syntax

```
HRESULT AD_MaxVoltage(
  [in] long Port,
  [out, retval] double* MaxVolts
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

MaxVolts   [out, retval] The voltage that must be applied to the specified port to cause AD_ReadDigital
                to return the maximum digital value.

Remarks

When the voltage is applied to the port that meets or exceeds the maximum voltage value returned by this call,
then a read of the digital value of the port will yield the maximum digital value. Use `AD_MaxDigital` to get
the A/D maximum digital value. The default maximum voltage value is 10.0.

AD_MinVoltage (Port)

Gets the voltage corresponding to the maximum A/D digital value.

Syntax

```
HRESULT AD_MaxVoltage(
  [in] long Port,
  [out, retval] double* MaxVolts
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

MaxVolts   [out, retval] The voltage that must be applied to the specified port to cause AD_ReadDigital
                to return the maximum digital value.

Remarks

When the voltage is applied to the port that meets or exceeds the minimum voltage value returned by this call, then a read of the digital value of the port will yield the minimum digital value. Use `AD_MinDigital` to get the A/D minimum digital value. The default minimum voltage value is 1.0.

AD_MaxDigital (Port)

Gets the maximum A/D digital value.

Syntax

```
HRESULT AD_MaxDigital(
  [in] long Port,
  [out, retval] long* MaxDigital
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

MaxDigital   [out, retval] The value returned when the maximum voltage is applied to the port.

Remarks

The maximum digital value is returned when the port's input voltage is at the maximum voltage level. Use `AD_MaxVoltage` to get the A/D maximum voltage level. By default the maximum digital value is 1023.

AD_MinDigital (Port)

Gets the maximum A/D digital value.

Syntax

```
HRESULT AD_MaxDigital(
  [in] long Port,
  [out, retval] long* MaxDigital
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

MaxDigital   [out, retval] The value returned when the maximum voltage is applied to the port.

Remarks

The minimum digital value is returned when the port's input voltage is at the minimum voltage level. Use `AD_MinVoltage` to get the A/D maximum voltage level. By default the minimum digital value is 1.

AD_GetPortCount ( )

Returns the number of Analog to Digital ports supported by the WACI hardware.

Syntax

```
HRESULT AD_GetPortCount(
  [out, retval] long* NumPorts
);
```

NumPorts   [out, retval] Number of A/D input ports supported by the hardware.

AD_SetVoltage (Port, Val)

Sets the output voltage of the specified port.

Syntax

```
HRESULT AD_GetPortCount(
  [in] long Port,
  [in] double Val,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Val [in] The voltage level

Success   [out, retval] Returns true if the change was successful.

AD_SetDigital (Port, Val)

Sets the digital value of the A/D Port.

Syntax

```
HRESULT AD_GetPortCount(
  [in] long Port,
  [in] long Val,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Val [in] Set the digital value of output voltage. Default range is 0 – 1024
                    Range can be set using the AD_MinVoltage() and AD_MaxVoltage()

Success   [out, retval] Returns true if the change was successful.

IR Methods

IR_SendCommand (Port, Group, Command)

Sends an IR command out the specified IR port.

Syntax

```
HRESULT IR_SendCommand(
  [in] long Port,
  [in] BSTR Group,
  [in] BSTR Command,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number of the IR Port of interest. Valid values are 1 and 2.

Group   [in] Name of the file that contains the command.

Command   [in] Name of the command to send.

Success   [out, retval] TRUE if command was successfully sent, and FALSE if the WACI failed to send
                the command.

Remarks

Commands are typically grouped together by device.

IR_SendCommandEx (Port, Group, Command, Sequence, Repeat)

Sends an IR command out the specified IR port.

Syntax

```
HRESULT IR_SendCommandEx(
  [in] long Port,
  [in] BSTR Group,
  [in] BSTR Command,
  [in] BSTR Sequence,
  [in] BSTR Repeat,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number of the IR Port of interest. Valid values are 1 and 2.

Group   [in] Name of the file that contains the command.

Command   [in] Name of the command to send.

Sequence   [in] The sequence of IR commands to send.

Repeat   [in] Number of times to Repeat the IR command.

Success   [out, retval] TRUE if command was successfully sent, and FALSE if the WACI failed to send
                the command.

Commands are typically grouped together by device.

IR_SendData (Port, Sequence, Data)

Converts the supplied Data to a valid IR command and sends the command out the specified IR port.

Syntax

```
HRESULT IR_SendData(
  [in] long Port,
  [in] long Sequence,
  [in] BSTR DATA,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Port   [in] Port number for the A/D converter. Valid values are 1, 2, 3, and 4.

Sequence   [in] Denotes which of the two sequences in the IR data stream should be output. Set to 1 to output the "one time" stream, and 2 for the repeat stream.

DATA   [in] A string containing the IR data to send. The format is defined below.

Success   [out, retval] TRUE if data was successfully sent, FALSE if sending the data failed.

Remarks

Data should be formatted as a set of hex values that represent the pulse widths of the signal to be generated. For example,

```
0000 005C 0000 0004 001C 09C4 0030 03CC 0030 03CC 0034 00D4
```

The sequence is formatted as follows:

Offset	Size (in 16 bit words)	Ident.	Name	Description	Sample
0	1	wFormat	Format.	0100: No carrier 0000: Use carrier	0000
1	1	wFreq	Carrier frequency	The formula below is used to calculate this value.	005C
2	1	wOnce	Once size	The number of on/off pairs in the sequence that is sent only once	0000
3	1	wRepeat	Repeat size	The number of on/off pairs in the repeating sequence	0004
4	2*wOnce	aOnce	Once sequence	The sequence that is sent only once	N/A
4 + 2*wOnce	2*wRepeat	aRepeat	Repeat sequence	The sequence that is repeated	001C 09C4 0030 03CC 0030 03CC 0034 00D4

wFreq Calculation:

wFreq = $4.145146 * 10^6$ / Carrier_Freq

or

Carrier_Freq = $4.145146 * 10^6$ / wFreq

In the above example, the wFreq value was 005C (or 92 decimal). So, the carrier frequency would be
Carrier_Freq = $4.145146 * 10^6$ / 92 = 45 kHz

The data sequences consist of on/off timing pairs, and represent the number of cycles the emitter is turned on and the number of cycles the emitter is turned off (The number of cycles is also known as the pulse width). In the above example, the first pair is `001C:09C4`. This pair turns the emitter on for 28 (`001C`) cycles (or 28*(1/45000) sec = 0.622ms) and off for 2500 (`09C4`) cycles (2500*(1/45000) sec = 55ms).

IR_ListAllGroups ( )

Returns a string listing all groups.

Syntax

```
HRESULT IR_ListAllGroups(
  [out, retval] BSTR* List
);
```

List   [out, retval] String containing a list of all group names. Commas separate the group names.

Remarks

A group typically represents a device, or more specifically, a group of commands for a device. Each group is stored in its own .WIR file.

IR_ListAllCommandsInGroup (Group)

Returns a string listing all commands in a specified group.

Syntax

```
HRESULT IR_ListAllCommandsInGroup(
  [in] BSTR Group,
  [out, retval] BSTR* List
);
```

Parameters

Group   [in] Name of group to examine.

List   [out, retval] String containing a comma separated list of all commands in the specified group.

Remarks

If a group contains no commands, then the returned string is "".

IR_ListAllCommands ( )

Returns an array containing all commands from all groups.

Syntax

```
HRESULT IR_ListAllCommands(
  [out, retval] VARIANT* Array
);
```

Parameters

Array   [out, retval] Array containing all commands from all groups

Remarks

The returned array is an array of strings. Each string is a comma-separated list of commands. The first entry in the each string is the name of the group. The following entries in of the string are the commands, e.g.

`DVDPLAYER,Play,Stop,Fwd,Rev`

In this example, `DVDPLAYER` is the group, and the remaining entries are the commands.

IR_GetGroupMake (Group)

Returns the make of a device. The device is specified by its group name.

```
HRESULT IR_GetGroupMake(
  [in] BSTR Group,
  [out, retval] BSTR* Make
);
```

Parameters

Group   [in] Name of the group representing the specified device.

Make   [out, retval] String containing the make of the device.

Remarks

This function returns the string that follows the "Make:" label in the .WIR file associated with the specified Group. The Make for the specified Group can also be changed in the IR Diagnostics page.

IR_GetGroupModel (Group)

Returns the model name or number of a device. The device is specified by its group name.

Syntax

```
HRESULT IR_GetGroupModel(
  [in] BSTR Group,
  [out, retval] BSTR* Model
);
```

Parameters

Group   [in] Name of group representing the specified device.

Model   [out, retval] String containing the model name or number of the device.

Remarks

This function returns the string that follows the "Model:" label in the .WIR file associated with the specified Group. The Model for the specified Group can also be changed in the IR Diagnostics page.

IR_GetGroupRemote (Group)

Returns the model of the remote that controls a device. The device is specified by its group name.

Syntax

```
HRESULT IR_GetGroupRemote(
  [in] BSTR Group,
  [out, retval] BSTR* Remote
);
```

Parameters

Group   [in] Name of the group representing the specified device.

Remote   [out, retval] String containing the remote's model number.

Remarks

This function returns the string that follows the "Model:" label in the .WIR file associated with the specified Group. The Model for the specified Group can also be changed in the IR Diagnostics page.

IR_GetGroupComment (Group)

Returns any stored comments about a device. The device is specified by its group name.

Syntax

```
HRESULT IR_GetGroupComment(
  [in] BSTR Group,
  [out, retval] BSTR* Comment
);
```

Parameters

Group   [in] Name of group representing the specified device.

Comment   [out, retval] String containing comments about the specified device.

Remarks

This function returns the string that follows the "Comment:" label in the .WIR file associated with the specified Group. The Comment for the specified Group can also be changed in the IR Diagnostics page.

IR_GetPortCount ( )

Returns the number of IR output ports supported by the WACI hardware.

Syntax

```
HRESULT IR_GetPortCount(
  [out, retval] long* NumPorts
);
```

Parameters

NumPorts   [out, retval] Number of IR ports supported by the hardware.

Event Manager Methods

IsEventManagerEnabled ( )

Returns whether the Event Manager is active. When active, the Event Manager monitors the system hardware and triggers Events to run.

Syntax

```
HRESULT IsEventManagerEnabled(
  [out, retval]long* Enabled
);
```

Parameters

Enabled   [out, retval] Returned as TRUE if the Event Manager is enabled and active, and returns FALSE when the Event Manager has been disabled.

Remarks

To enable or disable the Event Manager, call the `EnableEventManager` function.

EnableEventManager (Enable)

Enables or disables the monitoring of the hardware by the Event Manager. When disabled, the Event Manager will not execute any Events.

Syntax

```
HRESULT EnableEventManager(
  [in]VARIANT_BOOL Enable,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

Enable   [in] Set to TRUE to enable the Event Manager, and FALSE to disable the Event Manager. Disabling the Event Manager will stop the processing of all Events and Actions.

Success   [out, retval] TRUE if the Event Manager was successfully enabled or disabled, and FALSE if the state of the Event Manager didn't change.

Remarks

To determine whether the Event Manager is enabled or disabled, call the `IsEventManagerEnabled` function.

WaitOnChangeEvent (ChangeMask, TimeOut)

Use this function to be notified of a change in the WACI's hardware or change in a Variable.

NOTE: Calling this function will halt the web server while the wait is being processed.

Syntax

```
HRESULT WaitOnChangeEvent(
  [in]long ChangeMask,
  [in]long TimeOut,
  [out, retval]long* Result
);
```

Parameters

ChangeMask   [in] A bit mask that defines which Events to watch for.

TimeOut   [in] Number of milliseconds to wait for an Event. If the timeout expires, the function returns with Result set to 0.

Result   [out, retval] A bit-mask representing the Event(s) that cause the wait to terminate.

Remarks

This function will not return until one of the specified Events has happened. To be notified of a change, an Event of the hardware type your interested in must already exist.

Or the following bits together to set the `ChangeMask` as desired. The `ChangeMask` bits are defined as follows:

Bit Value	Bit #	Description
0x00000001	Bit 0	One of the Variables changed
0x00000002	Bit 1	Serial data received on one of the serial ports
0x00000004	Bit 2	One of the digital input ports changed state
0x00000008	Bit 3	The input voltage on one of the A/D inputs changed

The Result parameter will contain either a mask of the triggered events, or one of the following values:

Error Value	Description
0xFFFFFFFF (-1)	Error occurred when trying to create the wait event.
0x00010000 (65536)	The TimeOut expired. No events triggered.
0x00010001 (65537)	The Wait aborted due to some error.

Event Methods

Most of the Event methods use the Event's ID to retrieve information about an Event. The Event's ID is set when an Event is created, and is unique to the Event. The ID is re-issued when the system is powered up or the reset button is pressed.

The index number of the Event within the Event list could change, depending on whether new Events are added or existing Events are deleted.

AddEvent (Name, Type, Concurrent, Source, Match)

Create a new Event with the specified parameters.

Syntax

```
HRESULT AddEvent(
  [in] BSTR Name,
  [in] long Type,
  [in] long Concurrent,
  [in] BSTR Source,
  [in] BSTR Match,
  [out, retval] long* EventId
);
```

Parameters

Name	[in] Name of the new Event.
Type	[in] Type of the Event, e.g. Serial (see GetEventType (EventId), page 146 for a list of types).
Concurrent	[in] Set to 1 to execute Actions concurrently, and 0 otherwise.
Source	[in] Port, Variable, etc. to be used for the source of the Event's trigger (see GetEventSource (EventId), page 145 for details on valid Source values).
Match	[in] Value to match to trigger the Event.
EventId	[out, retval] ID of the new Event.

Remarks

Call this function to add a new Event. After making this call, a call to SetEventSchedule is required for all Events of type Clock or Timer.

A call to SetEventOption should also to be made using the newly returned EventId to set or clear the expression flag for the Event's match value.

AddRemoteEvent (Client, EventRecord)

Create a new temporary Event on a remote WACI.

Syntax

```
HRESULT AddRemoteEvent(
  [in]BSTR Client,
  [in]BSTR EventRecord,
  [out, retval]long* EventId
);
```

Parameters

Client　　　　　　[in] IP address or host name of the remote WACI.

EventRecord　　　[in] Text string that contains the information about the Event record (see **Error! Reference source not found.**, page **Error! Bookmark not defined.**) for the format of the record)

EventId　　　　　[out, retval] ID of the Event on the remote host.

Remarks

This function is called by the Event Manager when a local Event's host_name field has been filled in.

The remote Event will timeout after 1 minute, and will be deleted from the remote WACI, unless a call to refresh the Event is made before times out. To refresh the Event, make another call to AddRemoteEvent with the same Event record.

CloneEventById (EventId, CloneActions, NewName)

Clone (duplicate) an Event specified by its ID.

Syntax

```
HRESULT CloneEventById(
  [in] long EventId,
  [in] long CloneActions,
  [in] BSTR NewName,
  [out, retval] long* CloneID
);
```

Parameters

EventId　　　　　[in] ID of the Event to modify.

CloneActions　　[in] Duplicate all the Event's Actions too.

NewName　　　　[in] Name of the new duplicate Event.

CloneID　　　　　[out, retval] ID of the new duplicate Event.

Remarks

The name of the newly cloned Event will be set to "%s Copy", where the %s is replaced by the name of the Event being cloned. If a copy of the Event already exists, then an incrementing number is appended to the end, e.g. "MyEvent Copy3".

DeleteEventById (EventId)

Delete an Event specified by its ID.

Syntax

```
HRESULT DeleteEventById(
  [in] long EventId,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId　　　　　[in] ID of the Event to delete.

Success　　　　　[out, retval] TRUE if the specified Event was successfully deleted, FALSE if deleting the Event failed.

135

Calling this function deletes the Event as well as all Actions belonging to the Event.

DeleteEventByName (Name)

Delete an Event specified by its name.

Syntax

```
HRESULT DeleteEventByName(
  [in] BSTR Name,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Name   [in] Name of the Event to delete.

Success   [out, retval] TRUE if the specified Event was successfully deleted, FALSE if deleting the
                Event failed.

Remarks

Calling this function deletes the Event as well as all Actions belonging to the Event.

GetEventByIdx (Idx)

Returns the Event ID for a specified index number.

Syntax

```
HRESULT GetEventByIdx(
  [in] long Idx,
  [out, retval] long* EventId
);
```

Parameters

Idx   [in] The index of the Event in the Event list

EventId   [out, retval] Event ID of the specified Event.

Remarks

GetEventByIdx is normally used as part of a loop for getting the list of created Events. Use
GetEventCount to get the number of Events. Use GetEventByIdx to get the ID of the next Event in the
list. Then use the ID and the remaining Event functions to retrieve the information you are interested in, e.g.
Name, Type, Actions, etc.

The index numbers start at 0 and stop at Count-1, where Count is gotten with GetEventCount.

If -1 is returned, then index is out of range. If -2 is returned, then the Event represented by Idx is in the
process of being deleted.

GetEventByName (Name)

Returns an Event ID for an Event with a specified name.

Syntax

```
HRESULT GetEventByName(
  [in] BSTR Name,
  [out, retval] long* EventId
);
```

Parameters

Name   [in] Name of the Event to identify.

EventId   [out, retval] Event ID of the specified Event.

Remarks

The ID returned by `GetEventByName` is used by the Event functions to retrieve information about the named Event, e.g. Name, Type, Actions, etc.

If –1 is returned, then the Event specified by `Name` doesn't exist. If –2 is returned, then the Event represented by `Name` is in the process of being deleted.

GetEventConcurrent (EventId)

GetEventConcurrent returns a value signifying whether an Event's Actions should run concurrently or sequentially.

Syntax

```
HRESULT GetEventConcurrent(
  [in] long EventId,
  [out, retval] long* Concurrent
);
```

Parameters

EventId   [in] ID for the specified Event.

Concurrent [out, retval] 1 if Actions should be executed concurrently, 0 if Actions should be executed sequentially.

Remarks

When Actions are set to execute sequentially, the next Action in the list will not execute until the previous Action has completely executed. So, if the first Action is set as one that executes an infinite number of times, the second Action will never execute. Use the Up/Dn hyperlinks in the Event Manager Diagnostics screen's Action list to change the order of the Actions' execution.

When Actions execute concurrently, the Event Manager determines the next Action to run by calculating its next run time. The Event Manager uses the Action's Delay and Repeat Every values to make this calculation. The Action with the earliest next run time is run first. If two Actions have the same run time, the first Action in the Action list will be executed first.

This flag can also be obtained user `GetEventOption`.

GetEventCount ()

Returns the total number of Events.

```
HRESULT GetEventCount(
  [out, retval] long* NumEvents
);
```

Parameters

NumEvents   [out, retval] Total number of Events in the system.

Remarks

This function is used in conjunction with GetEventByIdx to get a list of the available Events.

GetEventGroup (EventId)

Returns the group name for the Event with a specified ID.

Syntax

```
HRESULT GetEventGroup(
  [in]long EventId,
  [out, retval]BSTR* Group
);
```

Parameters

EventId   [in] ID of the Event.

Group   [out, retval] A returned string filled in with the group for the specified Event.

GetEventNetCard (EventId)

Returns the group name for the Event with a specified ID.

Syntax

```
HRESULT GetEventGroup(
  [in]long EventId,
  [out, retval]BSTR* Netcard
);
```

Parameters

EventId   [in] ID of the Event.

Netcard   [out, retval] The ID of the Network port the event is assigned to.

Remarks

If the NetCard is 0, that means it can be triggered by any network port.

GetEventHost (EventId)

Returns the remote host name of the Event with a specified ID.

Syntax

```
HRESULT GetEventHost(
  [in]long EventId,
  [out, retval]BSTR* Host
);
```

EventId   [in] ID for the specified Event.

Host   [out, retval] The name of the remote WACI that the Event is monitoring.

Remarks

Use `SetEventHost` to set the name of the remote host. When a remote host name is specified, then the local Event is triggered on a change in the remote WACI's hardware or a change in one of the remote WACI's Variables.

GetEventIds ( )

Returns an array of all of the Event IDs.

Syntax

```
HRESULT GetEventIds(
  [out, retval]VARIANT* EventIds
);
```

Parameters

EventIds   [out, retval] The returned array of Event IDs.

Remarks

Use `GetEventIds` to get the complete list of Event IDs stored by the Event Manager. This function is more efficient than calling `GetEventCount` and `GetEventByIdx`.

GetEventIdxById (EventId)

Returns the index into the Event list for an Event with a specified ID.

Syntax

```
HRESULT GetEventIdxById(
  [in] long EventId,
  [out, retval] long* Idx
);
```

Parameters

EventId   [in] ID for the specified Event.

Idx   [out, retval] The index of the Event in the Event list.

Remarks

The index returned will be in the range `0` to `Count-1`, where `Count` is the value returned by `GetEventCount`.

`-1` is returned if the Event with the specified ID is not found.

GetEventInfo (EventId)

Returns an array filled with the data for an Event.

139

```
HRESULT GetEventInfo(
  [in]long EventId,
  [out, retval] VARIANT* Info
);
```

Parameters

EventId   [in] ID of the Event of interest.

Info   [out, retval] An array of data values that define the Event.

Remarks

The `Info` array is filled with the following:

Element	Field	Type
Info[0]	Name	BSTR
Info[1]	Type	long
Info[2]	Options	unsigned long
Info[3]	Source	BSTR
Info[4]	Match	BSTR
Info[5]	Host	BSTR
Info[6]	Group	BSTR

Use `GetEventIds` or `GetEventByIdx` to get the ID of the Event.

GetEventMatch (EventId)

Returns the expression to be evaluated or data to be compared when the Event is notified of a change in data

Syntax

```
HRESULT GetEventMatch(
  [in] long EventId,
  [out, retval] BSTR* Match
);
```

Parameters

EventId   [in] ID for the specified Event.

Match   [out, retval] The value that incoming data should be compared against to determine whether the Event should be triggered.

Remarks

The `Match` value is either an expression or simple text value. How the `Match` value is evaluated depends on the setting of "Expression" in the edit page for this Event.

If `Match` is an expression, then it is evaluated whenever there is a change in the Event's source port, Variable, etc. When the expression evaluates to non-zero, the Event is scheduled to run.

When `Match` is not an expression, then the incoming data is compared against the Match value. If the `Match` matches the data, then the Event is triggered to run.

GetEventName (EventId)

Returns the name of the Event with a specified ID.

Syntax

```
HRESULT GetEventName(
  [in] long EventId,
  [out, retval] BSTR* Name
);
```

Parameters

EventId   [in] ID for the specified Event.

Idx   [out, retval] Name of the specified Event.

GetEventOption (EventId, OptionType)

Returns the value of a specified option for a specified Event.

Syntax

```
HRESULT GetEventOption(
  [in] long EventId,
  [in] long OptionType,
  [out, retval] long* OptionVal
);
```

Parameters

EventId   [in] ID for the specified Event.

OptionType   [in] Type of option for the Event.

OptionVal   [out, retval] Value of the specified option for the Event.

Remarks

Pass one of the following values into `OptionType`:

Value	Option	`OptionVal` returned
1	Concurrent	0 – Event's Actions are executed serially 1 – Event's Actions are executed concurrently
2	Expression	0 – The trigger value is a simple text string 1 – The trigger value is an expression
3	Disable	0 – The Event is enabled 1 – The Event is disabled

GetEventSchClockType (EventId)

Returns the type of schedule used by a time based Event.

```
HRESULT GetEventSchClockType(
  [in] long EventId,
  [out, retval] long* ClockType
);
```

Parameters

EventId   [in] ID for the specified Event.

ClockType   [out, retval] Is set to 1, if the Event is a scheduled Event, and 0, if the Event is a timer based
                Event.

Remarks

This function is valid only for Clock and Timer Events.

Use this function to determine if the Event is based on a simple timer, or is using a more complicated schedule.

GetEventSchRecurType (EventId)

Returns the type of schedule used by the specified Event.

Syntax

```
HRESULT GetEventSchRecurType(
  [in] long EventId,
  [out, retval] long* RecurType
);
```

Parameters

EventId   [in] ID for the specified Event.

RecurType   [out, retval] Type of schedule to apply
                Valid values for `RecurType` are:

Value	Description
0	Once
1	Daily
2	Weekly
3	Monthly
4	Yearly

Remarks

This function is valid for Clock Events only.

GetEventSchMaskOrDay (EventId)

When the Event is set as a weekly Event, then this function returns a mask that defines the days of the week.
Yearly and monthly Event schedule types use this value for specifying the day of the month.

Syntax

```
HRESULT GetEventSchMaskOrDay(
  [in] long EventId,
  [out, retval] long* MaskDay
);
```

Parameters

EventId   [in] ID for the specified Event.

MaskDay   [out, retval] Day of the month, or a mask of bits for the days of the week.

MaskDay values for weekly Events:

Value	Bit #	Day
0x00000001 (1)	0	Sunday
0x00000002 (2)	1	Monday
0x00000004 (4)	2	Tuesday
0x00000008 (8)	3	Wednesday
0x00000010 (16)	4	Thursday
0x00000020 (32)	5	Friday
0x00000040 (64)	6	Saturday

Remarks

This function is valid only for Clock Events.

GetEventSchMonth (EventId)

Returns the month value for the Event's schedule.

Syntax

```
HRESULT GetEventSchMonth(
  [in] long EventId,
  [out, retval] long* Month
);
```

Parameters

EventId   [in] ID for the specified Event.

Month   [out, retval] Integer representing the scheduled month of the year. Values will range from 1 to 12.

Remarks

This function is valid only for Clock Events.

The month value is only used for yearly Event schedules.

GetEventSchYear (EventId)

Returns the year part of the Event's schedule.

```
HRESULT GetEventSchYear(
  [in] long EventId,
  [out, retval] long* Year
);
```

Parameters

EventId   [in] ID for the specified Event.

Year   [out, retval] Integer representing the scheduled year (for example, "1984" or "2003").

Remarks

This function gets a value that is not used by the Event Manager.

GetEventSchRecurEveryN (EventId)

Returns the "Recur every" value of the Event's schedule.

Syntax

```
HRESULT GetEventSchRecurEveryN(
  [in] long EventId,
  [out, retval] long* EveryN
);
```

Parameters

EventId   [in] ID for the specified Event.

EveryN   [out, retval] Repeat every N periods. The period depends on the type of schedule, e.g. Weekly, Monthly, etc.

Remarks

This function is valid only for Clock Events.

The "Recur Every" value (EveryN) is used by all the Event schedules, except one. The "Once" schedule does not repeat; therefore, it does not need the EveryN value.

GetEventSchHour (EventId)

Returns the hour that the Event is to be triggered.

Syntax

```
HRESULT GetEventSchHour(
  [in] long EventId,
  [out, retval] long* Hour
);
```

Parameters

EventId   [in] ID for the specified Event.

Hour   [out, retval] Integer representing the scheduled hour. Values range from 0 to 23.

Remarks

This function is valid only for Clock and Timer Events.

GetEventSchMinute (EventId)

Returns the minute that the Event is to be triggered.

Syntax

```
HRESULT GetEventSchMinute(
  [in] long EventId,
  [out, retval] long* Minute
);
```

Parameters

EventId   [in] ID for the specified Event.

Minute   [out, retval] Integer representing the scheduled minute. Values range from 0 to 59.

Remarks

This function is valid only for Clock and Timer Events.

GetEventSchSecond (EventId)

Returns the second that the Event is to be triggered.

Syntax

```
HRESULT GetEventSchSecond(
  [in] long EventId,
  [out, retval] long* Second
);
```

Parameters

EventId   [in] ID for the specified Event.

Second   [out, retval] Integer representing the scheduled second. Values range from 0 to 59.

Remarks

This function is valid only for Clock and Timer Events.

GetEventSource (EventId)

The port, Variable, or clock that is monitored by the specified Event

Syntax

```
HRESULT GetEventSource(
  [in] long EventId,
  [out, retval] BSTR* Source
);
```

Parameters

EventId   [in] ID for the specified Event.

Source   [out, retval] A string filled with the name of the Variable, port, or clock.

An Event monitors its Source for changes. When new data arrives, or the data changes, the Event is notified. The setting of the Event's match value determines whether the Event is signaled to execute on this change in data.

The value of `Source` depends on the type of Event.

Event Type	Value of `Source` string
Clock, Timer, and Startup	"RTC"
Variable	Name of the Variable
Serial, DIO, A/D, Relay, and IR	"Port%d", where %d represents a number
Telnet	IP address or host name of the Telnet server to connect to.

GetEventType (EventId)

Returns the type of the Event with a specified ID.

Syntax

```
HRESULT GetEventType(
  [in] long EventId,
  [out, retval] long* Type
);
```

Parameters

EventId	[in] ID for the specified Event.
Type	[out, retval] Type of the indicated Event.

Remarks

`Type` is returned as one of the following values:

Value	Type
0	Unknown
1	Scheduled
2	Timer
3	Variable
4	Serial Input
5	Digital Input
6	A/D Input
7	Startup
8	Telnet

SetEventById (EventId, Name, Type, Concurrent, Source, Match)

Sets the basic information about an existing Event.

```
HRESULT SetEventById(
  [in] long EventId,
  [in] BSTR Name,
  [in] long Type,
  [in] long Concurrent,
  [in] BSTR Source,
  [in] BSTR Match,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID for the specified Event.
Name	[in] Name of the specified Event.
Type	[in] Type of the Event, e.g. Serial (see GetEventType (EventId), page 146 for a list of types).
Concurrent	[in] Set to 1 to execute Actions concurrently, and 0 otherwise.
Source	[in] Port, Variable, etc. to be used for the source of the Event's trigger (see GetEventSource (EventId), page 145 for a list of valid values for `Source`).
Match	[in] Value to match to trigger the Event. If the Match value is an expression, then the expression will need to evaluate to 1 to cause the Event to trigger.
Success	[out, retval] TRUE if specified Event was successfully set, FALSE if setting the Event failed.

Remarks

Call this function to set up an Event. After making this call, a call to SetEventSchedule is required for all Events of type Clock or Timer.

A call to SetEventOption should also to be made to set or clear the expression flag for the Event's match value.

SetEventByName (Name, Type, Concurrent, Source, Match)

Using the name of an existing Event, this function sets the basic information about it.

Syntax

```
HRESULT SetEventByName(
  [in] BSTR Name,
  [in] long Type,
  [in] long Concurrent,
  [in] BSTR Source,
  [in] BSTR Match,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Name	[in] Name of the Event to set the information for.
Type	[in] Type of Event, e.g. Clock (see GetEventType (EventId), page 146 for a list of types).
Concurrent	[in] Set to 1 to execute Actions concurrently, and 0 otherwise
Source	[in] Port, Variable, etc. to be used as the source of the Event's trigger (see GetEventSource (EventId), page 145 for valid values for `Source`).

Match	[in] Value to match to trigger the Event. If the Match value is an expression, then the expression will need to evaluate to 1 to cause the Event to trigger.
Success	[out, retval] TRUE if specified Event was successfully set, FALSE if setting the Event failed.

Remarks

Call this function to set up an Event. After making this call, a call to `SetEventSchedule` is required for all Events of type Clock or Timer.

If Match is an expression, then a call to `SetEventOption` will also have to be made to set the expression flag for the Event.

SetEventNetCard (EventId, NetCard)

Set the network/telnet event to monitor on a certain network port.

Syntax

```
HRESULT SetEventByName(
  [in] long EventId,
  [in] long NetCard,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] Name of the Event to set the information for.
NetCard	[in] The ID of the network port the event should monitor
Success	[out, retval] TRUE if specified Event was successfully set, FALSE if setting the Event failed.

Remarks

If NetCard is set to 0, the event will monitor all network ports.

SetEventOption (EventId, OptionType, OptionVal)

Modify the option settings of a specified Event.

Syntax

```
HRESULT SetEventOption(
  [in] long EventId,
  [in] long OptionType,
  [in] BSTR OptionVal,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID of the Event to modify.
OptionType	[in] Integer specifying which option to modify.
OptionVal	[in] New value for the option.
Success	[out, retval] TRUE if specified option was successfully set, FALSE if setting the option failed.

Remarks

Pass one of the following values into `OptionType`:

148

Value	Option	`Set OptionVal to`
1	Concurrent	0 – Event's Actions will execute serially 1 – Event's Actions will execute concurrently
2	Expression	0 – The trigger value is a simple text string 1 – The trigger value is an expression
3	Disable	0 – The Event is enabled 1 – The Event is disabled

If both `SetEventSchedule` and `SetEventOption` need to be called for an Event, then `SetEventOption` should be called before `SetEventSchedule`.

SetEventSchedule (EventId, Recur, MaskOrDay, Month, Year, RecurEveryN, Hour, Minute, Second)

Modify the schedule settings of a specified Event.

Syntax

```
HRESULT SetEventSchedule(
   [in] long EventId,
   [in] long Recur,
   [in] long MaskOrDay,
   [in] long Month,
   [in] long Year,
   [in] long RecurEveryN,
   [in] long Hour,
   [in] long Minute,
   [in] long Second,
   [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId   [in] ID of the Event to modify the schedule of.

Recur   [in] The recurrence pattern used for this schedule. .

Recur value	Description
0	Once
1	Daily
2	Weekly
3	Monthly
4	Yearly

MaskOrDay   [in] Mask of days or specific day of the month.

MaskDay values for weekly Events:

Value	Bit #	Day
0x00000001 (1)	0	Sunday
0x00000002 (2)	1	Monday
0x00000004 (4)	2	Tuesday
0x00000008 (8)	3	Wednesday
0x00000010 (16)	4	Thursday
0x00000020 (32)	5	Friday
0x00000040 (64)	6	Saturday

Month   [in] Month setting for the schedule. An integer between 1 and 12. This parameter is used only for yearly schedules.

Year   [in] Year setting for the schedule. An integer such as "1984" or "2001"

150

RecurEveryN	[in] Number of periods between successive triggers of the Event.
Hour	[in] Hour setting for the schedule. An integer between 0 and 23.
Minute	[in] Minute setting for the schedule. An integer between 0 and 60.
Second	[in] Second setting for schedule. An integer between 0 and 60.
Success	[out, retval] TRUE if schedule was successfully set, FALSE if setting the schedule failed.

Remarks

This function is valid only for Clock and Timer Events.

If both SetEventSchedule and SetEventOption need to be called for an Event, then SetEventOption should be called before SetEventSchedule.

SortEvents (SortType, Direction)

Sorts the list of Events by the sort type and direction.

Syntax

```
HRESULT SortEvents(
  [in] long SortType,
  [in] long Direction,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

SortType	[in] Type of sort to apply to the list.
Direction	[in] Direction of the sort: ascending (1) or descending (0).
Success	[out, retval] TRUE if the Events are successfully sorted, FALSE if sorting the Events failed.

Remarks

This function is no longer supported.

TriggerEventByName (Name, Time, Source, Data)

Schedules an Event to be executed.

Syntax

```
HRESULT TriggerEventByName(
  [in]BSTR Name,
  [in]BSTR Time,
  [in]BSTR Source,
  [in]BSTR Data,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

Name	[in] The name of the Event to execute.
Time	[in] The current system time.
Source	[in] What port, Variable, etc. changed to cause the Event to need triggering.
Data	[in] The data received that caused the trigger.

151

Success   [out, retval] TRUE if the Events are successfully sorted, FALSE if sorting the Events failed.

Remarks

An Action executed by the triggered Event has access to the `Data` value by using the `\g` escape sequence within the Action's `output` value.

Action Methods

AddActionByEvId (EventId, Name, Type, Delay, DutyCycle, StopAfter, Port, Output)

Creates an Action for an Event, specified by the Event ID.

Syntax

```
HRESULT AddActionByEvId(
  [in] long EventId,
  [in] BSTR Name,
  [in] long Type,
  [in] double Delay,
  [in] double DutyCycle,
  [in] long StopAfter,
  [in] BSTR Port,
  [in] BSTR Output,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID of the Event to add the Action to.
Name	[in] Name to give the new Action.
Type	[in] Type of Action, e.g. Serial, Variable, etc.
Delay	[in] Number of seconds to delay before sending `Output` to `Port`. This is a real number, and non-integer values are allowed.
DutyCycle	[in] Number of seconds to delay between each successive execution of the Action. This is a real number, and non-integer values are allowed.
StopAfter	[in] Number of times to repeat the Action. Set to −1 to repeat infinitely.
Port	[in] Location where the output is sent.
Output	[in] Data that should be output.
Success	[out, retval] TRUE if a new Action was created with the specified settings, FALSE if creating the Action failed.

Remarks

Action types that the WACI supports, depend on the device's hardware capabilities (see GetActionType (ActionId), page 161 for valid type values).

The value of `Port` represents where the Action is going to send its output (see GetActionPort (ActionId), page 160 for details on valid port values).

The value for `Output` depends on the value of `Type` (see GetActionOutput (ActionId), page 159 for more information on valid values for `Output`).

AddActionByEvName (EventName, Name, Type, Delay, DutyCycle, StopAfter, Port, Output)

Creates an Action for an Event, specified by the Event name.

Syntax

```
HRESULT AddActionByEvName(
  [in] BSTR EventName,
  [in] BSTR Name,
  [in] long Type,
  [in] double Delay,
  [in] double DutyCycle,
  [in] long StopAfter,
  [in] BSTR Port,
  [in] BSTR Output,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventName   [in] ID the Event to add the Action to.

Name   [in] Name to give the new Action.

Type   [in] Type of Action, e.g. Serial, Variable, etc.

Delay   [in] Number of seconds to delay before sending `Output` to `Port`. This is a real number, and non-integer values are allowed.

DutyCycle   [in] Number of seconds to delay between each successive executions of the Action. This is a real number, and non-integer values are allowed.

StopAfter   [in] Number of times to repeat the Action. Set to -1 to repeat infinitely.

Port   [in] Location where the output is sent.

Output   [in] The value of this parameter depends on the value of `Type`.

Success   [out, retval] TRUE if specified Action settings were created, FALSE if creating the Action failed.

Remarks

Action types that the WACI supports, depend on the device's hardware capabilities (see GetActionType (ActionId), page 161 for valid type values).

The value of `Port` represents where the Action is going to send its output (see GetActionPort (ActionId), page 160 for details on valid port values).

The value for `Output` depends on the value of `Type` (see GetActionOutput (ActionId), page 159 for more information on valid values for `Output`).

DeleteActionById (ActionId)

Delete an Action with the given ID.

```
HRESULT DeleteActionById(
  [in] long ActionId,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

ActionId	[in] ID of the Action to delete.
Success	[out, retval] TRUE if the specified Action was successfully deleted, FALSE if deleting the Action failed.

Remarks

Using the Action's ID, this function locates the owning Event and removes the Action from the Event's list of Actions.

DeleteActionByIdx (EventId, Idx)

Delete an Action from the specified Event's list of Actions.

Syntax

```
HRESULT DeleteActionByIdx(
  [in] long EventId,
  [in] long Idx,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID of the Event that owns the Action.
Idx	[in] Index into the Event's list of Actions that identifies which Action to delete.
Success	[out, retval] TRUE if the specified Action was successfully deleted, FALSE if deleting the Action failed.

DeleteActionByName (EventId, Name)

Delete an Action specified by its Event ID and Action name.

Syntax

```
HRESULT DeleteActionByName(
  [in] long EventId,
  [in] BSTR Name,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID of the Event that owns the Action to delete.
Name	[in] Name of Action to delete.
Success	[out, retval] TRUE if the specified Action was successfully deleted, FALSE if deleting the Action failed.

GetActionByIdx (EventId, Idx)

Returns the Action ID for a specified Event ID and Action index number.

Syntax

```
HRESULT GetActionByIdx(
  [in] long EventId,
  [in] long Idx,
  [out, retval] long* ActionId
);
```

Parameters

EventId   [in] ID of the Event that contains the Action.

Idx   [in] Index number for the Action.

ActionId   [out, retval] Returned ID of the specified Action.

Remarks

Use this function to get the list of Actions associated with a particular Event. Each Event can contain up to 16 Actions.

GetActionByName (EventId, Name)

Returns the Action ID for a specified Event ID and Action name.

Syntax

```
HRESULT GetActionByName(
  [in] long EventId,
  [in] BSTR Name,
  [out, retval] long* ActionId
);
```

Parameters

EventId   [in] ID of specified Event.

Name   [in] Name of the Action

ActionId   [out, retval] Action ID of the named Action.

Remarks

The name comparison is case sensitive, and requires an exact match. Wild cards are not supported.

GetActionCount (EventId)

Returns the total number of Actions associated with an Event.

Syntax

```
HRESULT GetActionCount(
  [in] long EventId,
  [out, retval] long* NumActions
);
```

Parameters

EventId   [in] ID of the Event.

NumActions   [out, retval] Total number of Actions owned by specified Event.

GetActionDelay (ActionId)

Returns the "Delay before start" value for the specified Action.

Syntax

```
HRESULT GetActionDelay(
  [in] long ActionId,
  [out, retval] double* Delay
);
```

Parameters

ActionId   [in] ID of the Action to examine.

Delay   [out, retval] Number of seconds to delay. A millisecond is represented as 0.001 seconds.

Remarks

For Actions that execute concurrently and for the first Action in the execution list, the value returned by this function is the number of seconds that the Event Manager will wait after the owning Event has been triggered before executing the Action.

For Actions that execute serially, the value is the amount of time to wait after the previous Action completed.

GetActionDutyCycle (ActionId)

Returns the number of seconds to wait between successive executions of an Action.

Syntax

```
HRESULT GetActionDutyCycle(
  [in] long ActionId,
  [out, retval] double* Duty Cycle
);
```

Parameters

ActionId   [in] ID of the Action to examine.

DutyCycle   [out, retval] Number of seconds between successive executions.

Remarks

The `DutyCycle` returned is also known as the period of the Action. A serial Action with a `DutyCycle` of 1 second will send the output string out the serial port once every second.

GetActionHost (ActionId)

Returns the name of the host that the Action is to be run on.

Syntax

```
HRESULT GetActionHost(
  [in]long ActionId,
  [out, retval]BSTR* Host
);
```

Remarks

A return value of "" indicates that the Action will be executed locally

GetActionIds (EventId)

Returns the IDs of the Actions owned by the specified Event.

Syntax

```
HRESULT GetActionIds(
  [in]long EventId,
  [out, retval]VARIANT* ActionIds
);
```

Parameters

EventId   [in] ID for the owning Event.

ActionIds   [out, retval] The returned array of Action IDs.

Remarks

Use `GetActionIds` to get the complete list of Action IDs owned by an Event. This function is more efficient than calling `GetActionCount` and `GetActionByIdx`.

GetActionInfo (ActionId)

Returns an array filled with the data for an Action.

Syntax

```
HRESULT GetActionInfo(
  [in]long ActionId,
  [out, retval] VARIANT* Info
);
```

Parameters

ActionId   [in] ID for the specified Action.

Info   [out, retval] An array of data values that define the Action.

Remarks

The Info array is filled with the following:

Element	Field	Type
Info[0]	Name	BSTR
Info[1]	Type	long
Info[2]	Options	unsigned long
Info[3]	Delay	double
Info[4]	DutyCycle	double
Info[5]	StopAfter	long
Info[6]	PortName	BSTR
Info[7]	Output	BSTR

Info[8]	Host	BSTR
Info[9]	Group	BSTR

GetActionName (ActionId)

Returns the name of an Action specified by its ID.

Syntax

```
HRESULT GetActionName(
  [in] long ActionId,
  [out, retval] BSTR* Name
);
```

Parameters

ActionId   [in] ID for specified Action.

Name   [out, retval] Name of the Action.

GetActionOption (ActionId, OptionType)

Returns the option for an Action, specified by its ID and the option type.

Syntax

```
HRESULT GetActionOption(
  [in] long ActionId,
  [in] long OptionType,
  [out, retval] long* OptionVal
);
```

Parameters

ActionId   [in] ID for specified Action.

OptionType   [in] Type of option to examine.

OptionVal   [out, retval] Value of the specified option.

Remarks

Three option types are supported for Actions:

Value	Description	Returned `OptionVal`
1	Expression	0: output is a simple string 1: output is an expression
2	Disable	0: Action is enabled 1: Action is disabled
3	Remote Expression	0: Expression is evaluated locally 1: Expression is evaluated on remote WACI

GetActionOutput (ActionId)

Returns the value that is sent out the Action's port upon the Action being triggered.

```
HRESULT GetActionOutput(
  [in] long ActionId,
  [out, retval] BTSR* Output
);
```

Parameters

ActionId	[in] ID of the Action to examine.
Output	[out, retval] Value to be output

Remarks

The value for `Output` depends on the type of Action.

Action Type	Values to set `Output` to
Variable	Depends on the type of Variable being assigned (see below)
Serial	A simple string or complex expression
Digital I/O	"High": Opens the port "Low": Closes the port
Relay	"On": Energizes the relay "Off": Turns off the relay
Event	"0": Cancel the Event "1": Execute the Event "2": Disable the Event "3": Enable the Event
IR	Group/Command to execute. The string value is a combination of the name of the IR group, a "/", and the name of a command within that group, e.g. "VCR/Play"
HTTP Post	A simple string or complex expression. This value is posted to the URL specified in `Port`.
E-mail	The simple string or complex expression that makes up the body of the e-mail message.
Log	A simple string or complex expression
Telnet	A simple string or complex expression

Valid values for `Output` for Variable Actions:

Variable Type	Valid values to assign
Number	String or expression that evaluates to a number
Range	String of the form "%d to %d", where %d represents a number
String	Anything that can be converted to a string

GetActionPort (ActionId)

Returns the destination port of the Action specified by `ActionId`

```
HRESULT GetActionPort(
  [in] long ActionId,
  [out, retval] BTSR* Port
);
```

Parameters

ActionId   [in] ID of the Action to examine.

Port   [out, retval] Port to output to

Remarks

The value of `Port` represents where the Action is going to send its output.

Action Type	Value for `Port`
Variable	The name of the Variable
Event	The name of the Event to cancel or execute
HTTP Post	The URL to post to, e.g. `http://waci/rpc`
E-mail	The email addresses to send the e-mail to. Separate each e-mail address with a semicolon (;), and separate the To, Cc, and Bcc address lists using a pipe (|), e.g. `joe@xyz.com;bob@abc.com|tom@ccaddr.com|support@foo.com`
Log	The name of the file to write the logged information to
Telnet	The IP address or host name of the Telnet server to command
All other cases	The name of the port of the associated device, e.g. "Port1".

GetActionStopAfter (ActionId)

Returns the number of times to repeat the Action

Syntax

```
HRESULT GetActionStopAfter(
  [in] long ActionId,
  [out, retval] long* StopAfter
);
```

Parameters

ActionId   [in] ID of the Action to examine.

StopAfter   [out, retval] The number of times to execute the Action.

Remarks

If `-1` is returned in `StopAfter`, then the Action will execute continuously until the owning Event is specifically canceled by an Action of type "Event".

GetActionType (ActionId)

Returns the type of an Action, specified by its ID.

```
HRESULT GetActionType(
  [in] long ActionId,
  [out, retval] long* Type
);
```

Parameters

ActionId	[in] ID for specified Action.
Type	[out, retval] Type of the specified Action.

Remarks

Action types that the WACI supports, depends on the device's hardware capabilities. `Type` values are:

Value	Action Type	Supported on
1	Set a Variable's value	WACI NX+ and WACI NX Jr.
2	Send serial data	WACI NX+ and WACI NX Jr.
3	Change a digital output (high/low)	WACI NX+ only
4	Change a relay voltage level	WACI NX+ only
5	Trigger another Event	WACI NX+ and WACI NX Jr.
6	Send an Infrared command	WACI NX+ only
7	HTTP post command	WACI NX+ and WACI NX Jr.
8	Email message	WACI NX+ and WACI NX Jr.
9	Log to file	WACI NX+ and WACI NX Jr.
10	Telnet command	WACI NX+ and WACI NX Jr.

MoveActionByIdx (EventId, Idx, Where)

Changes the order of execution for the Actions under the specified Event.

Syntax

```
HRESULT MoveActionByIdx(
  [in] long EventId,
  [in] long Idx,
  [in] long Where,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

EventId	[in] ID of specified Event.
Idx	[in] Index number for the Action of interest.
Where	[in] Direction to move the Action in the execution order. To move the Action up (execute earlier) pass 0. To move it down, pass 1.
Success	[out, retval] TRUE if the Events are successfully sorted, FALSE if sorting the Events failed.

Remarks

If `Where == 0`, and the Action is already at the top of the execution list (`Idx == 0`), then the function returns an error. Similarly, if `Where == 1` and the Action is at the bottom of the list, then an error is returned.

SetActionById (ActionId, Name, Type, Delay, DutyCycle, StopAfter, Port, Output)

Modifies an Action specified by its Action ID.

Syntax

```
HRESULT SetActionById(
  [in] long ActionId,
  [in] BSTR Name,
  [in] long Type,
  [in] double Delay,
  [in] double DutyCycle,
  [in] long StopAfter,
  [in] BSTR Port,
  [in] BSTR Output,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Action ID	[in] ID for the specified Action.
Name	[in] Name to assign to the specified Action.
Type	[in] Type of Action, e.g. Serial, Variable, etc.
Delay	[in] Number of seconds to delay before sending `Output` to `Port`. This is a real number, and non-integer values are allowed.
DutyCycle	[in] Number of seconds to delay between each successive execution of the Action. This is a real number, and non-integer values are allowed.
StopAfter	[in] Number of times to repeat the Action. Set to -1 to repeat infinitely.
Port	[in] Location where the output is sent.
Output	[in] Data that should be sent.
Success	[out, retval] TRUE if specified Action settings were updated, FALSE if updating the Action failed.

Remarks

Action types that the WACI supports, depend on the device's hardware capabilities (see GetActionType (ActionId), page 161 for valid type values).

The value of `Port` represents where the Action is going to send its output (see GetActionPort (ActionId), page 160 for details on valid port values).

The value for `Output` depends on the value of `Type` (see GetActionOutput (ActionId), page 159 for more information on valid values for `Output`).

163

SetActionByIdx (EventId, Idx, Name, Type, Delay, DutyCycle, StopAfter, Port, Output)

Modifies an Action specified by its associated Event ID and index.

Syntax

```
HRESULT SetActionByIdx(
  [in] long EventId,
  [in] long Idx,
  [in] BSTR Name,
  [in] long Type,
  [in] double Delay,
  [in] double DutyCycle,
  [in] long StopAfter,
  [in] BSTR Port,
  [in] BSTR Output,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Event ID	[in] ID of the Event that contains the Action.
Idx	[in] Index of the Action in the Event's Action list.
Name	[in] Name to use for the Action.
Type	[in] Type of Action, e.g. Serial, Variable, etc.
Delay	[in] Number of seconds to delay before sending `Output` to `Port`. This is a real number, and non-integer values are allowed.
DutyCycle	[in] Number of seconds to delay between each successive execution of the Action. This is a real number, and non-integer values are allowed.
StopAfter	[in] Number of times to repeat the Action. Set to `-1` to repeat infinitely.
Port	[in] Location where the output is sent
Output	[in] Data that should be output.
Success	[out, retval] TRUE if specified Action's settings were updated, FALSE if updating the Action failed.

Remarks

Action types that the WACI supports, depend on the device's hardware capabilities (see GetActionType (ActionId), page 161 for valid type values).

The value of `Port` represents where the Action is going to send its output (see GetActionPort (ActionId), page 160 for details on valid port values).

The value for `Output` depends on the value of `Type` (see GetActionOutput (ActionId), page 159 for more information on valid values for `Output`).

SetActionByName (EventId, Name, Type, Delay, DutyCycle, StopAfter, Port, Output)

Modifies an Action specified by its associated Event ID and Action name.

```
HRESULT SetActionByName(
  [in] long EventId,
  [in] BSTR Name,
  [in] long Type,
  [in] double Delay,
  [in] double DutyCycle,
  [in] long StopAfter,
  [in] BSTR Port,
  [in] BSTR Output,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Event ID	[in] ID of the Event that contains the Action.
Name	[in] Name of the desired Action.
Type	[in] Type of Action, e.g. Serial, Variable, etc.
Delay	[in] Number of seconds to delay before sending `Output` to `Port`. This is a real number, and non-integer values are allowed.
DutyCycle	[in] Number of seconds to delay between each successive execution of the Action. This is a real number, and non-integer values are allowed.
StopAfter	[in] Number of times to repeat the Action. Set to `-1` to repeat infinitely.
Port	[in] Location where the output is sent
Output	[in] Data that should be output.
Success	[out, retval] TRUE if specified Action settings were updated, FALSE if updating the Action failed.

Remarks

Action types that the WACI supports, depend on the device's hardware capabilities (see GetActionType (ActionId), page 161 for valid type values).

The value of `Port` represents where the Action is going to send its output (see GetActionPort (ActionId), page 160 for details on valid port values).

The value for `Output` depends on the value of `Type` (see GetActionOutput (ActionId), page 159 for more information on valid values for `Output`).

SetActionHost (ActionId)

Sets the name or IP address of the remote WACI.

Syntax

```
HRESULT SetActionHost(
  [in]long ActionId,
  [in]BSTR Host,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

ActionId	[in] ID for specified Action.

Host	[in] Host name or IP address of the remote WACI.
Success	[out, retval] TRUE if the specified Action option was updated, FALSE if updating the option failed.

Remarks

When an Action's host name is set to the name or IP address of a remote WACI, then the Action will be performed as if it were running locally on the remote WACI.

SetActionOption (ActionId, OptionType, OptionVal)

Modifies the option for an Action, specified by its ID and the option type.

Syntax

```
HRESULT SetActionOption(
  [in] long ActionId,
  [in] long OptionType,
  [in] long OptionVal
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

ActionId	[in] ID for specified Action.
OptionType	[in] Type of option to modify.
OptionVal	[in] Updated value for the option.
Success	[out, retval] TRUE if the specified Action option was updated, FALSE if updating the option failed.

Remarks

Two options are supported for Actions:

OptionType	Description	OptionVal
1	Expression	0: output is a simple string 1: output is an expression
2	Disable	0: Action is enabled 1: Action is disabled

SortActions (SortType, Direction)

Sorts all Actions by the sort type and direction.

Syntax

```
HRESULT SortActions(
  [in] long SortType,
  [in] long Direction,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

SortType	[in] Type of sort to perform

Direction   [in] Direction of the sort: 1 for ascending, and 0 for descending.

Success   [out, retval] TRUE if the Actions were successfully sorted, FALSE if sorting the Actions failed.

Remarks

This function is no longer supported.

Variable Methods

AddVariable (Name, Type, Default, Value, Persist)

Creates a new Variable

Syntax

```
HRESULT AddVariable(
  [in] BSTR Name,
  [in] long Type,
  [in] BSTR Default,
  [in] BSTR Value,
  [in] long Persist,
  [out, retval] long VarId
);
```

Parameters

Name   [in] Name to give the newly added Variable.

Type   [in] Type of Variable to add.
Valid types are:

`Type` Value	Description
1	Number
2	Schedule
3	String
4	Range

Default   [in] Default value of the new Variable.

Value   [in] Current value of the new Variable.

Persist   [in] Set to 1 to make the Variable persistent, 0 if not.

VarId   [out, retval] ID for the new Variable.

Remarks

Default and Value
The returned value is a string that represents the stored value of the Variable. The string value will need to be converted by the caller into the actual type, e.g. a Variable of type `long`.

The format for the returned string depends on the Variable's type. In the examples below, %d represents an integer, %s represents a string.

Type	Format	Example
Number	"%d"	"25"
String	"%s"	"This is a string"
Range	"%d to %d"	"1 to 35"
Schedule	"%d:%d:%d"	"17:35:02"

Persistence

If a Variable is persistent, the value of the Variable is stored to permanent storage each time it is changed. So, if the device is reset or the power is turned off, the Variable will still have its last value when the system restarts.

If the Variable is not persistent, then the Variable's value is set to its default whenever the system restarts.

AssignVariable (VarName, IsExpression, Value)

Sets a Variable to a particular value.

Syntax

```
HRESULT AssignVariable(
  [in]BSTR VarName,
  [in]long IsExpression,
  [in]BSTR Value,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

VarName   [in] Name of the Variable to assign `Value` to.

IsExpression   [in] Set to 0 if `Value` is a simple string, and 1 if `Value` is an expression.

Value   [in] The value to be given to the Variable. The format of this parameter should match the type of the Variable to be assigned (see GetVariableValue (VarId), page 173).

Success   [out, retval] TRUE if specified Variable's value was updated, FALSE if updating the Variable failed.

Remarks

`AssignVariable` does not change the type of the Variable when the assignment is made. If the Value parameter does not match the Variable's type, the result is undefined.

DeleteVariableById (VarId)

Delete a Variable specified by its ID.

Syntax

```
HRESULT DeleteVariableById(
  [in] long VarId,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

VarId   [in] ID of the Variable to delete.

Success   [out, retval] TRUE if the specified Variable was successfully deleted, FALSE if deleting the Variable failed.

Remarks

Get the ID of the Variable by calling either `GetVariableByName` or `GetVariableByIdx`.

DeleteVariableByName (Name)

Delete a Variable specified by its name.

```
HRESULT DeleteVariableByName(
  [in] BSTR Name,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Name	[in] Name of the Variable to delete.
Success	[out, retval] TRUE if the specified Variable was successfully deleted, FALSE if deleting the Variable failed.

GetVariableByIdx (Idx)

Returns the ID of the Variable located at the specified index.

Syntax

```
HRESULT GetVariableByIdx(
  [in] long Idx,
  [out, retval] long* VarId
);
```

Parameters

Idx	[in] Index of the Variable in the array of Variables.
VarID	[out, retval] ID of specified Variable.

Remarks

Use this function within a loop to get a list of all Variables on the system. The first Variable in the list has an index of 0. Call `GetVariableCount` to get the number of Variables in the list.

GetVariableByName (Name)

Returns a Variable's ID (not the value) from its name.

Syntax

```
HRESULT GetVariableByName(
  [in] BSTR Name,
  [out, retval] long* VarId
);
```

Parameters

Name	[in] Name of Variable to get the ID of.
VarId	[out, retval] ID of the named Variable.

GetVariableCount ( )

Returns the total number of Variables.

Syntax

```
HRESULT GetVariableCount(
  [out, retval] long* NumVariables
);
```

Parameters

NumVariables   [out, retval] Total number of Variables in the system.

Remarks

Use `GetVariableCount` in conjunction with `GetVariableByIdx` to enumerate the list of available Variables.

GetVariableDefault (VarId)

Returns the default value of the specified Variable.

Syntax

```
HRESULT GetVariableDefault(
  [in] long VarId,
  [out, retval] BSTR* Default
);
```

Parameters

VarId   [in] ID of specified Variable.

Default   [out, retval] String containing default value of the specified Variable.

Remarks

The returned value is a string that represents the Variable's default value. The string value will need to be converted by the caller into the actual type, e.g. a Variable of type `long`.

The format for the returned string depends on the Variable's type (see GetVariableType (VarId) page 172).

GetVariableGroup (VarId)

Returns the group name for the Variable.

Syntax

```
HRESULT GetVariableGroup(
  [in]long VarId,
  [out, retval]BSTR* Group
);
```

Parameters

VarId   [in] ID of the Variable to get the group name for.

Group   [out, retval] Group name assigned to the Variable.

Remarks

The group name is an arbitrary value, and is used only to help manage and sort Variables within the Event Manager web pages.

GetVariableName (VarId)

Returns the Variable's name from its ID.

```
HRESULT GetVariableByName(
  [in] long VarId,
  [out, retval] BSTR* Name
);
```

Parameters

VarId	[in] ID of the Variable to get the name of.
Name	[out, retval] Name of specified Variable.

Remarks

Variable names consist of letters, numbers, and the underscore. All other characters are not permitted. The Variable name should start with a non-number.

GetVariablePersist (VarId)

Returns the persistence status of a Variable.

Syntax

```
HRESULT GetVariablePersist(
  [in] long VarId,
  [out, retval] long* Persist
);
```

Parameters

VarId	[in] ID of specified Variable.
Persist	[out, retval] Returns 1 if the Variable is persistent, and 0 if it is not.

Remarks

When a Variable is persistent, the value of the Variable is stored to permanent storage after each time it is changed. So, if the device is reset or the power is turned off, the Variable will still have its last value when the system restarts.

If the Variable is not persistent, then the Variable's value is set to its default whenever the system restarts.

GetVariableType (VarId)

Returns the type of the specified Variable.

Syntax

```
HRESULT GetVariableType(
  [in] long VarId,
  [out, retval] long* Type
);
```

Parameters

VarId	[in] ID of specified Variable.
Type	[out, retval] The type of value stored in the Variable. Valid types are:

Type Value	Description
1	Number
2	Schedule
3	String
4	Range

GetVariableValue (VarId)

Returns the current value of the specified Variable.

Syntax

```
HRESULT GetVariableValue(
  [in] long VarId,
  [out, retval] BSTR* Value
);
```

Parameters

VarId   [in] ID of specified Variable.

Value   [out, retval] String containing the current value of the specified Variable.

Remarks

The returned value is a string that represents the stored value of the Variable. The string value will need to be converted by the caller into the actual type, e.g. a Variable of type long.

The format for the returned string depends on the Variable's type (see GetVariableType (VarId) page 172).

SetVariableById (VarId, Name, Type, Default, Value, Persist)

Modifies a Variable, specified by its ID.

Syntax

```
HRESULT SetVariableById(
  [in] long VarId,
  [in] BSTR Name,
  [in] long Type,
  [in] BSTR Default,
  [in] BSTR Value,
  [in] long Persist,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

VarId   [in] ID of the Variable.

Name   [in] Name of the specified Variable (can be modified).

Type   [in] Variable type.

Default   [in] Default value for the Variable.

Value   [in] Current value for the Variable.

Persist   [in] Set to 1 to make the Variable persistent. Set to 0 otherwise.

Success   [out, retval] TRUE if specified Variable settings were updated, FALSE if updating the Variable failed.

Remarks

Variable names consist of letters, numbers, and the underscore. All other characters are not permitted.

Default and Value
The returned value is a string that represents the stored value of the Variable. The string value will need to be converted by the caller into the actual type, e.g. a Variable of type long.

The format for the returned string depends on the Variable's type (see GetVariableType (VarId) page 172).

Persistence
When a Variable is persistent, the value of the Variable is stored to permanent storage each time it is changed. So, if the device is reset or the power is turned off, the Variable will still have its last value when the system restarts.

If the Variable is not persistent, then the Variable's value is set to its default whenever the system restarts.

SetVariableByName (Name, Type, Default, Value, Persist)

Modifies a Variable, specified by its name.

```
HRESULT SetVariableByName(
  [in] BSTR Name,
  [in] long Type,
  [in] BSTR Default,
  [in] BSTR Value,
  [in] long Persist,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

Name	[in] Name of the Variable to change.
Type	[in] Variable type.
Default	[in] Default value for the specified Variable.
Value	[in] Current value for the Variable.
Persist	[in] Set to 1 to make the Variable persistent, 0 to clear the persistent flag.
Success	[out, retval] TRUE if specified Variable settings were updated, FALSE if updating the Variable failed.

Remarks

Use SetVariableByID to modify the Variable's name.

Default and Value
The returned value is a string that represents the stored value of the Variable. The string value will need to be converted by the caller into the actual type, e.g. a Variable of type `long`.

The format for the returned string depends on the Variable's type (see GetVariableType (VarId) page 172).

 Persistence
When a Variable is persistent, the value of the Variable is stored to permanent storage each time it is changed. So, if the device is reset or the power is turned off, the Variable will still have its last value when the system restarts.

If the Variable is not persistent, then the Variable's value is set to its default whenever the system restarts.

SetVariableGroup (VarId, Group)

Assigns a group name to a Variable.

Syntax

```
HRESULT SetVariableGroup(
  [in]long VarId,
  [in]BSTR Group,
  [out, retval]VARIANT_BOOL* Success
);
```

Parameters

VarId	[in] ID of the Variable.
Group	[in] A group name to assign to the Variable.
Success	[out, retval] TRUE if the Variable was successfully updated, FALSE if the Variable failed to be changed.

Remarks

The group name is an arbitrary value, and is used only to help manage and sort Variables within the Event Manager web pages.

SortVariables (SortType, Direction)

Sorts all Variables by the sort type and direction.

Syntax

```
HRESULT SortVariables(
  [in] long SortType,
  [in] long Direction,
  [out, retval] VARIANT_BOOL* Success
);
```

Parameters

SortType   [in] Type of sort to perform.

Direction   [in] Direction of the sort: 1 for ascending, and 0 for descending.

Success   [out, retval] TRUE if the Variables are successfully sorted, FALSE if sorting the Variables failed.

Remarks

This function is no longer supported.

176

XV. Error Codes

The following table lists the error codes shown by the blue Status LED.

Number of Blinks	Error Description
1	RAM self-test failed
2	Real-time clock failed to stabilize
3	Non-volatile RAM failed self-test
4	No firmware loaded in Flash ROM
5	Failed to initialize Ethernet adapter
6	Failed during download of firmware image
7	Failure when writing to Flash ROM
9	Failed to relocate and initialize Kernel

XVI. Limited 3 Year Warranty

Aurora Multimedia Corp. ("Manufacturer") warrants that this product is free of defects in both materials and workmanship for the product lifetime as defined herein for parts and labor from date of purchase. This Limited Lifetime warranty covers products purchased in the year of 2008 and after. Motorized mechanical parts (Hard Drives, DVD, etc) are covered for a period of 1 year. Supplied batteries are not covered by this warranty. During the warranty period, and upon proof of purchase, the product will be repaired or replaced (with same or similar model) at our option without charge for parts or labor for the specified product lifetime warranty period.

This warranty shall not apply if any of the following:

A.   The product has been damaged by negligence, accident, lightning, water, act-of-God or mishandling; or,

B.   The product has not been operated in accordance with procedures specified in operating instructions: or,

C.   The product has been repaired and or altered by other than manufacturer or authorized service center; or,

D.   The product's original serial number has been modified or removed: or,

E.   External equipment other than supplied by manufacturer, in determination of manufacturer, shall have affected the performance, safety or reliability of the product.

F.   Part(s) are no longer available for product.

In the event that the product needs repair or replacement during the specified warranty period, product should be shipped back to Manufacturer at Purchaser's expense. Repaired or replaced product shall be returned to Purchaser by standard shipping methods at Manufacturer's discretion. Express shipping will be at the expense of the Purchaser. If Purchaser resides outside the contiguous US, return shipping shall be at Purchaser's expense.

No other warranty, express or implied other than Manufacturer's shall apply.

Manufacturer does not assume any responsibility for consequential damages, expenses or loss of revenue or property, inconvenience or interruption in operation experienced by the customer due to a malfunction of the purchased equipment. No warranty service performed on any product shall extend the applicable warranty period.

This warranty does not cover damage to the equipment during shipping and Manufacturer assumes no responsibility for such damage.

This product warranty extends to the original purchaser only and will be null and void upon any assignment or transfer.

XVII. FCC Part 15 Statement

RADIO AND TELEVISION INTERFERENCE

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

     - Reorient or relocate the receiving antenna.

     - Increase the separation between the equipment and the receiver.

     - Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.

     - Consult the dealer or an experienced radio/TV technician for help.

You may also find helpful the following booklet, prepared by the FCC: "How to Identify and Resolve Radio-TV Interference Problems." This booklet is available from the U.S. Government Printing Office, Washington D.C. 20402.

Changes and Modifications not expressly approved by the manufacturer or registrant of this equipment can void your authority to operate this equipment under Federal Communications Commissions rules.

In order to maintain compliance with FCC regulations shielded cables must be used with this equipment. Operation with non-approved equipment or unshielded cables is likely to result in interference to radio & television reception.

XVIII.   Index