C-Max™ ActiveX Server

Version 2.00 PRELIMINARY May 10, 2001

1. CMX.EXE should be launched once to properly register the ActiveX Object.

2. Create the ActiveX Object in the client application, the C-Max™ Banner will appear for about 2 seconds. Calls can then be made to the ActiveX Server.

Included files:

CMX.exe	ActiveX Server
CMX.tlb	Type Library
CMX ActiveX.doc	This document

OpenComms(port_number)

Initializes the communications between the ActiveX Object and the CPUXA. This method needs to be called once, before any other methods are called.

Usage:

OpenComms(*port_number*)

Where:

port_number

is a valid com port between 1 and 8

Note: If specified com port does not exist, or is already in use, an error box will appear.

CloseComms()

Closes the communications between the ActiveX Object and the CPUXA. This method should be called at least 5 seconds before closing the client application.

LearnIR(number, frequency)

Instructs the CPUXA to "learn" an IR command into it's memory. When this call is made, a red window will appear until the IR is detected on the CPUXA.

Usage:

LearnIR(*number, frequency*)

Where:

number

is the IR number to store the learned command in.
(Rev 2 boxes 0-79) (Rev 3 boxes 0-399)

frequency

is the modulation frequency in kHz for the learned command.
A frequency of 38 (kHz) works for most manufacturers of AV equipment.

SendIR(number, module, zone)

Transmits IR command out a specified emitter.

Usage:

SendIR(*number ,module, zone*)

Where:

number

is the IR number to store the learned command in. (0-79, 0-399)

module

is "0" for sending the command out the CPUXA's local port
For sending IR commands out of remote SECU16IR modules, module is the address of that SECU16IR. (1-127)

zone

is the zone number (0-15) on the remote SECU16IR module to send the command from. (*zone* does not apply if *module* is "0")

GetIR(*number, *module, *zone)

Retrieves the next IR code received and recognized (from previous learns) from the CPUXA.

Usage:

GetIR(**number,* * module,* * zone*)

Where:

number

is the IR number recognized (1-79)
(65535 if no IR is sensed)
(0 or 255 if IR was sensed but no match was found)

module

is where the IR was received. (0 = CPUXA, 1-127 = Remote CPUXAs)

zone

is always 0

Note: **number, *module* and *zone* are passed as integer pointers.

GetX10(*house, *key, *data)

Retrieves the next X10 code received from the CPUXA.

Usage:

GetX10(**house,* *key,* *data*)

Where:

house
is the House Code (0-15)
(65535 if no X10 is received)

key
is the X10 key code (0-31)

data
is always 0

Note: **house*, **key* and **data* are passed as integer pointers.

SendX10(*house, *key, *repeats)

Sends the X10 code specified from the CPUXA.

Usage:

SendX10(*house, key, repeats*)

Where:

house
is the House Code (0-15)

key
is the X10 key code (0-31)

repeats
is the number of times to repeat the transmission (1-15)
(used only on DIM and BRIGHT commands)

GetPoint(module, point, *status)

Returns whether an input (or relay) point is on or off.

Usage:

GetPoint(*module, point,* status*)

Where:

module
is the module address (1-127) where the point (input) is located

point
is the point (0-15) number to sense

status
is the status of the point (0 = off, 1 = on)

Note: **status* is passed as an integer pointer.

SendPoint(module, point, status)

Turn a relay on or off.

Usage:

SendPoint(*module, point, status*)

Where:

module

is the module address (1-127) where the point (input) is located

point

is the point (0-15) number to sense

status

is the status of the point (0 = off, 1 = on)

GetVariable(number, *value)

Returns a CPU variable value

Usage:

GetVariable(*number, *value*)

Where:

number

is the variable number (1-127) to retrieve

**value*

variable value

Note: *value is passed as an integer pointer

GetTimer(number, *value)

Returns a CPU timer value

Usage:

GetTimer(*number, *value*)

Where:

number

is the number number (1-64) to retrieve

**value*

timer value

Note: *value is passed as an integer pointer

WriteUnitParameter(module, parm, data)

Change a module parameter

Usage:

WriteUnitParameter(*module, parm, data*)

Where:

module

is the module number (1-127) to write the parameter

parm

the paramter to write

data

new parameter value

WriteCPUParameter(parm, data)

Change a CPU parameter

Usage:

WriteCPUParameter(*parm, data*)

Where:

parm

the paramter to write

data

new parameter value

SetCPUClock(hour, minute, month, date, year, day)

Change the CPU clock

Usage:

SetCPUClock (*hour, minute, month, date, year, day*)

Where:

hour	*(0 – 23)*	
minute	*(0 – 59)*	
month	*(1 – 12)*	
date	*(0 – 31)*	
year	*(00 – 99)*	
day	*(0 - 6)*	*Day of the week, Sunday = 0*

Note: This function does not check for valid values, the programmer is responsible for ensuring all values passed are valid

GetCPUClock(*hour, *minute, *month, *date, *year, *day)

Read the CPU clock

Usage:

SetCPUClock (*hour, minute, month, date, year, day*)

Where:

hour	*(0 – 23)*
minute	*(0 – 59)*
month	*(1 – 12)*
date	*(0 – 31)*
year	*(00 – 99)*
day	*(0 - 6) Day of the week, Sunday = 0*

Note: All values are passed as integer pointers