ArtBus Manifest, Feb. 8, 07
by Ed Bennett (sbennettSYMBOLsaic.edu)

ArtBus is a hardware bus and a communications protocol for relatively low speed, low data rate command and control of discrete sensors and actuators (not streaming sound or video). It is a *distributed interface,* meaning that different parts of an ArtBus system can be spread about a space such a corridor or gallery. It is conceived of as being useful for small embedded systems in kinetic, robotic, or installation art or design projects. ArtBus is not a piece of hardware, per se, and is in fact hardware impartial. ArtBus "bugs" can be made on any embedded platform or microcontroller. The system is open hardware, open software under the GPL ver.2.

In the Summer of 2006, a large interactive installation art project was being planned in which various sensors and control points were to be distributed along a narrow, but 100 foot long space and and connected to a central computer running Max/MSP. This required various analog sensors to be read, and various lines to be turned on an off at different times. Initially, an Ezio or Arduino style I/O board seemed to be the right interface. On closer examination, it was realized that using those types of breakout board systems, there would not be enough of the right kind of I/O available on the interface boards, and the wire lengths made sending fragile sensor voltages or serial control data very awkward. Additionally, one of the devices being considered for use in multiples in the installation was a hacked MP3 player (needing specific pulsed commands to cue and play) which would have required more control intelligence close-by than the long-wires with Arduino model could provide. This set of problems created the context for designing a very minimal protocol and hardware layer that could be used as the basis for connecting many kinds of specialized I/O devices to a host controller over a single wire pair.

Openness and simplicity are basic virtues of ArtBus. It is designed so near-novice programmers and fabricators can extend the system for their own uses and share their work with others.

Specialized devices which provide just the right amount of just the right type of I/O are at the heart of the ArtBus concept. The first device made to test the bus concept was a hacked MP3 player. It was a silly place to start because it mixed two sets of unknowns into one problem set, but since it works, it really helps make the point about specialization in device function. Heinliein said "Specialization is for insects." Since the purpose of the ArtBus is to enable connection of specialized but disparate I/O, sensing, data, and devices, it is appropriate to refer to the individual devices as "bugs". What makes an ArtBus bug what it is, is that the commands and bit-twiddling needed to operate a specific sensor or actuator are wrapped in an ArtBus command. A local MCU attached to the raw device or devices in question operates the sensor or actuator and returns requested data back to the bus master. This "wrapping" makes all ArtBus messaging look as alike as possible on all platforms and devices, and allows any ArtBus bug to coexist on the same communications wire and in the same host development environment no matter what its function. Devices prone to being made into bugs include sonar rangers, motor controllers, realtime clock-calendars, lamp dimmers, motion sensors, RFID readers, GPS receivers, Bluetooth radios, Zigbee modems, accelerometers, servo controllers, LED drivers, MP3 players, proximity sensors, and so on.

When the real world intervenes and host-based multimedia applications show quirks in their ability to communicate with hardware (as in the case of Max/MSP or Flash), a program called a mediator can be run on the multimedia host to make its style of hardware I/O appear transparent to the ArtBus.

ArtBus single-master half-duplex system, with the master being any serial-capable system. This includes MAX/MSP, Director, Flash, Processing, and even, or especially, any generic microcontroller which can speak serial (which is all of them, as far as I know). Bridges to Internet and Bluetooth through specialized adapters are an easy and obvious extension to the bus. For example, the bus master could be a bluetooth to serial adapter.

Using the ArtBus protocol.

Connect the serial data line. Send "!" then address, e.g."A" then command. An addresses is a single character in upper case.

Z is the default group address. Bug addresses are named starting at A and going to Z. Bug groups are named starting at Z and going to A. It is unlikely that so many bugs would be connected that A-Z would not provide enough addresses and groups.

Commands consist of a single lower case character. Each type of bug has its own unique commands. ArtBus commands a through f must return values. Commands i through v do not return values. Semicolons follow numbers.

Example using an Analog to Digital converter Bug:

AB_analog at address F, read input 3	!Fa3;
AB_analog at address B, reboot	!Bz
Stop all devices:	!Zh
All devices resume:	!Zg

AB_analog commands:

a = read_a_line
- send: a3;
- what happens: analog input 3 is read and a three digit ascii string followed by a newline is returned

g = go (asynchronous)
- send: g
- what happens: if paused, action resumes and queued commands are executed from the buffer

h = stop (asynchronous)
- send: h
- what happens: command execution is paused, but commands still enqueue into buffer

x = ping_back
- send : x
- what happens: address of the current node, and the error status byte

z = reset_cpu
- send: z
- what happens: reset cpu

Commands g, h, w, x, y, and z are special because they mean the same thing to every type of bug, regardless of its particular function.