SimpleMessageSystem

SimpleMessageSystem is a library for Arduino 0004 and up. It facilitates communication with terminals or message based programs like Pure Data or Max/Msp. All serial input and output is interpreted as ASCII messages.

Get it here: http://tof.danslchamp.org/

Contact: tof [at] danslchamp [dot] org

The main advantages of this approach are:

- Send, receive and parse lists of characters and integers to and from the Arduino Board.
- A terminal program, Max/Msp and Pure Data can communicate with the same Arduino Board code.
- Can be adpated to send, receive and parse strings.
- Human readable.

The main disadvantage:

- Slower than a specialized serial protocol that treats bits or bytes as numbers instead of ASCII characters.

WHAT IS A MESSAGE?

A message is a series of words, made from ASCII characters, separated by spaces and terminated by a carriage return (and an optional line feed). Messages are built using the following structure:

word1 (space) word2 (space) word3 (carriage return)

In the current version of SimpleMessageSystem, words can be either one character or an integer. Once a message is terminated by a carriage return it is considered bulit and can be parsed by the code.

Arduino code example 1:

```
// Arduino code

    if (messageBuild()) { // Checks to see if the message is complete
       firstChar = messageGetChar()) { // Gets the first word as a character
       if (firstChar = 'r') { // Checking for the character 'r'
          secondChar = messageGetChar() // Gets the next word as a character
          if (firstChar = 'd') // The next character has to be 'd' to continue
             messageSendChar('d');  // Echo what is being read
             for (char i=2;i<14;i++) {
                messageSendInt(digitalRead(i)); // Read pins 2 to 13
             }
             messageEnd(); // Terminate the message being sent
          }
       }
    }

// Arduino code end
```

If the preceding Arduino+SimpleMessageSystem code receives the message *r d CR* (CR stands for a carriage return)

it will return the value of all the digital pins in a message taking the following structure:

d pin2 pin3 pin4 pin5 pin6 pin7 pin8 pin9 pin10 pin11 pin12 pin13 CR

Arduino code example 2:

```
// Arduino code

    if (messageBuild()) { // Checks to see if the message is complete
          firstChar = messageGetChar()) { // Gets the first word as a character
          if (firstChar = 'w') { // Checking for the character 'w'
              int pin = messageGetInt(); // Gets the next word an integer
              int state = messageGetInt(); // Gets the next word an integer
              pinMode(pin,OUTPUT);
              digitalWrite(pin,state);
     }

// Arduino code end
```

If the preceding Arduino+SimpleMessageSystem code receives the message

w 13 1 CR (CR stands for a carriage return)

it will set pin 13 HIGH (digitalWrite(13,HIGH)). In Arduino HIGH = 1, LOW = 0.

HOW TO IMPORT/INSTALL

1) Put the SimpleMessageSystem folder in "{arduino-0004\lib\targets\libraries\".
2) Open the Arduino program.
3) Create a new sketch (or open one) and select from the menubar "Sketch->Import Library->SimpleMessageSystem". Once the library is imported, an "#inlcude SimpleMessageSystem.h" line will appear at the top of your Sketch.
4) When you run (compile) your code, it will also compile the SimpleMessageSystem library. You will get warning messages, but that is “normal”.

EXAMPLE APPLICATIONS

Transfer the following example to the Arduino Board:

"Arduino Menubar->File->Sketchbook->Examples-> Library-SimpleMessageSystem->SimpleMessageSystem_example1"

Open one of the following Max/Msp or Pure Data examples to communicate with the Arduino Board:

Max/Msp Example

Open "SimpleMessageSystem.mxb" in the "MaxMsp Example" folder. Use the abstractions "max2asciimessage" and "asciimessage2max" in your custom Max/Msp patches to communicate with an Arduino Board loaded with the SimpleMessageSystem library.

Pure Data Example

Open "SimpleMessageSystem.pd" in the "Pure Data Example" folder. Use the abstractions "pd2ascii" and "ascii2pd" in your custom Pure Data patches to communicate with an Arduino Board loaded with the SimpleMessageSystem library. if you are running Linux and do not know the Arduino port number to use with comport execute the following bash (terminal) command:

```
i=0; for f in `ls /dev/tty[US]*`; do echo $i $f; i=$(( $i + 1 )); done
```

HOW TO CUSTOMIZE

To receive a message:

1. Make a call to *messageBuild()*. If it returns a 1 or more, the incomming message has been terminated by a carriage return and is considered complete (*messageBuild()* actually returns the number of characters in the message, including spaces).

2. Get the first word, by using *messageGetChar()* or *messageGetInt()*.

3. Repeat step 2 for as many words as your are expecting.

To send a message:

1. Make a call to *messageSendChar()* or *messageSendInt()* to send a character or an integer.

2. Repeat step 2 for every element you want to send. Spaces are automatically inserted between every call.

3. Make a call to *messageEnd()* to send a carriage return to signal that the message is terminated and can be processed by the target.

FUNCTIONS

Receiving

int messageBuild()

If the message has been terminated, returns the size of the message including spaces. WARNING: if you make a second call to messageBuild() it will discard the previous message!

char messageGetChar()

If a word is available, returns it as a char. If no word is available it will return 0. WARNING: if you send something like "foo", it will return 'f' and discard "oo".

int messageGetInt()

If a word is available, returns it as an integer. If no word is available it will return 0.

Sending

void messageSendChar(char value)

Send a character (automatically inserts spaces in the message between each character sent).

void messageSendInt(int value)

Send an integer (automatically inserts spaces in the message between every integer sent).

void messageEnd()

Terminates the message signaling the target that the message can be processed.