Arduino Toolkit 0.2.0

a somewhat MATLAB compatable Arduino toolkit for GNU Octave.

John Donoghue

Distribution

The GNU Octave arduino package is *free* software. Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. This means that everyone is free to use it and free to redistribute it on certain conditions. The GNU Octave arduino package is not, however, in the public domain. It is copyrighted and there are restrictions on its distribution, but the restrictions are designed to ensure that others will have the same freedom to use and redistribute Octave that you have. The precise conditions can be found in the GNU General Public License that comes with the GNU Octave arduino package and that also appears in Appendix A [Copying], page 25.

To download a copy of the GNU Octave arduino package, please visit `http://octave.sourceforge.net/arduino/`.

Table of Contents

1 Installing and loading

The Arduino toolkit must be installed and then loaded to be used.

It can be installed in GNU Octave directly from octave-forge, or can be installed in an off-line mode via a downloaded tarball.

The toolkit must be then be loaded once per each GNU Octave session in order to use its functionalty.

1.1 Online Direct install

With an internet connection available, the Arduino package can be installed from octave-forge using the following command within GNU Octave:

```
pkg install -forge arduino
```

The latest released version of the toolkit will be downloaded and installed.

1.2 Off-line install

With the arduino toolkit package already downloaded, and in the current directory when running GNU Octave, the package can be installed using the following command within GNU Octave:

```
pkg install arduino-0.2.0.tar.gz
```

1.3 Loading

Regardless of the method of installing the Arduino toolkit, in order to use its functions, the toolkit must be loaded using the pkg load command:

```
pkg load arduino
```

The toolkit must be loaded on each GNU Octave session.

2 Hardware setup

In order to use the arduino hardware with the toolkit, it must be programmed with special firmware.

2.1 Programming the Arduino

To program the hardware, using a default configuration, run the arduinosetup command:

```
arduinosetup
```

A temporary Arduino project will be created, with the Arduino toolkit files copied to it and the Arduino IDE will open.

Set the board type and port correctly for the connected Arduino and press the upload button on the IDE.

The sources will be compiled and then uploaded to the connected arduino board.

After successful upload the Arduino IDE should be closed.

3 Connecting to an arduino

To control an arduino device, a connection must be made to it by creating an arduino object.

3.1 Connecting to a single arduino

Assuming a single arduino device is connected to the computer, creating an arduno object with no arguments will find the connected arduino and connect to it:

```
ar = arduino()
```

3.2 Connecting to a specific arduino

Where multiple arduinos may be connected to the computer, a specific board can be connected by specifying the name of the port it is connected to:

```
ar = arduino("/dev/ttyACM0")
```

The port name will be operating system dependant.

3.3 Querying available arduinos

To list the ports of all *programmed* available arduinos, the scanForArduinos function can be used:

```
scanForArduinos
```

It will provide a list of all available boards it can find with the port they are connected to.

4 Basic Input and Output Overview

Basic input and output can be performed on a connected arduino device using by calling the read and write functions for a specific named pin on the arduino.

A list of available pins can get found from the pins property of the connected arduio object and are also displayed as part of the default shown properties:

```
ar = arduino();
% get the pin names
pins = ar.availablepins
```

Pin generally follow a naming scheme of D<number> for digital pins and A<number> for analog pins.

Digital pins can be used to read and write digital data, but can not read analog voltages. Analog pins can perform digital I/O as well as reading voltages.

4.1 Performing Digital I/O

A pin's digital logic value can be true (1) or false (0) and can be set using the writeDigialPin function.

The following example attempts to set the D2 pin of the connected arduini object "ar" to true, waits 5 seconds and then sets it to false:

```
writeDigitalPin (ar,  "d2", true);
pause 5
writeDigitalPin (ar,  "d2", false);
```

Using the readDigitalPin will read the current logic state of the pin.

```
value = readDigitalPin (ar,  "d2");
```

4.2 Performing Analog Input

For analog pins, the voltage level can be read using a analog to digital conversion and will return a voltage level between 0 and the boards voltage (nominally 5V):

```
value = readVoltage (ar,  "a0");
```

The raw digital value of the pin can also be read instead of a volatge, giving a value between 0 and 2^x where x is the number of bits used by the analog to digtial converter.

```
value = readAnalogPin (ar,  "a0");
```

5 Protocol based I/O Overview

The arduino toolkit supports more complex I/O for SPI, I2C, Servo control and more.

5.1 SPI communication

SPI communication can be performed by creating a SPI dev object and then calling the writeRead function:

```
spi = spidev (ar,  "d2");
```

The function call expects a connected arduino object as the first argument, followed by the chip slelect pin of the SPI device.

After a device is created, a write to device followed by read can can be made using the writeRead function:

```
spi = spidev (ar,  "d2");
data = writeRead (spi,  100);
```

5.2 I2C communication

I2C communication can be performed by creating an I2C dev object for a specific I2C address.

The following example creates an i2c device that will communicate with a I2C device at address 100"

```
i2c = i2cdev (ar,  100);
```

After creating an I2C device, data can be read and writen using read, write, readRegister and writeRegister. The data to send and recieve will be device dependent.

5.3 Servo communication

Servo communication can be performed after creating a servo device object to operate on a PWM pin:

```
servoobj = servo(ar, "d9", "minpulseduration", 1.0e-3, ...
  "maxpulseduration", 2e-3);
```

The servo function expects the connected arduino object and the PWM pin that the servo is connnected to. Optional properties can be specified to control the setup of device.

In the example, the min and max pulse width values are set.

Using the servo object the current position and be read or set with values ranging between 0 to 1, with 0 being the minimum pulse width and 1 being the maximum.

The following example sets the servo to its middle position.

```
servoobj = servo(ar, "d9", "minpulseduration", 1.0e-3, ...
  "maxpulseduration", 2e-3);

writePosition (servoobj, 0.5);
```

5.4 Shift Registers

A shift register can be controlled by creating a shiftRegister object:

```
registerobj = shiftRegister(ar, '74hc164', "d2", "d3");
```

The parameters required are dependant on the type of shift register created.

Once a register object has been created, it can be read and written to using the read and write functions.

6 Examples

6.1 Blinking an LED

This example shows blinking the inbuilt LED on the Arduino board. Code is available by running:

```
edit examples/example_blink
```

Hardware setup

This example uses in the builtin LEDS, so requires only a connection of the Arduino board to computer for communication.

Create an Arduino object

```
ar = arduino ();
```

If you have more than one Arduino board connected, you may need to specify the port in order to connect to the correct device.

Query Device for pins connected to builtin LEDS

The pin connected to the Arduino UNO built in led if D13.

```
led_pin = "d13";
```

The connected pins can be queried programatically if desired.

```
pins = getLEDTerminals (ar);
```

Connected to a Arduino UNO would return a list pins containing only one item '13'.

The terminal number can be converted to a pin using getPinsFromTerminals:

```
led_pin = getPinsFromTerminals (ar, pins{1});
```

Turn the LED off

Write a 0 value to the pin to turn it off.

```
writeDigitalPin (ar, led_pin, 0);
```

Turn the LED on

Write a 1 value to the pin to turn it on

```
writeDigitalPin (ar, led_pin, 1);
```

Making the LED blink

Add a while loop with a pause between the changes in the pin state to blink.

```
while true
  writeDigitalPin (ar, led_pin, 0);
  pause (0.5)
  writeDigitalPin (ar, led_pin, 1);
  pause (0.5)
endwhile
```

6.2 Using I2C to communicate with an EEPROM

This example shows using I2C to communicate with a EEPROM chip. Code is available by running:

```
edit examples/example_i2c_eeprom
```

Hardware setup

Using an Arduino UNO, the board should be configured with the following connections between the board and a 24XX256 EEPROM chip:

A4	Connected to pin 5 of EEPROM
A5	Connected to pin 6 of EEPROM
5V	Connected to pin 8 of EEPROM
GND	Connected to pin 1,2,3,4 of EEPROM

Create an Arduino object

```
ar = arduino ();
```

If you have more than one Arduino board connected, you may need to specify the port in order to connect to the correct device.

Query I2C pins

Display the I2C terminals of the board:

```
getI2CTerminals(ar)
```

Scan the arduino for the connected device

```
scanI2Cbus(ar)
```

The devices listed should contain 0x50, the address of the EEPROM chip.

Create an I2C object to communicate to the EEPROM

```
eeprom = i2cdev(ar, 0x50)
```

Write data to the EEPROM

The EEPROM expects the first byte to be the page number, the second the offset, followed by data, so to write 1 2 3 4, starting address 0 (page 0, offset 0):

```
write(eeprom, [0 0 1 2 3 4])
```

Reading from the EEPROM

Reading from the EEPROM requires first writing the address to read from, in this case, if we want to read the 3, 4, this would be page 0, offset 2:

```
write(eeprom, [0 2])
```

Next read the 2 bytes:

```
data = read(eeprom, 2)
```

6.3 Using SPI to communicate with a mcp3002 10 bit ADC

This example shows using SPI to communicate with an mcp3002 10 bit ADC. Code is available by running:

```
edit examples/example_spi_mcp3002
```

Hardware setup

Using an Arduino UNO, the board should be configured with the following connections between the board and a mcp3002 chip:

D10 Connected to pin 1 (CS) of MCP3002

D11 Connected to pin 5 (DI) of MCP3002

D12 Connected to pin 6 (DO) of MCP3002

D13 Connected to pin 7 (CLK) MCP3002

VCC Connected to pin 8 (VDD) MCP3002

GND Connected to pin 4 (VSS) MCP3002

Analog input
Connected from pin 2 of the MCP3002 to a LOW (< 5V) voltage to measure

Create an Arduino object

```
ar = arduino ();
```

If you have more than one Arduino board connected, you may need to specify the port in order to connect to the correct device.

Create an SPI object to communicate to the MCP3002

```
adc = spidev(ar, "d10")
```

The d10 is the chip select pin connected from the Arduino to the MCP3002.

Read the ADC

The MCP3002 expects specific commands in order to read a channel.

For isslustration for the command to read chan 0 in single ended mode:

```
command (bits) in MSB mode to device:
[START SGL ODN MSBF X X X X] [ X X X X X X X X ]
   1    1   0    1   1 1 1 1    1 1 1 1 1 1 1 1
      [chan 0 ] MSB
data back:
   X    X  X     X   X 0 D D    D D D D D D D D
```

D is a output data bit

X is a dont care what value is input/output

The first byte contains the command and start of the data read back, the second bytes is written to clock out the rest of the ADC data.

In hex, this corresponds to 0xDF 0xFF,

```
data = writeRead(adc, [hex2dec("DF") hex2dec("FF")])
```

Of the data returned, the last 10 bits is the actual data, so convert data to a 16 bit value:

```
val = uint16(data(1))*256 + uint16(data(2))
```

Then bitand it to remove the non value parts, to get the ADC value:

```
val = bitand (val, hex2dec('3FF'))
```

To make the vaue correspond to a voltage it needs to be scaled as 0 will be 0Volts, 1023 will be 5Volts.

```
volts = double(val) * 5.0 / 1023.0;
```

7 Function Reference

The functions currently available in the Arduino toolkit are described below;

7.1 General Functions

7.1.1 arduinosetup

retval = arduinosetup ()
retval = arduinosetup (*propertyname*, *propertyvalue*)

Open the arduino config / programming tool to program the arduino hardware for usage with the Octave arduino functions.

A sequence of property name/value pairs can be given to the function to set defaults while programming.

Currently the following properties can be set:

libraries
: The value should be the name of a library, or string array of libraries to program on the arduino board.

arduinobinary
: The value should be the name/path of the arduino ide binary for programming. If not specified, the function will attempt to find the binary itself.

arduinosetup will create a temporary project using the arduino IDE and allow compiling aand programning of the code to an arduino.

See also: arduino, __arduino_binary__.

7.1.2 listArduinoLibraries

retval = listArduinoLibraries ()

Retrieve list of all known arduino library modules that are available.

retval is an cell array of string library names that are available for programming to the arduino.

See also: arduino, arduinosetup.

7.1.3 scanForArduinos

retval = scanForArduinos (*maxCount*)
retval = scanForArduinos (*maxCount*, *type*)

Scan system for programmed arduino boards.

scanForArduinos will scan the system for programmed arduino boards and return at most *maxCount* of them as a cell array in *retval*.

Each cell value of the cell array will contain a structure with values of:

port
: the serial port the arduino is connected to

board
: the board type of the arduino

if *maxCount* is not specified, or is a less than 1, the function will return as many arduino boards as it can detect.

If *type* is specified, the board type must match for the arduino to be added to the return list.

See also: arduino.

7.2 Arduino Functions

7.2.1 @arduino/arduino

retval = arduino ()
retval = arduino (*port*)
retval = arduino (*port*, *board*)
retval = arduino (*port*, *board*[, [*propname*, *propvalue*]*)

Create a arduino object with a connection to an arduino board.

port - full patth of serial port to connect to. For linux, usually /dev/ttySXXX, for windows COMXX.

board - name of board to connect (default is 'uno').

propname, *propvalue* - property name and value pair for additional properties to pass to the creation of the arduino object. Currently properties are ignored.

if the arduino function is called without parameters, it will scan for the first available arduino it can find and connect to it.

Function returns a arduino object is successfully connected.

See also: scanForArduinos.

7.2.2 @arduino/configurePin

mode = configurePin (*ar*, *pin*)
configurePin (*ar*, *pin*, *mode*)

Set/Get pin mode for a specified pin on arduino connection.

configurePin (*ar*, *pin*) will get the current mode of the specified pin.

configurePin (*ar*, *pin*, *mode*) will attempt set the pin to the specified mode if the mode is unset.

ar - the ardiuno object of the connection to an arduino board.

pin - string name of the pin to set/get the mode of.

mode - string mode to set the pin to or is currently set to.

Modes can be:

- AnalogInput - Acquire analog signals from pin
- DigitalInput - Acquire digital signals from pin
- DigitalOutput - Generate digital signals from pin
- I2C - Specify a pin to use with I2C protocol
- Pullup - Apecify pin to use a pullup switch
- PWM - Specify pin to use a pulse width modulator
- Servo - Specify pin to use a servo
- SPI - Specify a pin to use with SPI protocol
- Unset - Clears pin designation. The pin is no longer reserved and can be automatically set at the next operation.

See also: arduino.

7.2.3 @arduino/configurePinResource

mode = `configurePinResource` (*ar*, *pin*)
`configurePinResource` (*ar*, *pin*, *owner*, *mode*)
`configurePinResource` (*ar*, *pin*, *owner*, *mode*, *force*)

Set/Get pin mode for a specified pin on arduino connection.

configurePinResource (*ar*, *pin*) will get the current mode of the specified pin.

configurePinResource (*ar*, *pin*, *owner*, *mode*) will attempt set the pin to the specified mode and owner.

If the pin is already owned by another owner, the configure will fail unless the force option is used. If the mode is lready set, configure will fail unless force is used.

ar - the ardiuno object of the connection to an arduino board.

pin - string name of the pin to set/get the mode of.

mode - string mode tto set the pin to or is currently set to.

owner - string name to use as the pin owner.

force - boolean to force mode change. If not set, it will be false.

Modes can be:

- AnalogInput - Acquire analog signals from pin
- DigitalInput - Acquire digital signals from pin
- DigitalOutput - Generate digital signals from pin
- I2C - Specify a pin to use with I2C protocol
- Pullup - Specify pin to use a pullup switch
- PWM - Specify pin to use a pulse width modulator
- Servo - Specify pin to use a servo
- SPI - Specify a pin to use with SPI protocol
- Unset - Clears pin designation. The pin is no longer reserved and can be automatically set at the next operation.

See also: arduino, configurePin.

7.2.4 @arduino/decrementResourceCount

count = `decrementResourceCount` (*ar*, *resource*)

Decrement the count of a named resource by 1 and return the new count.

ar - connected arduino object

resource - name of resource to decrement count.

count = count of uses registered to resource.

See also: getResourceCount. incrementResourceCount.

7.2.5 @arduino/display

`display` (*ar*)

Display the arduino object in a verbose way, showing the board and available pins.

ar - the arduino object.

If the arduino object has debug mode set, additional information will be displayed.

See also: arduino.

7.2.6 @arduino/getI2CTerminals

pinlist = `getI2CTerminals (`*ar*`)`

Get a cell list of pin Ids available are used for I2C mode.

ar - the arduino object.

pinlist - cell list of pin numbers available for I2C use.

See also: arduino.

7.2.7 @arduino/getLEDTerminals

pinlist = `getLEDTerminals (`*ar*`)`

Get a cell list of pin Ids available are connected natively to LEDs.

ar - the arduino object.

pinlist - cell list of pin numbers available for LED use.

See also: arduino.

7.2.8 @arduino/getMCU

mcu = `getMCU (`*ar*`)`

Get the MCU used by the connected arduino.

ar - arduino object connected to a arduino board.

mcu - string representing the mcu used by the arduino board.

See also: arduino.

7.2.9 @arduino/getPWMTerminals

pinlist = `getPWMTerminals (`*ar*`)`

Get a cell list of pin Ids available for PWM use.

ar - the arduino object.

pinlist - cell list of pin numbers available for PWM use.

See also: arduino.

7.2.10 @arduino/getPinsFromTerminals

pinnames = `getPinsFromTerminals (`*ar*, *terminals*`)`

Get the pin names from the input terminal values.

ar - the connected arduino object.

terminals - the numeric pin number, or array of pin numbers to get pin names.

pinnames - the string names of each input pin. If terminals was a single value, the return will be a single string, otherwise it will return a cell array of each pin name.

See also: arduino, getTerminalsFromPins.

7.2.11 @arduino/getResourceCount

count = `getResourceCount (`*ar*, *resource*`)`

Get the count of uses of a given resource.

ar - connected arduino object

resource - name of resource to get count for.

count = count of uses registered to resource.

See also: incrementResourceCount. decrementResourceCount.

7.2.12 @arduino/getResourceOwner

owner = getResourceOwner (*ar*, *terminal*)
Get the owner of pin allocated previously by configurePinResource.

ar - connected arduino object

terminal - terminal number to get owner of.

owner = owner of the terminal pin, or "" if not owned.

See also: configurePinResource.

7.2.13 @arduino/getSPITerminals

pinlist = getSPITerminals (*ar*)
Get a cell list of pin Ids available for SPI mode.

ar - the arduino object.

pinlist - cell list of pin numbers available for SPI use.

See also: arduino.

7.2.14 @arduino/getServoTerminals

pinlist = getServoTerminals (*ar*)
Get a cell list of pin Ids available for servo use.

ar - the arduino object.

pinlist - cell list of pin numbers available for servo use.

See also: arduino, getPWMTerminals.

7.2.15 @arduino/getTerminalMode

mode = getTerminalMode (*ar*, *terminal*)
Get the mode of a pin allocated previously by configurePinResource.

ar - connected arduino object

terminal - terminal number to get owner of.

mode - mode of the terminal pin, or "not_set" if not owned.

See also: configurePinResource, getResourceOwner.

7.2.16 @arduino/getTerminalsFromPins

pinnums = getTerminalsFromPins (*ar*, *pins*)
Get the terminal number for each pin.

ar - connected arduino object

pins - single pin name orcell or vector array of pin names.

pinnums - pin number of each named pin. If the input was a single string, returns a number. if the input pins was a vector or cell araay, return a cell array of npin numbers corresponding to each input pin name.

See also: arduino, getPinsFromTerminals.

7.2.17 @arduino/incrementResourceCount

count = incrementResourceCount (*ar*, *resource*)

Increment the count value of a named resource by 1 and return the new count

ar - connected arduino object

resource - name of resource to increment count.

count = count of uses registered to resource.

See also: getResourceCount. decrementResourceCount.

7.2.18 @arduino/isTerminalAnalog

ret = isTerminalAnalog (*obj*, *terminal*)

Return true if pin is capable of analog input

ar - the connected arduino object

terminal is a terminal number ot check

ret return 1 of is a analog pin, 0 otherwise

7.2.19 @arduino/isTerminalDigital

ret = isTerminalDigital(*obj*, *terminal*)

Return true if pin is capable of digital functions

ar - the connected arduino object

terminal is a terminal number ot check

ret return 1 of is a digital pin, 0 otherwise

7.2.20 @arduino/playTone

playTone (*obj*, *pin*, *freq*, *duration*)

Play a tone of a given frequency on a specified pin.

Currently function does nothing.

7.2.21 @arduino/readAnalogPin

value = readAnalogPin (*ar*, *pin*)

Read analog voltage of *pin*.

ar - connected arduino object.

pin - string name of the pin to read.

value - analog value of the pin

Example usage:

```
ar = arduino ();
readAnalogPin(ar, "A4");
ans =
87
```

See also: arduino, readVoltage.

7.2.22 @arduino/readDigitalPin

value = readDigitalPin (*obj*, *pin*)

Read digital value from a digital I/O pin.

ar - connected arduino object.

pin - string name of the pin to read.

value - the logical value (0, 1, true false) of the current pin state.

Example usage

```
a = arduino ();
pinvalue = readDigitalPin (a, 'D5');
```

See also: arduino, writeDigitalPin.

7.2.23 @arduino/readVoltage

voltage = readVoltage (*ar*, *pin*)

Read analog voltage of a pin.

ar - connected arduino.

pin - pin name or number to query for voltage

voltage - scaled pin value as a voltage

Example usage:

```
ar = arduino ();
readVoltage(ar, "A4");
ans =
1.401
```

See also: arduino, readAnalogPin.

7.2.24 @arduino/reset

reset (*ar*)

Send reset command to arduino hardware to force a hardware reset.

ar - connected arduino object.

See also: arduino.

7.2.25 @arduino/sendCommand

outdata, outsize = sendCommand (*ar*, *libname*, *commandid*)
outdata, outsize = sendCommand (*ar*, *libname*, *commandid*, *data*)
outdata, outsize = sendCommand (*ar*, *libname*, *commandid*, *data*, *timeout*)

Send a command with option data to the connected arduino, waiting up to a specified number of seconds for a response.

ar - connected arduino object.

libname - library sending the command. The name should match a programmed library of the arduino, or an error will be displayed.

commandid - integer value for the command being sent to the arduino.

data - optional data sent with the command.

timeout - optional timeout to wait for data

outdata - data returned back from the arduino in response to command

outsize - size of data recieved out

If the arduino fails to respond with a valid reply, sendCommand will error.

See also: arduino.

7.2.26 @arduino/validatePin

`validatePin` (*`ar`*, *`pin`*, *`type`*)

Validate that the mode is allowed for specified pin

ar - connected arduino object

pin - name of pin to query mode validity of

mode - mode to query

Known modes are:

- 'I2C'
- 'SPI'
- 'PWM'
- 'Servo'
- 'analog'
- 'digital'

See also: arduino, configurePin.

7.2.27 @arduino/writeDigitalPin

`writeDigitalPin` (*`ar`*, *`pin`*, *`value`*)

Write digital value to a digital I/O pin.

ar - connected arduino object.

pin - string name of the pin to write to.

value - the logical value (0, 1, true false) to write to the pin.

If pin was unconfigured beore using, pin is set into digital mode.

Example usage:

```
a = arduino();
writeDigitalPin(a,'D5',1);
```

See also: arduino, readDigitalPin.

7.2.28 @arduino/writePWMDutyCycle

`writePWMDutyCyle` (*`ar`*, *`pin`*, *`value`*)

Set pin to output a square wave with a specified duty cycle.

ar - connected arduino object

pin - pin to write to.

value - duty cycle value where 0 = off, 0.5 = 50% on, 1 = always on.

Example usage:

```
a = arduino();
writePWMDutyCycle(a,'D5',0.5);
```

See also: arduino, writePWMVoltage.

7.2.29 @arduino/writePWMVoltage

`writePWMVoltage (ar, pin, voltage)`

Emulate an approximate voltage out of a pin using PWM.

ar - connected arduino object

pin - pin to write to.

voltage - voltage to emulate with PWM, between 0 - 5.0

Example usage:

```
a = arduino();
writePWMVoltage(a,'D5',1.0);
```

See also: arduino, writePWMDutyCycle.

7.3 Arduino I2C Functions

7.3.1 @i2cdev/display

`display (dev)`

Display i2cdev object.

See also: i2cdev.

7.3.2 @i2cdev/i2cdev

`dev = i2cdev (ar, address)`
`dev = i2cdev (ar, address, propname, propvalue)`

Create an i2cdev object to communicate to the i2c port on a connected arduino.

ar - connected arduino object

address - address to use for device on i2c bus.

propname, *propvalue* - property name/value pair for values to pass to devices.

Currently known properties:

bus — bus number (when arduino board supports multiple i2c buses) with value of 0 or 1.

See also: arduino.

7.3.3 @i2cdev/read

`dev = read (dev, numbytes)`
`dev = read (dev, numbytes, precision)`

Read a specified number of bytes from a i2cdev object using optional precision for bytesize.

dev - connected i2c device opened using i2cdev

numbytes - number of bytes to read.

precision - Optional precision for the output data read data. Currently known precision values are uint8 (default), int8, uint16, int16

See also: arduino, i2cdev.

7.3.4 @i2cdev/readRegister

`dev = readRegister (dev, reg, numbytes)`
`dev = readRegister (dev, reg, numbytes, precision)`

Read a specified numer of bytes from a register of an i2cdev object using optional precision for bytesize.

dev - connected i2c device opened using i2cdev

reg - registry value number

numbytes - number of bytes to read.

precision - Optional precision for the output data read data. Currently known precision values are uint8 (defualt), int8, uint16, int16

See also: arduino, i2cdev.

7.3.5 @i2cdev/subsref

`val =  subsref (dev, sub)`

subref for i2cdev

See also: i2cdev.

7.3.6 @i2cdev/write

`dev = write (dev, datain)`
`dev = write (dev, datain, precision)`

Write data to a i2cdev object using optional precision for the data byte used for the data.

dev - connected i2c device opened using i2cdev

datain - data to write to device. Datasize should not exceed the contraints of the data type specified for the precison.

precision - Optional precision for the input write data. Currently known precision values are uint8 (defualt), int8, uint16, int16

See also: arduino, i2cdev, read.

7.3.7 @i2cdev/writeRegister

`dev = writeRegister (dev, reg, datain)`
`dev = writeRegister (dev, dev, datain, precision)`

Write data to i2cdev object at a given registry position using optional precision for the data byte used for the data.

dev - connected i2c device opened using i2cdev

reg - registry position to write to.

datain - data to write to device. Datasize should not exceed the contraints of the data type specified for the precison.

precision - Optional precision for the input write data. Currently known precision values are uint8 (defualt), int8, uint16, int16

See also: arduino, i2cdev, read.

7.3.8 scanI2Cbus

`retval = scanI2Cbus (ar)`
`retval = scanI2Cbus (ar, bus)`
Scan arduino for devices on the I2C bus.

ar - arduino object connected to a arduino board.

bus - bus number to scan I2C devices, when multiple buses are available. If the bus is not specified, it will default to 0.

retval - cell array of addresses as strings in format of "0xXX".

Example usage:

```
# create arduino connection.
ar = arduino();
# get the pins that will be used for I2C
scanI2Cbus (ar)
% output
ans =
{
[1,1] = 0x50
}
```

See also: arduino, i2cdev.

7.4 Arduino Servo Functions

7.4.1 @servo/display

`display (dev)`
Display servo object.

See also: servo.

7.4.2 @servo/readPosition

`position = readPosition (servo)`
Read the position of a servo

servo - servo object created from arduino.servo.

position - value between 0 .. 1 for the current servo position, where 0 is the servo min position, 1 is the servo maximum position.

See also: servo, writePosition.

7.4.3 @servo/servo

obj = servo (*arduinoobj*, *pin*)
obj = servo (*arduinoobj*, *pin*, *propertyname*, *propertyvalue*)

Create a servo object using a specified oin on a arduino board.

obj - servo object

arduinoobj - connected arduino object

propertyname, *propertyvalue* - name value pairs for properties to pass to the created servo object.

Current properties are:

minpulseduration
: min PWM pulse value in seconds.

maxpulseduration
: max PWM pulse value in seconds.

Example:

```
# create arduino connection
ar = ardiuino();
# create hobby servo (1 - 2 ms puse range)
servo = servo(ar, "d9", "minpulseduration", 1.0e-3, "maxpulseduration", 2e-
3);
# center the servo
writePosition(servo, 0.5);
```

See also: arduino, readPosition, writePosition.

7.4.4 @servo/subsref

val = subsref (*dev*, *sub*)

subref for servo

See also: servo.

7.4.5 @servo/writePosition

writePosition (*servo*, *position*)

Write the position to a servo.

servo - servo object created from arduino.servo.

position - value between 0 .. 1 for the current servo position, where 0 is the servo min position, 1 is the servo maximum position.

See also: servo, readPosition.

7.5 Arduino Shiftregister Functions

7.5.1 @shiftRegister/display

retval = display (*register*)

Display the register object in a verbose way,

register - the arduino regsiter object created with shiftRegister.

See also: shiftRegister.

7.5.2 @shiftRegister/read

`retval = read (register)`
`retval = read (register, precision)`
read a value from the shift register.

register - shift register created from shiftRegister call.

precision - optional precision of the data, where precision can be a number in a multipe of 8 (ie: 8,16,32) or can be a named integer type: 8 of 'uint8', 'uint16', 'uint32'. The default precision is 8.

retval - returned data read from the register.

See also: shiftRegister, write.

7.5.3 @shiftRegister/reset

`reset (register)`
clear the shift register value.

register - shift register created from shiftRegister call.

See also: shiftRegister, read, write.

7.5.4 @shiftRegister/shiftRegister

`register = shiftRegister (ar, shifttype, dataPin, clockPin ...)`
`register = shiftRegister (ar,'74hc164', dataPin, clockPin, resetPin)`
`register = shiftRegister (ar,'74hc165', dataPin, clockPin, loadPin, clockEnablePin)`
`register = shiftRegister(ar,'74hc595', dataPin, clockPin, latchPin , resetPin)`
Create shift register of a given type, controlled by the input pins.

Common function parameter definition:

ar - connected arduino object.

shifttype - string name of the shift register type.

dataPin - pin used for data in/out of the device.

clockPin - pin used for clocking data on the shiftRegister.

register - register object

Other variables are dependant on the shift register type:

'74hc164' Additional inputs:

resetPin - optional pin for resetting the shift register.

'74hc165' Additional inputs:

loadPin - load pin to the shift register. *clockEnablePin* - clock enable pin.

'74hc595' Additional inputs:

latchPin - latching data to the shift register. *resetPin* - optional pin for resetting the shift register.

See also: arduino.

7.5.5 @shiftRegister/write

`write (register, dataIn)`
`write (register, dataIn, precision)`
Write a value to the shift register.

register - shift register created from shiftRegister call.

dataIn - data to clock into the shiftRegister.

precision - optional precision of the data, where precision can be a number in a multiple of 8 (ie: 8,16,32) or can be a named integer type of 'uint8', 'uint16', 'uint32'. The default precision is 8.

See also: shiftRegister, read.

7.6 Arduino SPI Functions

7.6.1 @spidev/display

`display (dev)`
Display spidev object.

See also: spidev.

7.6.2 @spidev/spidev

`dev = spidev (ar, cspin)`
`dev = spidev (ar, cspin, propname, propvalue)`
Create an spidev object to communicate to the spi port on a connected arduino.

ar - connected arduino object

cspin - chip select pin of selected spi port.

propname, *propvalue* - property name/value pair for values to pass to devices.

Currently known properties:

bitrate	bit rate speed in Mbs
bitorder	'msbfirst' or 'lsbfirst'
mode	SPI mode 0 - 3.

See also: arduino, readWrite.

7.6.3 @spidev/subsref

`val =  subsref (dev, sub)`
subref for spidev

See also: i2cdev.

7.6.4 @spidev/writeRead

`dataOut = readWrite (spi, dataIn)`
Write uint8 data to spi device and return back clocked out response data of same size.

spi - connected spi device on arduino

dataIn - uint8 sized data to send to spi device framed between SS frame.

dataOut - uint8 data clocked out during send to dataIn.

See also: arduino, spidev.

7.7 Arduino Addons

7.7.1 addon

retval = addon (*ar*, *addonname*)
retval = addon (*ar*, *addonname*, *varargs*)
Create an addon object using the addon named class.

retval is an cell array of string library names.

addonname is the name of the addon to create. The addon name can be a user addon or an inbuilt addon, however must appear in the listArduinoLibraries output and have been programmed onto the arduino.

varargs are optional values that will be provided verbatim to the the addon class constructor.

See also: arduino, arduinosetup, listArduinoLibraries.

7.7.2 arduinoioaddons.ExampleAddon.Echo

lb = arduinoioaddons.ExampleAddon.Echo ()
Basic Example matlab/octave code to ilustrate creating a user addon.

See also: addon.

7.8 Arduino I/O package

7.8.1 arduinoio.FilePath

retval = arduinoio.FilePath (*fullpathname*)
Get the directory component of a pathname.

fullpathname filepath to get directory component of.

retval the directory part of the filename.

7.8.2 arduinoio.LibFiles

filelist = arduinoio.LibFiles ()
Get the list of files used for the buildin arduino library

7.8.3 arduinoio.LibraryBase

lb = arduinoio.LibraryBase ()
Base class used for arduino library plugins

The return value *lb* is an object of the arduinio.LibraryBase class.

See also: arduino, listArduinoLibraries, addon.

7.8.4 arduinoio.getBoardConfig

retval = arduinoio.GetBoardConfig (*boardname*)
Return the configuration for a known arduino board type

Function is used to get the expected pin/board configuration for a named board type which is used to verify and identify the functionaly of the board.

boardname - name of board to get configuration of ie: "uno"

retval configuration struct.

Appendix A GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable

section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything

designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRIT-

ING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Index

U

V

W