Tutorial Arduino parte 3

En este tutorial veremos como conectar la placa Arduino a la red de 220V de nuestra casa , y poder utilizarlo para controlar nuestros aparatos eléctricos .

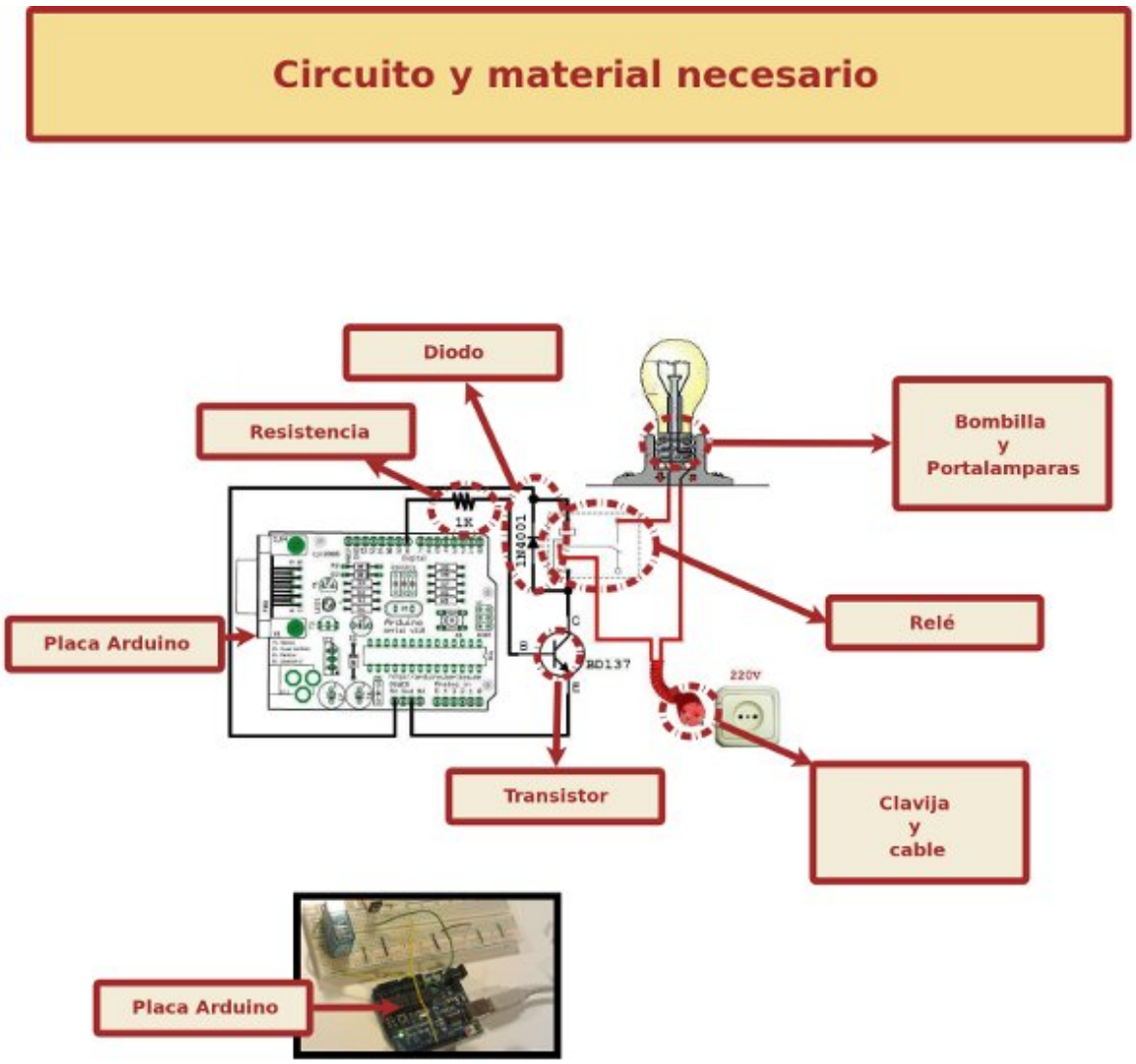
En este ejemplo mostraremos como encender una bombilla a 220V desde un lugar remoto .

Este tutorial está basado en un tutorial de la página de Arduino , ver referencias al final del post.

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro smartphone Android.

Material necesario:

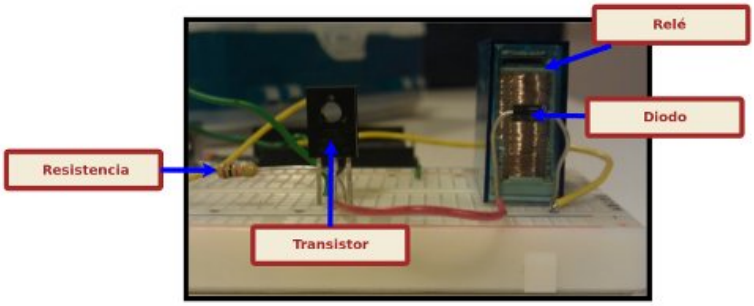
-Placa Arduino
-1 Relé de 5V ó 6V a 220V
-1 Diodo
-1 Transistor 5V
-1 Resistencia
-Cable
-1 Clavija
-1 Portalamparas
-1 Bombilla

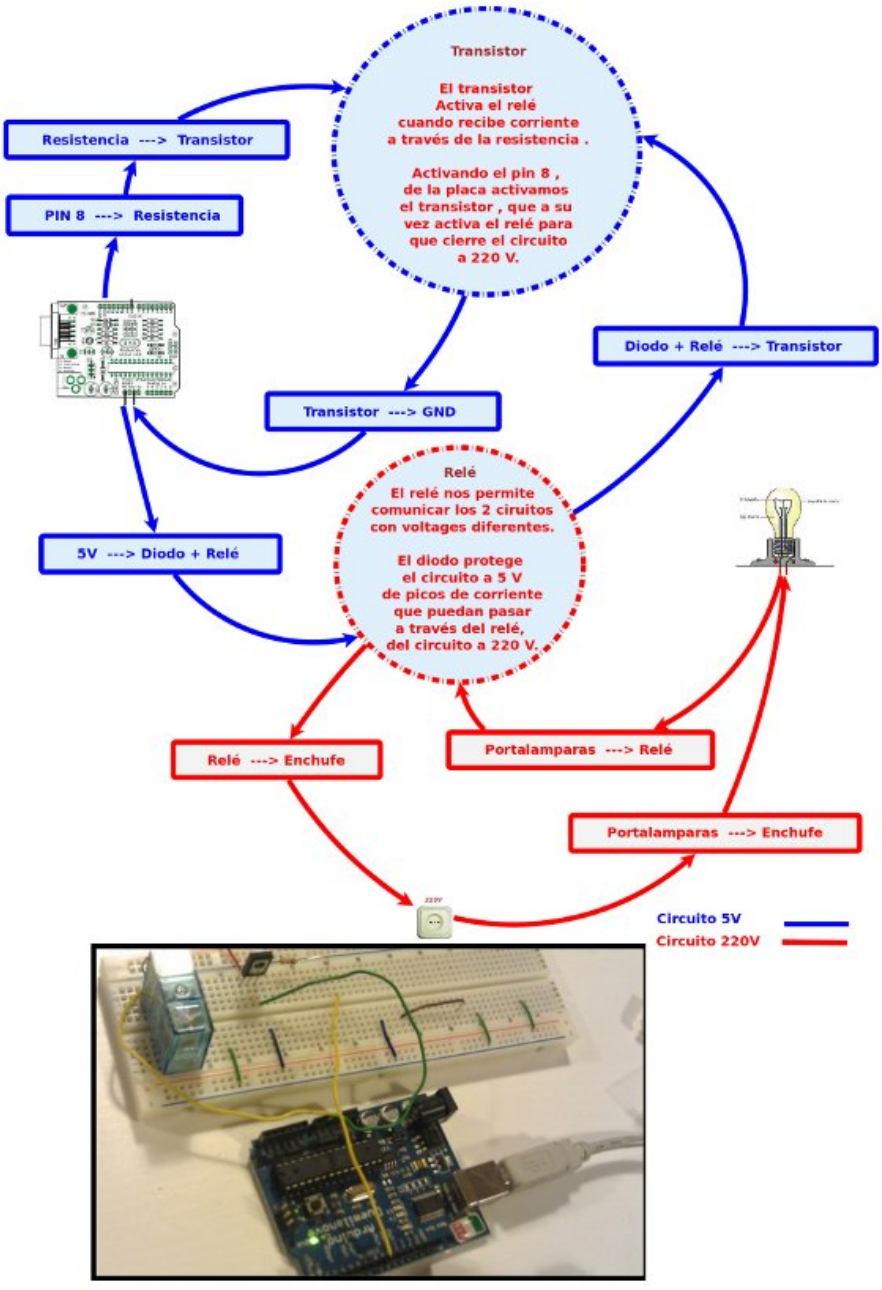


Material necesario

Una vez tengamos el material necesario procedemos a realizar el cableado de los circuitos .

Descripción del cableado de los circuitos





descripción circuitos

Una vez preparado el cableado , el código necesario para la placa Arduino es muy sencillo.
Utilizamos el pin 8 para activar el relé por medio del transistor.

Abrimos el puerto USB 115200 en modo lectura para recibir las notificaciones que nos enviará el servidor PHP.

Si por el puerto nos entra el caracter 2 , activa el relé , si es 1 lo desactiva.

Código Arduino:

```
int ledPin = 8;

int number_in = 0;

void setup() {

pinMode(ledPin, OUTPUT);

Serial.begin(115200);

}

void loop() {

if (Serial.available() > 0) {

number_in = Serial.read();

}

if (number_in > 0) {

if(number_in==2)
{
digitalWrite(ledPin, HIGH);
}
else
{
```

```
if(number_in==1)
{
digitalWrite(ledPin, LOW);
}
}

}

number_in = 0;

}
```

Utilizaremos un simple servidor php para enviar la señal a la placa Arduino conectada a nuestro servidor GNU/linux , en este caso a través del cable USB.

En nuestro caso la ruta al dispositivo que vamos a utilizar es /dev/ttyUSB0.

El puerto de la conexión USB que utilizaremos para comunicarnos con la placa Arduino será el 115200 .

Para encender o apagar la bombilla enviaremos por get a la ruta de nuestro servidor la variable hight , para encender , y la variable low para apagar.

Finalmente el servidor responde con un mensaje si ha recibido una señal correcta.

Código PHP:

```
<?php // Nonzero number to be sent to Arduino $c = 0; if(isset($_GET["hight"])) { $c=2; } if(isset($_GET["low"])) { $c=1; } if($c>0)
{
// Include the PHP serial class
require_once("phpSerial.php");

// Start a new serial class

$serial = new phpSerial;
```

```
// Specify the device being used

$serial->deviceSet("/dev/ttyUSB0");

// Set baud rate

$serial->confBaudRate(115200);

$serial->confParity("none");

$serial->confCharacterLength(8);

$serial->confStopBits(1);

$serial->confFlowControl("none");

// Open the device

$serial->deviceOpen();

// Write to the device

$serial->sendMessage(chr($c));

// Close the port

$serial->deviceClose();
$message= "Signal ".$c." Received! .
";

die(json_encode(array('status' => 'success', 'data' =>
$message)));
}
else
{
$message= "No Signal Received! .";
die(json_encode(array('status' => 'failed', 'data' =>
$message)));
}
?>
```

Y ya lo tenemos , tenemos una página php que nos funciona como

un servicio para poder encender o apagar una bombilla .

Podríamos crear en este punto una pequeña interfaz html , aprovechando el mismo fichero .php , comentando las líneas die(… , creando unos botones para encender y apagar la bombilla , pero nuestra intención es utilizarlo como servicio para una aplicación Android que os explicaremos en el próximo tutorial.

En este ejemplo hemos utilizado la librería phpSerial.php ,no recuerdo de donde la saqué así que os dejo el código , aunque en principio cualquier librería que os permita comunicar con el puerto serie debería valer.

este es el código:

```
<?php define ("SERIAL_DEVICE_NOTSET", 0); define ("SERIAL_DEVICE_SET", 1); define ("SERIAL_DEVICE_OPENED", 2); /** * Serial port control class * * THIS PROGRAM COMES WITH ABSOLUTELY NO WARANTIES ! * USE IT AT YOUR OWN RISKS ! * * @author Rémy Sanchez  * @thanks Aurélien Derouineau for finding how to open serial ports with windows
* @thanks Alec Avedisyan for help and testing with reading
* @copyright under GPL 2 licence
*/
class phpSerial
{
var $_device = null;
var $_windevice = null;
var $_dHandle = null;
var $_dState = SERIAL_DEVICE_NOTSET;
var $_buffer = "";
var $_os = "";

/**
* This var says if buffer should be flushed by sendMessage (true) or manualy (false)
*
```

```
* @var bool
*/
var $autoflush = true;

/**
* Constructor. Perform some checks about the OS and setserial
*
* @return phpSerial
*/
function phpSerial ()
{
setlocale(LC_ALL, "en_US");

$sysname = php_uname();

if (substr($sysname, 0, 5) === "Linux")
{
$this->_os = "linux";

if($this->_exec("stty --version") === 0)
{
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("No  stty  availible,  unable  to  run.",
E_USER_ERROR);
}
}
elseif(substr($sysname, 0, 7) === "Windows")
{
$this->_os = "windows";
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("Host OS is neither linux nor windows, unable tu
run.", E_USER_ERROR);
```

```
exit();
}
}

//
// OPEN/CLOSE DEVICE SECTION -- {START}
//

/**
* Device set function : used to set the device name/address.
* -> linux : use the device address, like /dev/ttyS0
* -> windows : use the COMxx device name, like COM1 (can also
be used
* with linux)
*
* @param string $device the name of the device to be used
* @return bool
*/
function deviceSet ($device)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
if ($this->_os === "linux")
{
if (preg_match("@^COM(\d+):?$@i", $device, $matches))
{
$device = "/dev/ttyS" . ($matches[1] - 1);
}

if ($this->_exec("stty -F " . $device) === 0)
{
$this->_device = $device;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}
elseif ($this->_os === "windows")
{
```

```
if (preg_match("@^COM(\d+):?$@i", $device, $matches) and
$this->_exec(exec("mode " . $device)) === 0)
{
$this->_windevice = "COM" . $matches[1];
$this->_device = "\\.\com" . $matches[1];
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}

trigger_error("Specified serial port is not valid",
E_USER_WARNING);
return false;
}
else
{
trigger_error("You must close your device before to set an
other one", E_USER_WARNING);
return false;
}
}

/**
* Opens the device for reading and/or writing.
*
* @param string $mode Opening mode : same parameter as fopen()
* @return bool
*/
function deviceOpen ($mode = "r+b")
{
if ($this->_dState === SERIAL_DEVICE_OPENED)
{
trigger_error("The device is already opened", E_USER_NOTICE);
return true;
}

if ($this->_dState === SERIAL_DEVICE_NOTSET)
{
```

```
trigger_error("The device must be set before to be open",
E_USER_WARNING);
return false;
}

if (!preg_match("@^[raw]\+?b?$@", $mode))
{
trigger_error("Invalid opening mode : ".$mode.". Use fopen()
modes.", E_USER_WARNING);
return false;
}

$this->_dHandle = @fopen($this->_device, $mode);

if ($this->_dHandle !== false)
{
stream_set_blocking($this->_dHandle, 0);
$this->_dState = SERIAL_DEVICE_OPENED;
return true;
}

$this->_dHandle = null;
trigger_error("Unable to open the device", E_USER_WARNING);
return false;
}

/**
* Closes the device
*
* @return bool
*/
function deviceClose ()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
return true;
}

if (fclose($this->_dHandle))
```

```
{
$this->_dHandle = null;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}

trigger_error("Unable to close the device", E_USER_ERROR);
return false;
}

//
// OPEN/CLOSE DEVICE SECTION -- {STOP}
//

//
// CONFIGURE SECTION -- {START}
//

/**
* Configure the Baud Rate
* Possible rates : 110, 150, 300, 600, 1200, 2400, 4800, 9600,
38400,
* 57600 and 115200.
*
* @param int $rate the rate to set the port in
* @return bool
*/
function confBaudRate ($rate)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the baud rate : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$validBauds = array (
110 => 11,
```

```
150 => 15,
300 => 30,
600 => 60,
1200 => 12,
2400 => 24,
4800 => 48,
9600 => 96,
19200 => 19,
38400 => 38400,
57600 => 57600,
115200 => 115200
);

if (isset($validBauds[$rate]))
{
if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " . (int)
$rate, $out);
}
elseif ($this->_os === "windows")
{
$ret = $this->_exec("mode " . $this->_windevice . " BAUD=" .
$validBauds[$rate], $out);
}
else return false;

if ($ret !== 0)
{
trigger_error ("Unable to set baud rate: " . $out[1],
E_USER_WARNING);
return false;
}
}
}

/**
* Configure parity.
```

```
* Modes : odd, even, none
*
* @param string $parity one of the modes
* @return bool
*/
function confParity ($parity)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set parity : the device is either not
set or opened", E_USER_WARNING);
return false;
}

$args = array(
"none" => "-parenb",
"odd" => "parenb parodd",
"even" => "parenb -parodd",
);

if (!isset($args[$parity]))
{
trigger_error("Parity mode not supported", E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
$args[$parity], $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " PARITY=" .
$parity{0}, $out);
}

if ($ret === 0)
```

```
{
return true;
}

trigger_error("Unable to set parity : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of a character.
*
* @param int $int length of a character (5 <= length <= 8) *
@return bool */ function confCharacterLength ($int) { if
($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set length of a character : the
device is either not set or opened", E_USER_WARNING);
return false;
}

$int = (int) $int;
if ($int < 5) $int = 5; elseif ($int > 8) $int = 8;

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " cs" .
$int, $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " DATA=" .
$int, $out);
}

if ($ret === 0)
{
return true;
```

```
}

trigger_error("Unable to set character length : " .$out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of stop bits.
*
* @param float $length the length of a stop bit. It must be
either 1,
* 1.5 or 2. 1.5 is not supported under linux and on some
computers.
* @return bool
*/
function confStopBits ($length)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the length of a stop bit : the
device is either not set or opened", E_USER_WARNING);
return false;
}

if ($length != 1 and $length != 2 and $length != 1.5 and
!($length == 1.5 and $this->_os === "linux"))
{
trigger_error("Specified stop bit length is invalid",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
(($length == 1) ? "-" : "") . "cstopb", $out);
}
```

```
else
{
$ret = $this->_exec("mode " . $this->_windevice . " STOP=" .
$length, $out);
}

if ($ret === 0)
{
return true;
}

trigger_error("Unable to set stop bit length : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Configures the flow control
*
* @param string $mode Set the flow control mode. Availible
modes :
* -> "none" : no flow control
* -> "rts/cts" : use RTS/CTS handshaking
* -> "xon/xoff" : use XON/XOFF protocol
* @return bool
*/
function confFlowControl ($mode)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set flow control mode : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$linuxModes = array(
"none" => "clocal -crtscts -ixon -ixoff",
"rts/cts" => "-clocal crtscts -ixon -ixoff",
```

```
"xon/xoff" => "-clocal -crtscts ixon ixoff"
);
$windowsModes = array(
"none" => "xon=off octs=off rts=on",
"rts/cts" => "xon=off octs=on rts=hs",
"xon/xoff" => "xon=on octs=off rts=on",
);

if ($mode !== "none" and $mode !== "rts/cts" and $mode !==
"xon/xoff") {
trigger_error("Invalid  flow  control  mode  specified",
E_USER_ERROR);
return false;
}

if ($this->_os === "linux")
$ret = $this->_exec("stty -F " . $this->_device . " " .
$linuxModes[$mode], $out);
else
$ret = $this->_exec("mode " . $this->_windevice . " " .
$windowsModes[$mode], $out);

if ($ret === 0) return true;
else {
trigger_error("Unable to set flow control : " . $out[1],
E_USER_ERROR);
return false;
}
}

/**
* Sets a setserial parameter (cf man setserial)
* NO MORE USEFUL !
* -> No longer supported
* -> Only use it if you need it
*
* @param string $param parameter name
* @param string $arg parameter value
```

```
* @return bool
*/
function setSetserialFlag ($param, $arg = "")
{
if (!$this->_ckOpened()) return false;

$return = exec ("setserial " . $this->_device . " " . $param .
" " . $arg . " 2>&1");

if ($return{0} === "I")
{
trigger_error("setserial: Invalid flag", E_USER_WARNING);
return false;
}
elseif ($return{0} === "/")
{
trigger_error("setserial:  Error  with  device  file",
E_USER_WARNING);
return false;
}
else
{
return true;
}
}

//
// CONFIGURE SECTION -- {STOP}
//

//
// I/O SECTION -- {START}
//

/**
* Sends a string to the device
*
* @param string $str string to be sent to the device
```

```
* @param float $waitForReply time to wait for the reply (in
seconds)
*/
function sendMessage ($str, $waitForReply = 0.1)
{
$this->_buffer .= $str;

if ($this->autoflush === true) $this->flush();

usleep((int) ($waitForReply * 1000000));
}

/**
* Reads the port until no new datas are availible, then return
the content.
*
* @pararm int $count number of characters to be read (will
stop before
* if less characters are in the buffer)
* @return string
*/
function readPort ($count = 0)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened to read it",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$content = ""; $i = 0;

if ($count !== 0)
{
do {
if ($i > $count) $content .= fread($this->_dHandle, ($count -
```

```
$i));
else $content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}
else
{
do {
$content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}

return $content;
}
elseif ($this->_os === "windows")
{
/* Do nohting : not implented yet */
}

trigger_error("Reading serial port is not implemented for
Windows", E_USER_WARNING);
return false;
}

/**
* Flushes the output buffer
*
* @return bool
*/
function flush ()
{
if (!$this->_ckOpened()) return false;

if (fwrite($this->_dHandle, $this->_buffer) !== false)
{
$this->_buffer = "";
return true;
}
else
```

```
{
$this->_buffer = "";
trigger_error("Error while sending message", E_USER_WARNING);
return false;
}
}

//
// I/O SECTION -- {STOP}
//

//
// INTERNAL TOOLKIT -- {START}
//

function _ckOpened()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened", E_USER_WARNING);
return false;
}

return true;
}

function _ckClosed()
{
if ($this->_dState !== SERIAL_DEVICE_CLOSED)
{
trigger_error("Device must be closed", E_USER_WARNING);
return false;
}

return true;
}

function _exec($cmd, &$out = null)
{
```

```
$desc = array(
1 => array("pipe", "w"),
2 => array("pipe", "w")
);

$proc = proc_open($cmd, $desc, $pipes);

$ret = stream_get_contents($pipes[1]);
$err = stream_get_contents($pipes[2]);

fclose($pipes[1]);
fclose($pipes[2]);

$retVal = proc_close($proc);

if (func_num_args() == 2) $out = array($ret, $err);
return $retVal;
}

//
// INTERNAL TOOLKIT -- {STOP}
//
}
?>
```

Referencias:

[Tutorial de la página de Arduino]

[Vídeo del turorial]

Próximamente:

Tutorial Arduino parte 4

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro celular Android.