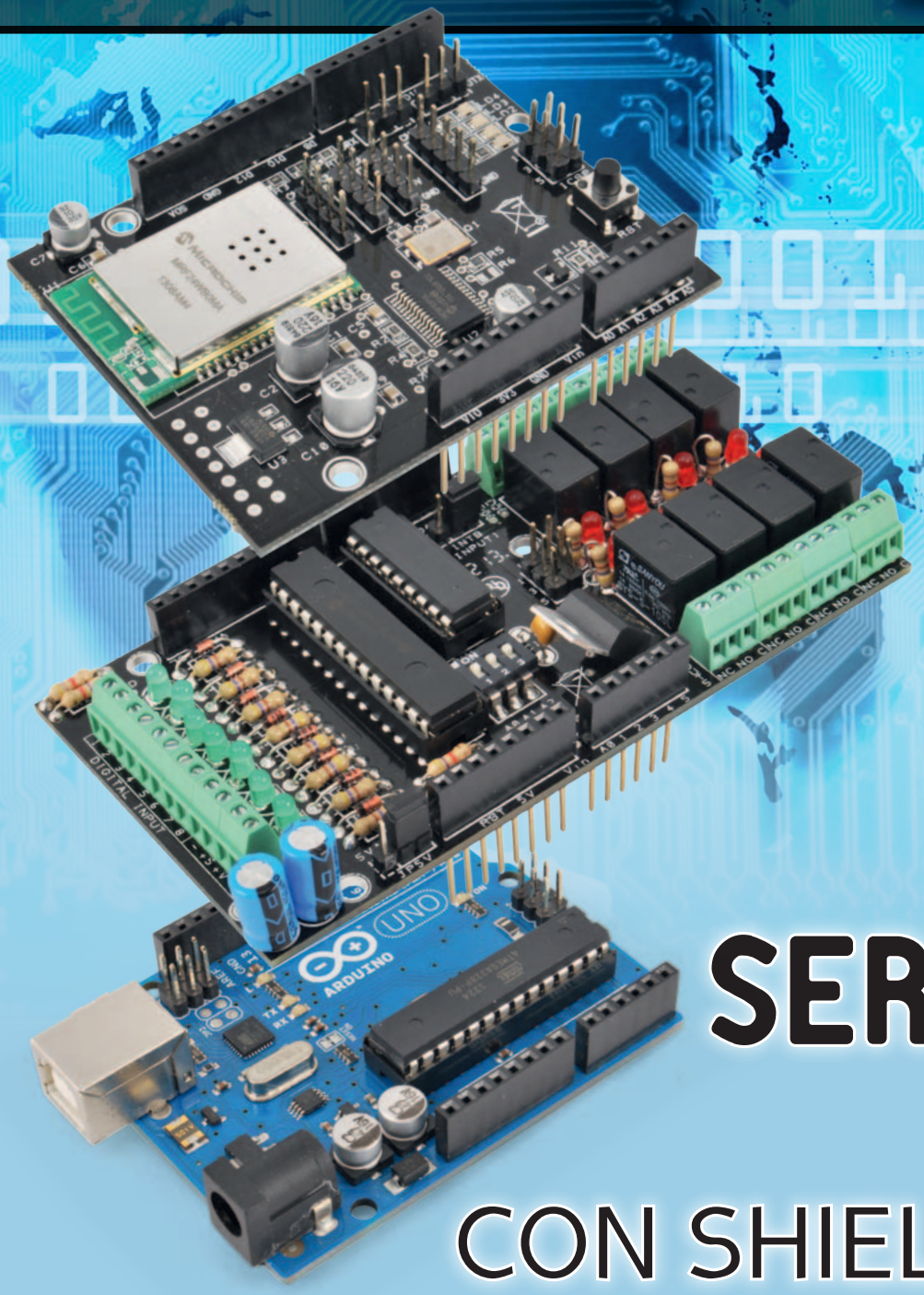


Conectamos nuestro shield WiFi con Arduino y realizamos, con la librería especifica HTTPLib, un Servidor Web Servidor que, con la ayuda de un shield de extensión E/S, lea estados lógicos y controle relés desde una página web.

SERVIDOR WEB CON SHIELD Wi-Fi

DANIELE DENARO

En una edición anterior ya hemos descrito el uso de la librería específicamente escrita por nosotros para utilizar el shield WiFi en la modalidad Cliente Web de un sitio y, más preciso, de un sitio público para el registro de datos recogidos por los sensores (en específico, de Xively). Como anunciamos en aquella ocasión, ahora explicaremos como utilizar el shield para realizar, junto con Arduino, la modalidad Servidor WebServidor. Cuando funciona como servidor web, a Arduino se puede acceder utilizando un navegador normal, a través del cual leer, por ejemplo, los valores de algunos sensores o activar salidas de relé.

En los ejemplos distribuidos con la librería HTTPLib está ya presente un sketch que realiza un Servidor Web, sin embargo en el ejemplo se utilizan las funciones que permiten construir y enviar páginas enteras que incluyen los valores a visualizar. Si se quiere

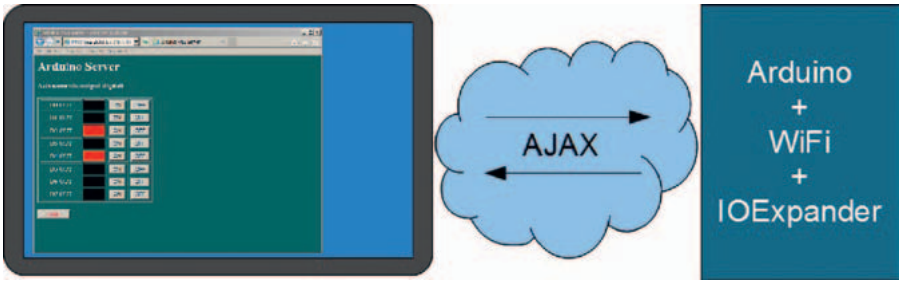


Fig. 1 *- Dialogo en background.*

comprobar cualquier cambio de los valores es necesario recargar la página con un refresco manual. También la activación digital o PWM se realiza enviando indirectamente los valores a través de un "modulo" (form), que restablece la página "refrescada".
Así pues, hemos pensado utilizar la librera WiFi con mecanismos más automáticos y eficientes como el uso de funciones Javascript insertadas en el código HTML, y el uso del dialogo navegador-servidor realizado a espaldas de la página a través de mecanismos AJAX (**Fig. 1**) o con el objeto Javascript XMLHttpRequest (ver la explicación en el recuadro apropiado).
Utilizar funciones Javascript inmersas en el código HTML significa, en sustancia, descargar sobre el navegador parte del rendimiento grafico de la página. El navegador esta normalmente activo sobre un hardware mucho más potente que Arduino y por tanto puede realizar automatismos y animaciones impensables de gestionar con la débil CPU de Arduino; además con esta solución se reduce el tráfico de red.
Como se ha dicho, mediante Javascript, Arduino descarga parte del código de ejecución sobre el navegador cliente. Naturalmente esto conlleva una mayor dificultad a la hora de depurar. Conviene construir y testear los script aparte y después integrarlos en el sketch.
Para diseñar páginas HTML y testear los script se pueden utilizar distintos programas comerciales como Dreamweaver de Adobe o FrontPage de Microsoft, pero también editores gratuitos tanto visuales como Eclipse, PageBreeze, KompZer (portable), como de texto como PSPad, Notepad++.
Para la depuración de los script se puede utilizar directamente el navegador, ya que todos los navegador tienen en sus menús la función "herramientas de desarrollo", a través de la cual se puede abrir una ventana para la ejecución paso a paso del código de scripting: en las versiones recientes de Internet Explorer a tales funciones se accede desde el menú *Herramientas*, con el comando *Herramientas de desarrollo* (o **F12** desde teclado) mientras en los más recientes Firefox se accede con el comando *Herramientas* > *Desarrollo web*. En casi todos los navegadores actuales se accede con F12.
Las nuevas funcionalidades implementadas en la librera HTTPLib (Versión 2.4) son de dos tipos: funciones para la construcción modular de las páginas HTML y función para el intercambio rápido de datos a utilizar con las peticiones AJAX (**Tabla 1**). La librería completa se puede descargar desde la web de la revista (**MWIFI24.zip**).
Las primeras han sido introducidas porque, al complicarse las páginas HTML que contienen funciones Javascript, es fácil saturar los 32 kB de memoria de programa, pero también porque a

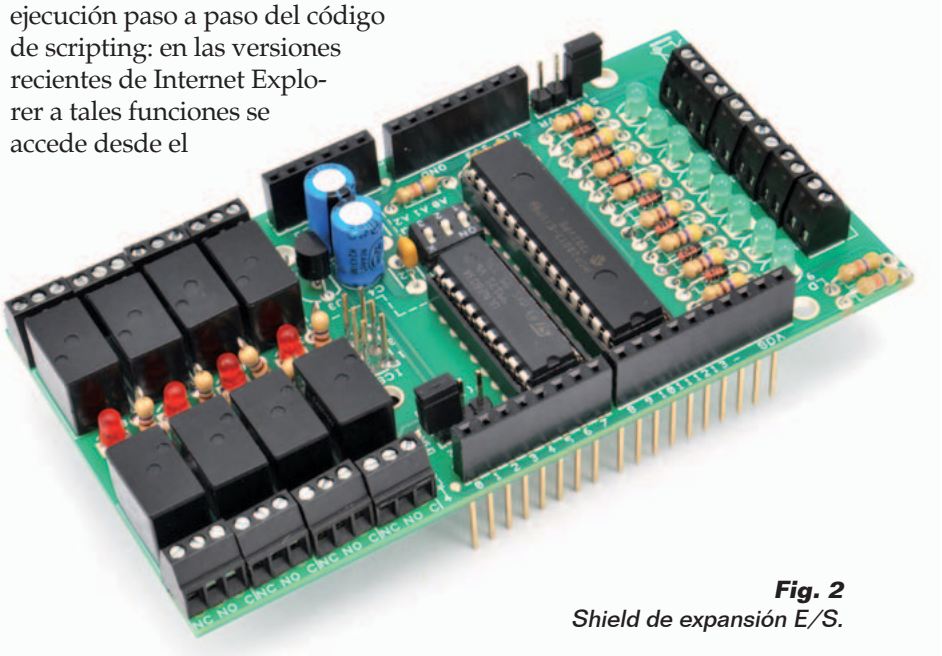


Fig. 2
Shield de expansión E/S.

Tabla 1 *- Nuevas funciones de la librería HTTPLib.*

Función	Descripción
void sendResponse(int sk,prog_char* amodule[],uint8_t nm); void sendDynResponse(int sk,prog_char* amodule[],uint8_t nm,int npar,char *param[]);	Responden a la "request" con una secuencia de módulos de código HTML memorizado en variables prog_char. (nm es el número de módulos, o la dimensión del array amodule[])
void sendShortResponse(int sk,char *data);	Responde a la "request" con un buffer de datos (en formato cadena)

Fig. 3 *- Menú del Servidor Web .*

menudo las funciones Javascript son comunes entre las distintas páginas. Por estas razones es mucho más cómodo estructurar las paginas HTML en bloques (por ejemplo: título, routine Javascript, body, footer) y componerlas según las exigencias de las paginas individuales.
El segundo tipo de función "response" ha sido introducida para enviar indirectamente pocos datos (a menudo en formato JSON) utilizando un paquete entero y no en pedazos (chunk) como se había pensado para las paginas HTML enteras; esto tiene como objetivo mejorar la eficiencia.

ADQUIRIR LOS INPUT Y GESTIONAR LOS OUTPUT DE LA PÁGINA

La aplicación aquí descrita prevé la actualización en la página web del estado de un cierto número de entradas y la posibilidad de modificar la condición de las salidas, actuando desde la página misma. Para realizar la aplicación, interactuando con el conjunto Arduino+WiFi Shield con un hardware especifico compuesto por un shield de expansión E/S montado sobre el Arduino, el cual nos permitirá leer el estado de un cierto número de líneas digitales y controlar dispositivos.
El shield para Arduino es básicamente el mismo que describimos en la edición 319 para Raspberry Pi, así que os remitimos a los esquemas incluidos en ese artículo. Recordaremos solo a las características principales del proyecto.
El shield se conecta con Arduino mediante un I²C bus (líneas SDA-SCL de Arduino) y por tanto requiere la librería Wire y la disponibilidad de los pines A4 y A5 del mismo Arduino; para su uso tiene una librería propia que permite la gestión simplificada de las E/S. Aún así se han previsto en la placa puentes gracias a los cuales es posible asignar el I²C-Bus a otras líneas (entendiéndose que es necesario modificar la librería como resultado).
El shield dispone de 8 entradas con pull-up (es decir, que en reposo son puestas a nivel alto por las correspondientes resistencias); poniendo a masa cada entrada, se enciende un LED verde de señalización colocado en serie con la resistencia pull-up. En la página web del servidor se usa una representación contraria: verde=nivel alto de entrada, negro=entrada a masa.
En el shield están disponibles ocho salidas a relé (disponibles tanto los contactos normalmente abiertos como los normalmente cerrados); para cada relé, se ha previsto un LED rojo de señalización del estado, colocado en paralelo a la bobina y alimentado por la correspondiente salida del driver de línea integrado (ULN2803).

Tabla 2 *- Disponibilidad de E/S del Servidor Web.*

Equipo I/O	E/S correspondientes
4 Input analógicos	A0, A1, A2, A3 de Arduino
8 entradas digitales	D0, D1, D2, D3, D4, D5, D6, D7 del shield de expansión E/S
8 salidas digitales	D0, D1, D2, D3, D4, D5, D6, D7 del shield de expansión E/S
5 salidas PWM	D5, D6, D9, D10, D11 de Arduino

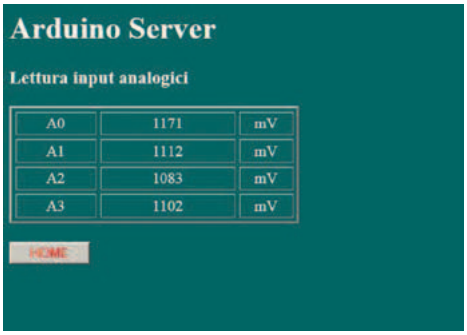


Fig. 4 *- Lectura entradas analógicos.*

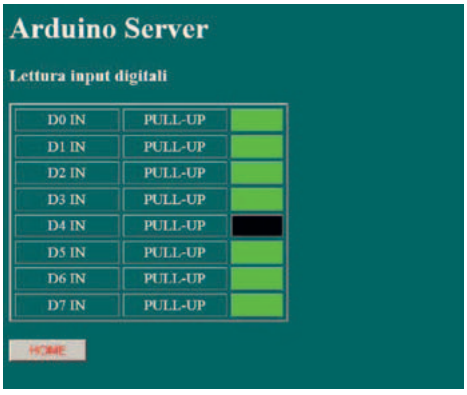


Fig. 5 *- Lectura entradas digitales..*

Fig. 6 *- Control salidas digitales.*

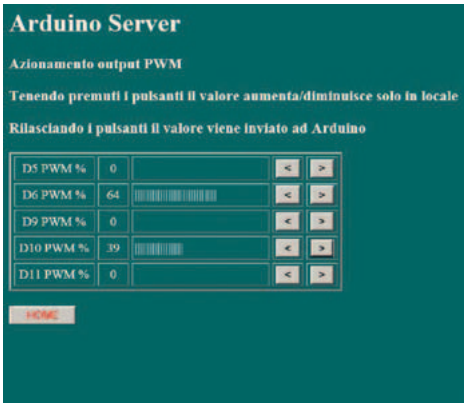


Fig. 7 *- Control salidas PWM.*

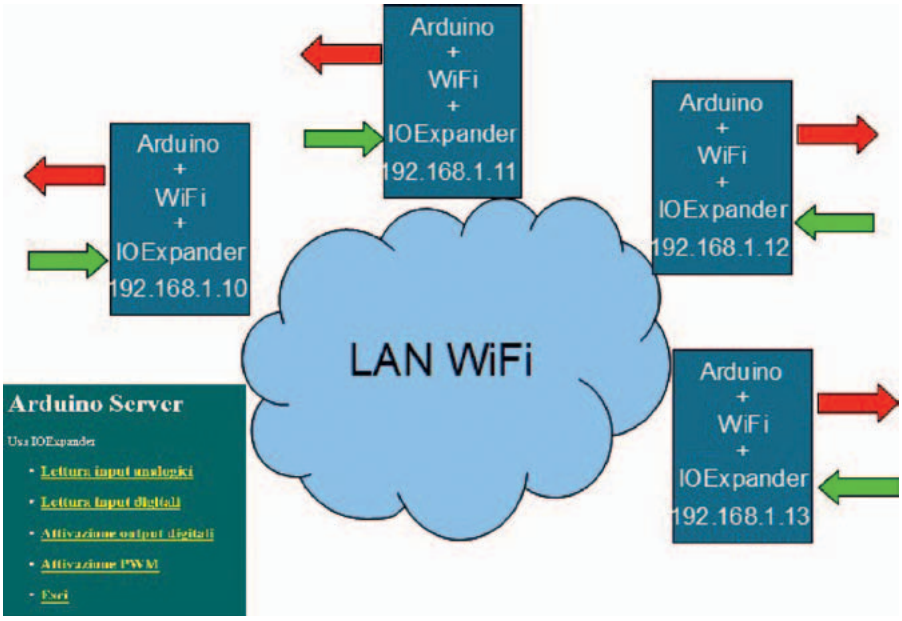


Fig. 8 *- Ejemplo LAN.*

Listado 1 - Módulo inicial con registro y estilo

```
prog_char Prologue[] PROGMEM=
"<html><head>"
"<title>Arduino Servidor WEB </title>" //Titulo ventana
"<style type='text/css'>"
"body,td,th {color: #FFF;}"          //Color caracteres
"body {background-color: #066;}"     //Color fondo
"a:link {color: #FF0;}"              //Color link
"a:visited {color: #FF0;}"           //Color link ya visitado
".on {background-color:#00FF00;}"    //Color pin input on (verde)
".off {background-color:#000;}"      //Color pin off
".onr {background-color:#FF0000;}"   //Color pin out on (rojo)
"</style>";
```

EL SERVIDOR WEB

Teniendo presente que A4 y A5 los utiliza el bus I²C y que D2, D3 y D7 los usa el shield WiFi, podemos integrar los restantes pines de Arduino para tener un servidor con un significativo número de Entradas/Salidas (**Tabla 2**).

Naturalmente el sketch propuesto puede servir como ejemplo para utilizar los mismos métodos de implementación y el dialogo AJAX con el Arduino o con el uso de diferentes shield de E/S. El aspecto del Servidor WEB es el que muestran las **Fig. 3**, **Fig.** 4, **Fig.** 5, **Fig.** 6, **Fig.** 7. Tanto el menú como las páginas de activación de entradas y salidas analógicas y digitales se refrescan automáticamente cada minuto; esto con el objetivo de evitar que el navegador cierre autónomamente la comunicación en el caso que se deje inactiva la página por un largo tiempo. Por lo demás, las paginas son fijas y son solo los valores (o los colores que representan estados on-off) cabiarán con un refresco oculto basado en las técnicas AJAX.

En el caso de lecturas analógicas, el refresco oculto actúa cada segundo mientras para las lecturas digitales se efectúa cada 300 mS.

En las páginas de activación digital, los relés son activados o desactivados mediante una petición oculta y la respuesta a esta última modifica el estado visualizado. En la página de activación PWM ha sido implementada una animación a través de Javascript: haciendo clic sobre el botón ">" o sobre el "<" el valor, y la barra que lo representa gráficamente, son modificados solo en local. Pero soltando el pulsador del mouse, el valor aplicado se envía a Arduino, el cual lo confirma con la propia respuesta.

De este modo, con intercambios de red no demasiado pesados en total, se realiza una ventana remota sobre las E/S de Arduino.

Tabla 3 *- Estructura del sketch.*

Función	Descripción
Define las variables globales	Nome accesspoint; password WiFi (eventual); user name y WEB password (en caso de acceso autentificado) y otro...
Paginas HTML en bloques	Bloque de registro y definición de estilos; bloque con creación y lanzamiento de objeto XMLHttpRequest (AJAX); bloques con Javascript para las paginas pedidas; bloques con la parte body de las paginas pedidas, bloque final.
Setup	Inicializa WiFi y I/OExpander, codifica el username-password (eventuales) para acceso autentificado, se conecta al punto de acceso y activa el modo servidor en puerto 80.
Loop	Si no está conectado aún, se conecta; si llega una petición responde. Tanto si se trata de una petición GET de página o de una petición POST a través AJAX, activa la función correspondiente que enviará una página o interactuará con la E/S y responderá con una shortResponse.
Funciones de conexion	Realizan la conexión con el punto de acceso y la condición de oyente sobre el puerto 80.
Funciones recurso (paginas pedidas)	Apilan los bloques de la página y la envían.
Funciones recursos para llamadas AJAX	Si se trata de una lectura (refresh), detectan los valores totales, los formatean en modo cadena JSON y los envían como shortResponse. Si se trata de activación, activan el pin correspondiente y envían oculto un refresh de todos los valores.

Listado 2 - Módulo AJAX

```
prog_char Ajax[] PROGMEM=
"<script type='text/javascript'>"
"var fbusy=0;"                  //Semáforo
"var REFRESH=null;"
"function waitReq()"            //Stop eventual refresh
"{if (REFRESH != null) clearInterval(REFRESH);return fbusy;}"
"function ajax(resource,cmd,pin,val)" //XMLHttpRequest
"{"
 "if (fbusy==1) return;" //Stop si aún activa precede petición
 "var params=null;"      //Inicialización eventuales parámetros
 "if (cmd!=null)"
 "{params=\"cmd=\"+cmd;"
  "if (pin!=null) params=params+\"&pin=\"+pin;"
  "if (val!=null) params=params+\"&val=\"+val;}"
 "Req=new XMLHttpRequest();"    //Creación objeto XMLHttpRequest
 "Req.onreadystatechange= function()"
 "{if ((Req.readyState == 4)&(Req.status == 200)){fbusy=0;setData(Req.responseText.trim());}};" //Callback function
 "Req.open('POST',resource,true);"
 "Req.timeout=1000;"                    //Timeout
 "Req.ontimeout = function() {fbusy=0;};"
 "fbusy=1;"                             // Set semáforo
 "if (params==null) Req.send();else Req.send(params);" //Envía
"}"
"</script>";
```

Listado 3 - Script para página lectura analógica

```
prog_char ScriptARead[] PROGMEM=
"<script type='text/javascript'>"
"var nr=4;"
"var data=new Array();" //ogg.JSON: array de 4 parejas num-cadena
"function update()"
 "{ajax('/getAnalog',null,null,null);}"
"function setData(resp)"
"{"
  "if (resp==null) return;"
  "data=eval(resp);"    //cadena [[nnn,"aa"],...] en array data
  "for (i=0;i<4;i++)"   //actualiza valores en los campos
  "{"
    "el=document.getElementById('AV'+i);"
    "el.innerHTML=data[i][0];"
    "el=document.getElementById('AU'+i);"
    "el.innerHTML=data[i][1];"
  "}"
 "}"
"function make()"       // construye la Tabla
 "{document.write(\"<table width='300' border='2' cellpadding='2' cellspacing='4'>\");"
 "for (i=0;i<nr;i++)"  //Tabla de 4 líneas de 3 columnas
  "{"
    "document.write(\"<tr align='center'>\");"
    "document.write(\"<td width='30%'>A\"+i+\"</td>\");"
    "document.write(\"<td width='50%' id='AV\"+i+\"'> </td>\");"
    "document.write(\"<td id='AU\"+i+\"'> </td>\");"
    "document.write(\"</tr>\");}"
    "document.write(\"</table>\");"
  "}"
"</script>";
```

no en tiempo real y sin utilizar protocolos propietarios vía socket. El sketch completo (y comentado) que realiza las funciones descritas se puede desde nuestra web *www.nuevaelectronica.com.*

Una posible evolución del proyecto podría consistir en tener distintos servidores Web de estos basados en Arduino conectados en red local a un punto de acceso (AP) y gestionadas por un puesto PC con navegador (**Fig.8**). El uso en red geográfica (WAN) es sin embargo más complejo y es además desaconsejable porque no es posible implementar niveles de seguridad HTTPS con Arduino, mientras que en red local la protección WPA unida a la autenticación web proporciona una razonable seguridad en la gestión de sensores y sobre todo de los actuadores. Para la red WAN se puede pensar en poner una Raspberry Pi como interfaz con el exterior.

Pero ahora veamos la estructura general y algunos aspectos significativos del sketch; esto

Listato 4 - setup()

```
void setup()
{
  Serial.begin(9600);

  WIFI.begin();               // startup wifi shield

  pinMode(5,OUTPUT);          // pin 5 como output PWM
  pinMode(6,OUTPUT);          // pin 6 como output PWM
  pinMode(9,OUTPUT);          // pin 9 como output PWM
  pinMode(10,OUTPUT);         // pin 10 como output PWM
  pinMode(11,OUTPUT);         // pin 11 como output PWM

  IOExpanderSetting();        // inicialización IOExpander

                              // si requiere autenticación
                              // codifica username-password
  if (FAUTH) Wkey=WIFI.codeWebKey(WUSER,WPASSW);

  netConnection();            // entra en red
}
```

Listato 5 - loop()

```
void loop()
{
  char *req=NULL;
                  // si no está conectado ya prueba a conectarse
  if(!fc) {delay (10000);netConnection();}
                  // sino procede
  if(fc)
  {
                  // si no está inicializado como servidor inicializa
    if (!fserver) serverStartup();
    if (!fs){listenConnect();}  //escucha
    if (fs)                     // si llega petición la gestiona
    {
                // de manera contrario si es usada o no autentificación
     if(FAUTH)
          req=WIFI.getRequest(csocket,10,rs,Wkey);
     else
          req=WIFI.getRequest(csocket,10,rs);
    }
  }
}
```

Listato 6 - Matriz de WEBRES (asociación recurso-función)

```
WEBRES rs[]=
{
 {"/index",PIndex},        //paginas (pedidas con un GET)
 {"/AnalogRead",PAnalogR},
 {"/DigitalRead",PDigitalR},
 {"/DigitalWrite",PDigitalW},
 {"/PWMWrite",PPwmW},
 {"/End",PEnd},

 {"/getAnalog",getAna},    //recursos pedidos por XMLHttpRequest
 {"/getDigital",getDig},
 {"/setDigital",setDig},
 {"/setPWM",setPwm}
};
```

es muy consistente debido a las distintas librerías incluidas y a las paginas HTML memorizadas en la memoria del programa. Su dimensión está al límite de los 32 k disponibles (ocupa casi 31 k). En realidad el código verdadero y propio es bastante pequeño. El sketch permite el uso de puntos de acceso no protegidos o protegidos por password (WPA). En este caso si encuentra una keyword ya memorizada en memoria EEPROM, la utiliza; sino se conecta a través de password y memoriza en EEPROM la keyword calculada. En el caso que use la keyword y esta sea errónea (porque ha cambiado, o porque se ha cambiado el punto de acceso) se conecta a través de password y actualiza la keyword en EEPROM.

Recordamos que el acceso a través de keyword anteriormente calculada es muy rápido, mientras que si para obtener el acceso, el sistema debe calcular la keyword, el cálculo mimo puede requerir incluso un minuto y por consiguiente el proceso de acceso es definitivamente más lento. Ya desde la versión 2.2 de la librería se ha introducido la gestión de la keyword WPA en EEPROM; además se ha previsto también el uso de autenticación WEB a través de la combinación usuario-password.

En la **Tabla 3** podemos ver la estructura del sketch.

Veamos ahora algunos extractos significativos de la fuente, recordando que el sketch completo puede ser descargado de la web www.nuevaelectrónica.com

En el **Listado 1** se muestra el bloque HTML común a todas las páginas y relativo al registro y a la definición de estilos.

En el **Listado 2** se muestra el bloque HTML común a todas las páginas y relativo a las funcionalidades AJAX. A propósito de esto, observamos que utilizando un comportamiento asíncrono para **XMLHttpRequest**, estamos obligados a regular el uso del mismo a través de un semáforo, porque Arduino es monotarea y puede responder a una sola petición a la vez.

La función "Javascript ajax()" tiene cuatro parámetros: el recurso pedido y tres eventuales parámetros correspondientes a un código comando, un numero de pin y un valor. Estos tres parámetros serán leídos por la función recurso sobre

AJAX y XMLHttpRequest

Asynchronous Javascript And XML (AJAX) es una tecnología nacida en el 2005 (por obra de J.J.Garret) para ir en contra del aumento de interacciones requerido por el navegador al servidor. Hasta entonces, cada petición era contestada por el envió de una página entera (estática o dinámica, es decir, construida al momento). Claramente este comportamiento, además ralentizar la red, no permitía una verdadera interacción entre usuario y el sitio web. La solución ha sido la definición de una clase XMLHttpRequest insertada en las librerías Javascript implementadas en todos los navegador. Una instancia de esta clase permite enviar una petición al servidor sin salir de la página, o actuando en background. El objeto XMLHttpRequest se preocupa también de restituir la respuesta demandada por el servidor (normalmente en modo asíncrono después de haber definido una callback function). En sustancia, en el ámbito de un script Javascript:

- se instancia un objeto XMLHttpRequest;
- se usa su método Open para pasarles la URL requerida y el método a utilizar (POST o GET);
- se define la función callback lanzada a la llegada de la respuesta (existe también la modalidad síncrona pero es desaconsejable);
- se envía la petición y la función de callback se ocupara de utilizar la respuesta.

El prefijo XML había sido puesto porque se preveía gestionar la respuesta en tal formato. En realidad la respuesta puede ser en formato plain/text, y normalmente se utiliza el formateo JavaScript Object Notation (JSON), o una cadena que pueda representar un objeto Javascript (variable, objeto, array, array de objetos, etc.) que pueda ser interpretado por la funcione eval() de Javascript y por tanto utilizado como tal en el interior de un script.

Arduino para ejecutar la petición. La función "callback" activada al recibir la respuesta tiene un nombre único ("setData") pero es implementada de manera distinta en las páginas, y tienen también el deber de resetear el semáforo para habilitar la petición siguiente.
En el **Listado 3** se muestra uno de los bloques con las funciones Javascript relativas a la construcción y gestión de una página (en el caso de la página de lectura de los input analógicos). Hay que observar que la implementación de la función "setData", callback de XMLHttpRequest, y la función "make", común a todas las paginas, tiene la tarea de construir la página a su llegada. De hecho, debido a la forma intrínsecamente tabular de las páginas, se ha preferido automatizar con un ciclo "for" la construcción de las tablas, ahorrando código HTML en beneficio de la pobre memoria de Arduino. Vayamos ahora a las funciones sobre Arduino: en el **Listado 4** se muestra la función "setup()" mientras en el **Listado 5** la función "loop()". La función fundamental lanzada en loop() es la "getRequest". La función "getRequest" se encarga de recibir el mensaje con la petición, decodificarla y de activar el recurso objeto de la petición. Para asociar el nombre del recurso a la función Arduino dedicada a él, utiliza una matriz de correspondencias (**Listado 6**). Una parte de estas responden con páginas enteras. Las otras son aquellas peticiones de la actividad en background AJAX. Veamos finalmente un ejemplo de respuesta a una petición de página (ilustrado en el **Listado 7**) y la respuesta a una petición de refresco de datos (es el código contenido en el **Listado 8**). *(186095)*

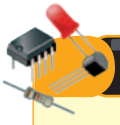
Listato 7 - Función para la página de lectura analógica

```
void PAnalogR(char *query)
{
   module[0]=Prologue;    //pagina compuesta por 5 módulos
   module[1]=Ajax;
   module[2]=ScriptARead;
   module[3]=PageAnaRead;
   module[4]=Footer;
   WIFI.sendResponse(csocket,module,5);
}
```

Listato 8 - función para refresh (llamada por XMLHttpRequest)

```
prog_char jsonFormat1[] PROGMEM=
"[[%d,%s],[%d,%s],[%d,%s],[%d,%s]]";  //formato datos a enviar

void getAna(char *query)
{
  int AV0=(analogRead(0)*4.88);char AU0[]="'mV'";
  int AV1=(analogRead(1)*4.88);char AU1[]="'mV'";
  int AV2=(analogRead(2)*4.88);char AU2[]="'mV'";
  int AV3=(analogRead(3)*4.88);char AU3[]="'mV'";
  char json[50];  // buffer da spedire
  snprintf_P(json,50,jsonFormat1,AV0,AU0,AV1,AU1,AV2,AU2,AV3,AU3);
  WIFI.sendShortResponse(csocket,json); //envía buffer datos
}
```



el MATERIAL

Los shield a los que nos referimos en este artículo ya han sido presentados en esta revista: shield de expansión E/S para Arduino (cod. FT1079K, en kit, 34,00 Euros) y Shield Wi-Fi para Arduino (cod. FT1076M, montado, 56,00 Euros). Recordamos que la tarjeta Arduino Uno Rev. 3 cuesta 24,50 Euros.

Precios IVA incluido sin gastos de envío.
Puede hacer su pedido en:
www.nuevaelectronica.com
pedidos@nuevaelectronica.com