```
//--------------------------------------------------------------------------------------------------
// Copyright (C) 2009 - 2012, MyDoorOpener.com
//--------------------------------------------------------------------------------------------------
//
// Release Notes:
//
// v1.0 [02/23/2010] - Initial release
//
// v1.1 [03/02/2010] - Added support for up to three garage doors.
//
// v1.2 [08/11/2010] - Deprecated HTTP POST in favor of HTTP GET parameters.
//                   - Added additional serial debugging.
//
// v2.0 [05/11/2012] - Updated to support unlimited number of devices (used to be max 3).
//                   - Updated to support Arduino v1.0 IDE.
//
// v2.2 [12/15/2012] - Updated to support SMTP email notifications
//                   - Updated to support SMS notifications (via carrier provided SMTPtoSMS gateway)
//                   - Updated to support iOS push notifications (via Prowl)
//                   - Added logic to support multiple status reading strategies (3v5v, NC, NO, etc)
//                   - Relocated most string literals to flash memory
//                   - Certified compatibility with ArduinoEthernet (all-in-one) and DFRobot RelayShield hardware
//                   - Certified compatibility with Arduino v1.0.3 IDE
//
//--------------------------------------------------------------------------------------------------

// Uncomment to turn ON serial debugging
//#define MYDOOROPENER_SERIAL_DEBUGGING 1
//#define WEBDUINO_SERIAL_DEBUGGING 1
//#define NOTIFICATIONS_SERIAL_DEBUGGING 1
//#define SMTP_SERIAL_DEBUGGING 1
//#define PUSH_SERIAL_DEBUGGING 1

#include <Arduino.h>
#include <SPI.h>
#include <Time.h>
#include <Ethernet.h>
#include <WebServer.h>
#include <aes256.h>
#include <Wire.h>

//#define I2C_ADDR  0x20  // 0x20 is the address with all jumpers removed
//*****************************************************************
// Global configuration (adjust to reflect your setup)
//*****************************************************************

// EthernetShield IP address (DHCP reserved, never allocated to anyone else). This is the internal
// network IP address you want your Arduino assigned. This is not the address you will be accessing
// your Arduino from the iPhone application or internet ... You need to configure NAT forwarding
// on your home router to do that. See http://en.wikipedia.org/wiki/Port_forwarding for more details.

static const uint8_t ip[4] = { 192, 168, 1, 10 };
static const uint8_t mydns[4] = { 192, 168, 1, 254};
```

```
static const uint8_t gateway[4] = { 192, 168, 1, 254 };

// password required for operating door [max length = 16] (status fetching doesn't require password). This
// must match the password you set in the iPhone application (be careful as case is sensitive).

#define PASSWORD "*****************************"

//*****************************************************************
// Device/Door configuration (relais and sensors)
//*****************************************************************

// *** IMPORTANT NOTE ***
//
// THE ARDUINO ETHERNET SHIELD RESERVES DIGITAL PINS 10, 11, 12, 13 AS WELL AS
// ANALOG PINS 0, 1, THEREFORE YOU SHOULD NOT USE ANY OF THOSE PINS FOR YOUR DEVICE(S)
OR DOOR(S)

// open/close trigger relay should be connected to these digital output pins (digitalWrite).
// Adjust to match the number of devices you have hooked up (examples provided below in comment) ...

//static const uint8_t relayPins[] = { 2 }; // single device at pin #2
static uint8_t relayPins[] = { 2, 3 }; // select if using DFRobot RelayShield
//static uint8_t relayPins[] = { 2, 3 }; // two devices at pins #2 and #3
//static uint8_t relayPins[] = { 2, 3, 4, 5, }; // even more devices at pins #2, #3, #4, etc ...

// status contact should be connected to these analog input pins (anologRead).
// Adjust to match the number of devices you have hooked up (examples provided below in comment) ...

//static const uint8_t statusPins[] = { 3 }; // single device at pin #3
static uint8_t statusPins[] = { 2, 3 }; // select if using DFRobot RelayShield
//static uint8_t statusPins[] = { 2, 3 }; // two devices at pins #2 and #3
//static uint8_t statusPins[] = { 2, 3, 4, 5, }; // even more devices at pins #2, #3, #4, etc ...

// status reading strategy (uncomment only one ... the one that reflects how you want status to be interpreted)

//#define STATUS_STRATEGY_3VCLOSED_5VOPENED // initial approach - uses analogRead combined with
STATUS_OPEN_TRESHOLD (opened == +5v, closed == +3v)
//#define STATUS_STRATEGY_5VCLOSED_3VOPENED // alternate approach - uses analogRead combined with
STATUS_OPEN_TRESHOLD (opened == +3v, closed == +5v)
//#define STATUS_STRATEGY_NORMALLY_CLOSED // classic door sensor - uses digitalRead to interpret
door/device status (opened == high-impedance, closed == GND)
#define STATUS_STRATEGY_NORMALLY_OPENED // alternate approach - uses digitalRead to interpret
door/device status (opened == GND, closed == high-impedance)

// Analog boundary value (0-1023) used to distinguish between device/door status == opened and closed. Only
applicable
// when STATUS_STRATEGY_3VCLOSED_5VOPENED or STATUS_STRATEGY_5VCLOSED_3VOPENED is
being used.

#define STATUS_OPEN_TRESHOLD 1000

//*****************************************************************
// Notifications
//*****************************************************************
```

```
// if defined, will fire a notification when any door/device stays open more than the specified number of minutes
//#define NOTIFICATIONS_WATCHDOG_MINUTES  5

// if defined, will fire a notification every time and as soon as any door/device gets opened
#define NOTIFICATIONS_OPEN

//**********************************************************************
// iOS push notifications (via Prowl)
//**********************************************************************

// if defined, will fire notifications using iOS push notifications (uncomment to turn ON)
#define PUSH_NOTIFICATIONS

#if defined(PUSH_NOTIFICATIONS)

// after installing Prowl (iTunes store) on your iPhone, set to match the Prowl API key that was
// assigned for your iPhone http://www.prowlapp.com/api_settings.php
static const char prowlApiKey[] = "7032cf78b800163a8369a959ee2f636b1653ae77";

// set to Prowl HTTP server name (typically api.prowlapp.com)
static const char prowlServerName[] = "api.prowlapp.com";

// set to Prowl HTTP port number (typically port 80)
static const int prowlServerPort = 80;

// set to Prowl API base url (typically /publicapi/add)
static const char prowlApiBaseUrl[] = "/publicapi/add";

#endif

//**********************************************************************
// SMS notifications (via carrier provided SMS gateway, relies on SMTP)
//**********************************************************************

// if defined, will fire notifications using SMS notifications (uncomment to turn ON - rember to adjust SMTP settings
too)
//#define SMS_NOTIFICATIONS

//#if defined(SMS_NOTIFICATIONS)

// using http://en.wikipedia.org/wiki/List_of_SMS_gateways set to match your mobile carrier and phone number
static const char smtpToForSms[] = "your-mobile-number@your-carrier-gateway.com";

//#endif

//**********************************************************************
// SMTP email notifications (also used for SMS notifications)
//**********************************************************************

// if defined, will fire notifications using SMTP notifications (uncomment to turn ON)
//#define SMTP_NOTIFICATIONS

//#if defined(SMTP_NOTIFICATIONS) || defined(SMS_NOTIFICATIONS)
```

```
// set to your ISP's SMTP server name
static const char smtpServerName[] = "aspmx.l.google.com";

// set to your ISP's SMTP server port (typically port 25)
static const int smtpServerPort = 25;

// set to the email address you want notifications sent to
static const char smtpToForEmail[] = "freddejong64@gmail.com";

//#endif

//**************************************************************************
// Misc configuration constants (should not require any changes)
//**************************************************************************

// EthernetShield MAC address.

static uint8_t mac[6] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

// Arduino HTTP server listening port number.

WebServer webserver("", **********);

// number of milliseconds relay pin will be held high when triggered.

#define RELAY_DELAY 1000

// misc size constants

#define HTTP_PARAM_NAME_SIZE 16
#define HTTP_PARAM_VALUE_SIZE 64

#define PASSWORD_HEX_SIZE 32
#define PASSWORD_SIZE 16
#define AES256_CRYPTO_KEY_SIZE 32
#define CHALLENGE_TOKEN_SIZE 16

//-----------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------
#if defined(PUSH_NOTIFICATIONS)

//-----------------------------------------------------------------------------------------------
void getUrlEncoded(const char* msg, char* encodedMsg)
{
  const char hex[] = "0123456789abcdef";

  while(*msg != '\0')
  {
    if(('a' <= *msg && *msg <= 'z') || ('A' <= *msg && *msg <= 'Z') || ('0' <= *msg && *msg <= '9'))
    {
      *encodedMsg = *msg;
      encodedMsg++;
    }
```

```
    else
    {
      *encodedMsg = '%';
      encodedMsg++;

      *encodedMsg = hex[*msg >> 4];
      encodedMsg++;

      *encodedMsg = hex[*msg & 15];
      encodedMsg++;
    }

    msg++;
  }
}

EthernetClient pushClient;

//--------------------------------------------------------------------------------------------------
void notifyViaPush(const char* subject, const char* body)
{
  #if defined(PUSH_SERIAL_DEBUGGING)
    Serial.print(F("*** Push - server name: '"));
    Serial.print(prowlServerName);
    Serial.print(F("' - "));
    Serial.print(F("apiKey: '"));
    Serial.print(prowlApiKey);
    Serial.print(F("' - "));
    Serial.print(F("subject: '"));
    Serial.print(subject);
    Serial.print(F("' - "));
    Serial.print(F("body: '"));
    Serial.print(body);
    Serial.println(F("' ***"));
  #endif

  if (pushClient.connect(prowlServerName, prowlServerPort))
  {
    #if defined(PUSH_SERIAL_DEBUGGING)
      Serial.println(F("*** Push - connection established ***"));
    #endif

    pushClient.print(F("GET "));

    pushClient.print(prowlApiBaseUrl);
    pushClient.print(F("?apikey="));
    pushClient.print(prowlApiKey);
    pushClient.print(F("&application=MyDoorOpener"));
    pushClient.print(F("&url=mydooropener%3A%2F%2Fstatus"));

    char encodedBuffer[100] = "";

    pushClient.print(F("&event="));
    memset(encodedBuffer, 0, 100);
```

```
    getUrlEncoded(subject, encodedBuffer);
    pushClient.print(encodedBuffer);

    pushClient.print(F("&description="));
    memset(encodedBuffer, 0, 100);
    getUrlEncoded(body, encodedBuffer);
    pushClient.print(encodedBuffer);

    pushClient.println(F(" HTTP/1.1"));

    pushClient.print(F("Host: "));
    pushClient.println(prowlServerName);

    pushClient.print(F("Accept: *"));
    pushClient.print(F("/"));
    pushClient.println(F("*"));
    pushClient.println(F("Connection: close"));

    pushClient.println();

    pushClient.stop();

    #if defined(PUSH_SERIAL_DEBUGGING)
      Serial.println(F("*** Push - connection stopped ***"));
    #endif
   }

  #if defined(PUSH_SERIAL_DEBUGGING)
    Serial.println(F("*** Push - completed ***"));
  #endif
}

#endif

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
#if defined(SMTP_NOTIFICATIONS) || defined(SMS_NOTIFICATIONS)

EthernetClient smtpClient;

//--------------------------------------------------------------------------------------------------
void notifyViaEmail(const String& to, const String& subject, const String& body)
{
  #if defined(SMTP_SERIAL_DEBUGGING)
    Serial.print(F("*** SMTP - server name: '"));
    Serial.print(smtpServerName);
    Serial.print(F("' - "));
    Serial.print(F("to: '"));
    Serial.print(to);
    Serial.print(F("' - "));
    Serial.print(F("subject: '"));
    Serial.print(subject);
    Serial.print(F("' - "));
    Serial.print(F("body: '"));
```

```
    Serial.print(body);
    Serial.println(F("' ***"));
  #endif

  if (smtpClient.connect(smtpServerName, smtpServerPort))
  {
    #if defined(SMTP_SERIAL_DEBUGGING)
      Serial.println(F("*** SMTP - connection established ***"));
    #endif

    smtpClient.println(F("HELO relai.mydooropener.com"));

    smtpClient.print(F("MAIL FROM:"));
    smtpClient.println(F("noreply@mydooropener.com"));

    smtpClient.print(F("RCPT TO:"));
    smtpClient.println(to);

    smtpClient.println(F("DATA"));

    smtpClient.print(F("SUBJECT: "));
    smtpClient.println(subject);

    smtpClient.println();

    smtpClient.print(body);
    smtpClient.print(F(" "));
    smtpClient.println(F("mydooropener://status"));

    smtpClient.println(F("."));
    smtpClient.println(F("."));

    smtpClient.println(F("QUIT"));

    smtpClient.stop();

    #if defined(SMTP_SERIAL_DEBUGGING)
      Serial.println(F("*** SMTP - connection stopped ***"));
    #endif
  }

   #if defined(SMTP_SERIAL_DEBUGGING)
    Serial.println(F("*** SMTP - completed ***"));
  #endif
}

//--------------------------------------------------------------------------------------------------
void notifyViaEmail(const char* subject, const char* body)
{
  notifyViaEmail(smtpToForEmail, subject, body);
}

#endif
```

```
//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
#if defined(SMS_NOTIFICATIONS)

//--------------------------------------------------------------------------------------------------
void notifyViaSms(const char* subject, const char* body)
{
  notifyViaEmail(smtpToForSms, subject, body);
}

#endif

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
void configureStatusPin(int pinNumber)
{
  #if defined(STATUS_STRATEGY_3VCLOSED_5VOPENED) ||
defined(STATUS_STRATEGY_5VCLOSED_3VOPENED)
    pinMode(pinNumber, INPUT);
  #elif defined(STATUS_STRATEGY_NORMALLY_CLOSED) ||
defined(STATUS_STRATEGY_NORMALLY_OPENED)
    pinMode(pinNumber+14, INPUT_PULLUP); // addressing analog pins as digital pins (+14)
  #endif
}

//--------------------------------------------------------------------------------------------------
boolean isOpen(int pinNumber)
{
  #if defined(STATUS_STRATEGY_3VCLOSED_5VOPENED) ||
defined(STATUS_STRATEGY_5VCLOSED_3VOPENED)
    int status = analogRead(pinNumber);
  #elif defined(STATUS_STRATEGY_NORMALLY_CLOSED) ||
defined(STATUS_STRATEGY_NORMALLY_OPENED)
    int status = digitalRead(pinNumber+14); // addressing analog pins as digital pins (+14)
  #endif

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.print(F("*** isOpen - status value for pin: '"));
    Serial.print(pinNumber);
    Serial.print(F("' is '"));
    Serial.print(status);
  #endif

  boolean retVal = false;

  #if defined(STATUS_STRATEGY_3VCLOSED_5VOPENED)
    retVal = status >= STATUS_OPEN_TRESHOLD;
  #elif defined(STATUS_STRATEGY_5VCLOSED_3VOPENED)
    retVal = status <= STATUS_OPEN_TRESHOLD;
  #elif defined(STATUS_STRATEGY_NORMALLY_CLOSED)
    retVal = status == LOW;
  #elif defined(STATUS_STRATEGY_NORMALLY_OPENED)
    retVal = status == HIGH;
  #endif
```

```
  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.print(F("' returing: '"));
    Serial.print(retVal ? F("Opened") : F("Closed"));
    Serial.println(F("' ***"));
  #endif

  return retVal;
}

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
void output(WebServer &server, char* data, bool newLine)
{
  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    if (newLine)
      Serial.println(data);
    else
      Serial.print(data);
  #endif

  if (newLine)
    server.println(data);
  else
    server.print(data);
}

//--------------------------------------------------------------------------------------------------
void output(WebServer &server, int number, bool newLine)
{
  char str[10] = "";
  itoa(number, str, 10);

  output(server, str, newLine);
}

//--------------------------------------------------------------------------------------------------
void webRequestHandler(WebServer &server, WebServer::ConnectionType type, char *url, bool isUrlComplete)
{
  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.println(F("*** WebServerRequestHandler begin ***"));
  #endif

  // holder for submitted password (as hex)

  char submitPassword[PASSWORD_HEX_SIZE + 1];
  memset(&submitPassword, 0, sizeof(submitPassword));

  // door on which open/close action is to be carried out. If unspecified, assume door #1

  char relayPinAsString[10];
  memset(&relayPinAsString, 0, sizeof(relayPinAsString));
  int relayPin = -1;
```

```
// holder for current challenge token value. The following must not be
// initialized (memset) because it is static and must persist across HTTP calls

static char currentChallengeToken[CHALLENGE_TOKEN_SIZE + 1] = "";

// handle HTTP GET params (if provided)

char name[HTTP_PARAM_NAME_SIZE + 1];
char value[HTTP_PARAM_VALUE_SIZE + 1];

// process all HTTP GET parameters

if (type == WebServer::GET)
{
 #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
  Serial.println(F("*** GET Request ***"));
 #endif

 while (url && strlen(url))
 {
  // process each HTTP GET parameter, one at a time

  memset(&name, 0, sizeof(name));
  memset(&value, 0, sizeof(value));

  server.nextURLparam(&url, name, HTTP_PARAM_NAME_SIZE, value, HTTP_PARAM_VALUE_SIZE);

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
   Serial.print(F("*** HTTP GET PARAM - name: '"));
   Serial.print(name);
   Serial.print(F("' - "));
   Serial.print(F("value: '"));
   Serial.print(value);
   Serial.println(F("' ***"));
  #endif

  // keep hold of submitted encrypted hex password value

  if (strcmp(name, "password") == 0)
   strcpy(submitPassword, value);

  // keep hold of relay pin which should be triggered

  else if (strcmp(name, "relayPin") == 0)
  {
   strcpy(relayPinAsString, value);
   relayPin = atoi(relayPinAsString);
  }
 }
}

// the presence of an HTTP GET password param results in a request
// to trigger the relay (used to be triggered by an HTTP request of type POST)
```

```
if(strlen(submitPassword) > 0)
{
 #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
  Serial.print(F("*** submitPassword: '"));
  Serial.print(submitPassword);
  Serial.println(F("' ***"));
 #endif

 // decrypt password using latest challenge token as cypher key

 uint8_t cryptoKey[AES256_CRYPTO_KEY_SIZE + 1];
 memset(&cryptoKey, 0, sizeof(cryptoKey));

 for (int i = 0; i < strlen(currentChallengeToken); ++i)
  cryptoKey[i] = currentChallengeToken[i];

 uint8_t password[PASSWORD_SIZE + 1];
 memset(&password, 0, sizeof(password));

 // convert password from hex string to ascii decimal

 int i = 0;
 int j = 0;
 while (true)
 {
  if (!submitPassword[j])
   break;

  char hexValue[3] = { submitPassword[j], submitPassword[j+1], '\0' };
  password[i] = (int) strtol(hexValue, NULL, 16);

  i += 1;
  j += 2;
 }

 // proceed with AES256 password decryption

 aes256_context ctx;
 aes256_init(&ctx, cryptoKey);
 aes256_decrypt_ecb(&ctx, password);
 aes256_done(&ctx);

 char passwordAsChar[PASSWORD_SIZE + 1];
 memset(&passwordAsChar, 0, sizeof(passwordAsChar));

 for (int i = 0; i < sizeof(password); ++i)
  passwordAsChar[i] = password[i];

 #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
  Serial.print(F("*** passwordAsChar: '"));
  Serial.print(passwordAsChar);
  Serial.println(F("' ***"));
 #endif
```

```
  // if password matches, trigger relay

  if (strcmp(passwordAsChar, PASSWORD) == 0)
  {
    #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
      Serial.println(F("*** password matched ***"));
    #endif

    // trigger relay pin and hold it HIGH for the appropriate number of milliseconds

    if (relayPin != -1)
    {
      #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
        Serial.println(F("*** relay triggered ***"));
      #endif

      digitalWrite(relayPin, HIGH);
      delay(RELAY_DELAY);
      digitalWrite(relayPin, LOW);
    }
  }
}

// write HTTP headers

server.httpSuccess("text/xml; charset=utf-8");

#if defined(MYDOOROPENER_SERIAL_DEBUGGING)
  Serial.println(F("*** XML output begin ***"));
#endif

// write opening XML element to output stream

output(server, "<?xml version=\"1.0\"?>", true);
output(server, "<myDoorOpener>", true);

// write current door status

for (int i = 0; i < sizeof(statusPins); ++i)
{
  output(server, "<status statusPin=\"", false);
  output(server, statusPins[i], false);
  output(server, "\">", false);

  // write current open/close state to output stream

  output(server, (char*)(isOpen(statusPins[i]) ? "Opened" : "Closed"), false);
  output(server, "</status>", true);
}

// re-generate new challenge token

sprintf(currentChallengeToken, "Cyber%i%i%i", hour(), minute(), second());
```

```
  // write challenge token to output stream

  output(server, "<challengeToken>", false);
  output(server, currentChallengeToken, false);
  output(server, "</challengeToken>", true);

  // write closing XML element to output stream

  output(server, "</myDoorOpener>", true);

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.println(F("*** XML output end ***"));
    Serial.println(F("*** WebServerRequestHandler end ***"));
  #endif
}

//-----------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------
#if defined(NOTIFICATIONS_WATCHDOG_MINUTES)

//-----------------------------------------------------------------------------------------------
void watchDogNotificationsHandler()
{
  static time_t initialOpen = NULL;
  time_t latestOpen = NULL;

  static boolean notificationSent = false;
  boolean openDetected = false;

  for (int i = 0; i < sizeof(statusPins); ++i)
  {
    if (isOpen(statusPins[i]))
    {
      if (!initialOpen)
        initialOpen = now();

      latestOpen = now();

      if((latestOpen - initialOpen) > NOTIFICATIONS_WATCHDOG_MINUTES * 60)
      {
        #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
          Serial.print(F("*** watchdog notification handler - detected opened device/door @ pin #"));
          Serial.print(statusPins[i]);
          Serial.println(F(" ***"));
        #endif

        if (!notificationSent)
        {
          #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
            Serial.println(F("*** watchdog notification handler - sending notification ***"));
          #endif

          char subject[] = "MyDoorOpener Notification";
          char body[100] = "";
```

```
        sprintf(body, "A door or device has been opened for more than %i minute(s).",
NOTIFICATIONS_WATCHDOG_MINUTES);

        #if defined(PUSH_NOTIFICATIONS)
          notifyViaPush(subject, body);
        #endif

        #if defined(SMS_NOTIFICATIONS)
          notifyViaSms(subject, body);
        #endif

        #if defined(SMTP_NOTIFICATIONS)
          notifyViaEmail(subject, body);
        #endif

        notificationSent = true;
       }
       else
       {
        #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
          Serial.println(F("*** watchdog notification handler - NOT sending notification ***"));
        #endif
       }
      }

     openDetected = true;
     break;
    }
  }

  // if all door/devices are closed, reset notificationSent semaphore and timer so that further notifications
  // can be sent. This is done to avoid floading recipient with notifications for a same event occurrence.

  if (!openDetected)
  {
   notificationSent = false;
   initialOpen = NULL;
   delay(500);
  }
}

#endif

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
#if defined(NOTIFICATIONS_OPEN)

//--------------------------------------------------------------------------------------------------
void openNotificationsHandler()
{
  static boolean notificationSent = false;
  boolean openDetected = false;

  for (int i = 0; i < sizeof(statusPins); ++i)
```

```
  {
    if (isOpen(statusPins[i]))
    {
      #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
        Serial.print(F("*** open notification handler - detected an opened device/door @ pin #"));
        Serial.print(statusPins[i]);
        Serial.println(F(" ***"));
      #endif

      if (!notificationSent)
      {
        #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
          Serial.println(F("*** open notification handler - sending notification ***"));
        #endif

        char subject[] = "MyDoorOpener Notification";
        char body[] = "A door or device has just been opened at home.";

        #if defined(PUSH_NOTIFICATIONS)
          notifyViaPush(subject, body);
        #endif

        #if defined(SMS_NOTIFICATIONS)
          notifyViaSms(subject, body);
        #endif

        #if defined(SMTP_NOTIFICATIONS)
          notifyViaEmail(subject, body);
        #endif

        notificationSent = true;
      }
      else
      {
        #if defined(NOTIFICATIONS_SERIAL_DEBUGGING)
          Serial.println(F("*** open notification handler - NOT sending notification ***"));
        #endif
      }

      openDetected = true;
      break;
    }
  }

  // if all door/devices are closed, reset notificationSent semaphore so that further notifications
  // can be sent. This is done to avoid floading recipient with notifications for a same event occurrence.

  if (!openDetected)
  {
    notificationSent = false;
    delay(500);
  }
}
```

```
#endif

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
void setup()
{
  delay( 200 );   // allow some time (50 ms) after powerup and sketch start, for the Wiznet W5100 Reset IC to release
and come out of reset.

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING) || defined(WEBDUINO_SERIAL_DEBUGGING) ||
defined(SMTP_SERIAL_DEBUGGING) || #defined(PUSH_SERIAL_DEBUGGING) ||
defined(NOTIFICATIONS_SERIAL_DEBUGGING)
    Serial.begin(9600);
  #endif

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.println(F("*** MyDoorOpener setup begin ***"));
  #endif

  for (int i = 0; i < sizeof(statusPins); ++i)
    configureStatusPin(statusPins[i]);

  for (int i = 0; i < sizeof(relayPins); ++i)
  {
    pinMode(relayPins[i], OUTPUT);
    digitalWrite(relayPins[i], LOW);
  }

  // set arbitrary time - used for always-changing challenge token generation

  setTime(0, 0, 0, 1, 1, 2010);

  // start web server

  Ethernet.begin(mac, ip, mydns, gateway );

  webserver.setDefaultCommand(&webRequestHandler);
  webserver.addCommand("", &webRequestHandler);
  webserver.begin();

  #if defined(MYDOOROPENER_SERIAL_DEBUGGING)
    Serial.println(F("*** MyDoorOpener setup completed ***"));
  #endif
}

//--------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------
void loop()
{
  char buffer[200];
  int len = sizeof(buffer);

  webserver.processConnection(buffer, &len);
```

```
  #if defined(NOTIFICATIONS_WATCHDOG_MINUTES)
    watchDogNotificationsHandler();
  #endif

  #if defined(NOTIFICATIONS_OPEN)
    openNotificationsHandler();
  #endif
}
```