Tutorial
BMS Server Studio

UDP

ÍNDICE

Página

0. Introducción

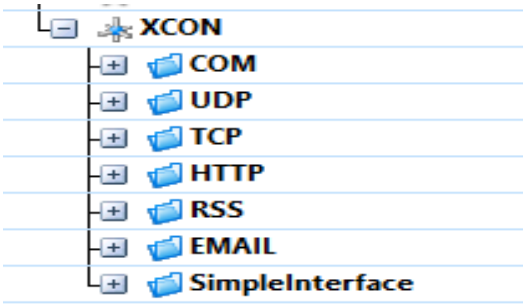
En este documento explicaremos el funcionamiento del XCON del NETx BMS Studio, donde se encuentran los diferentes elementos que se pueden utilizar para la comunicación con otros sistemas o servicios de red (Puerto COM, UDP, TCP, HTTP, correo electrónico). En concreto nos centraremos en el protocolo UDP.

UDP (User Datagram Protocol) es un protocolo del nivel de transporte, basado en el intercambio de datagramas (Capa 4 del modelo OSI).

Este protocolo permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. No contiene datos de confirmación ni control de flujo y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción.

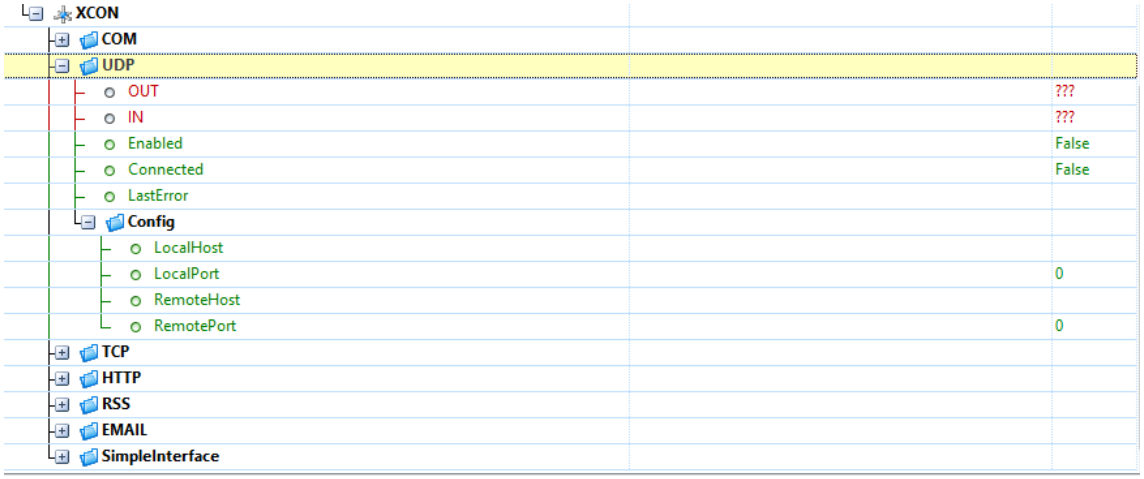
En la familia de protocolos de Internet UDP proporciona una sencilla interfaz entre la capa de red y la capa de aplicación. UDP no otorga garantías para la entrega de sus mensajes (por lo que realmente no se debería encontrar en la capa 4) y el origen UDP no retiene estados de los mensajes UDP que han sido enviados a la red. Cualquier tipo de garantías para la transmisión de la información deben ser implementadas en capas superiores.

Su uso principal es para protocolos como DHCP o DNS, en los que el intercambio de paquetes de conexión/desconexión son de gran tamaño, así como para la transmisión de audio y vídeo en tiempo real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.



1. Configuración del puerto UDP

Para el uso de esta interfaz es necesario configurar una serie de parámetros previamente. A continuación detallaremos la funcionalidad de cada punto y la configuración necesaria para el correcto funcionamiento de la interfaz, debido a que es necesario que sea configurado y habilitado antes de ser usado.



OUT

- Tipo de dato: STRING
- Valor por defecto: Ninguno
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.OUT
- Descripción: El buffer de bytes se envía a la interfaz UDP se escribe en este punto.

IN

- Tipo de dato: STRING
- Valor por defecto: Ninguno
- Derecho de acceso: Sólo lectura
- Ruta estándar: NETx.XCON.UDP.IN
- Descripción: El buffer de bytes que lee de la interfaz UDP se escribe en este punto.

ENABLED

- Tipo de dato: BOOL
- Valor por defecto: 0 (False)
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.Enabled
- Descripción: En este punto se habilita o deshabilita la interfaz UDP. Es responsable de crear la conexión o de que caiga.

CONNECTED

- Tipo de dato: BOOL
- Valor por defecto: 0 (False)
- Derecho de acceso: Sólo lectura
- Ruta estándar: NETx.XCON.UDP.Connected
- Descripción: Muestra si la conexión UDP está abierta o cerrada.

LASTERROR

- Tipo de dato: STRING
- Valor por defecto: Cadena vacía
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.LastError
- Descripción: Si surge algún error, el mensaje se muestra en este punto.

LOCALHOST

- Tipo de dato: STRING
- Valor por defecto: Ninguno
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.Config.LocalHost
- Descripción: Este punto debe contener la dirección IP del dispositivo local.

LOCALPORT

- Tipo de dato: INT4
- Valor por defecto: 0
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.Config.LocalPort
- Descripción: Aquí se debe reflejar el puerto del dispositivo local para que la comunicación pueda ser establecida correctamente.

REMOTEHOST

- Tipo de dato: STRING
- Valor por defecto: Ninguno
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.Config.RemoteHost
- Descripción: Este punto debe de contener la dirección IP del dispositivo remoto para que la comunicación pueda ser establecida correctamente.

REMOTEPORT

- Tipo de dato: INT4
- Valor por defecto: 0
- Derecho de acceso: Lectura y Escritura
- Ruta estándar: NETx.XCON.UDP.Config.RemotePort
- Descripción: Este punto contiene el puerto del dispositivo remoto para que la comunicación pueda ser establecida correctamente.

2. Ejemplos

A continuación configuraremos la interfaz UDP y realizamos una prueba de comunicaciones con Socket Test.

Para realizar la prueba usaremos la dirección IP local.

2.1 Configuración manual

Para las pruebas de comunicación usaremos una herramienta JAVA llamada *SocketTest*. Se puede utilizar para probar cualquier servidor o cliente que utiliza el protocolo TCP o UDP para comunicarse.

2.1.1 Configuración SocketTest

Arrancamos el SocketTest y nos posicionamos en la pestaña UDP. Como vemos en la siguiente figura debemos introducir cuatro parámetros para que la comunicación se produzca:

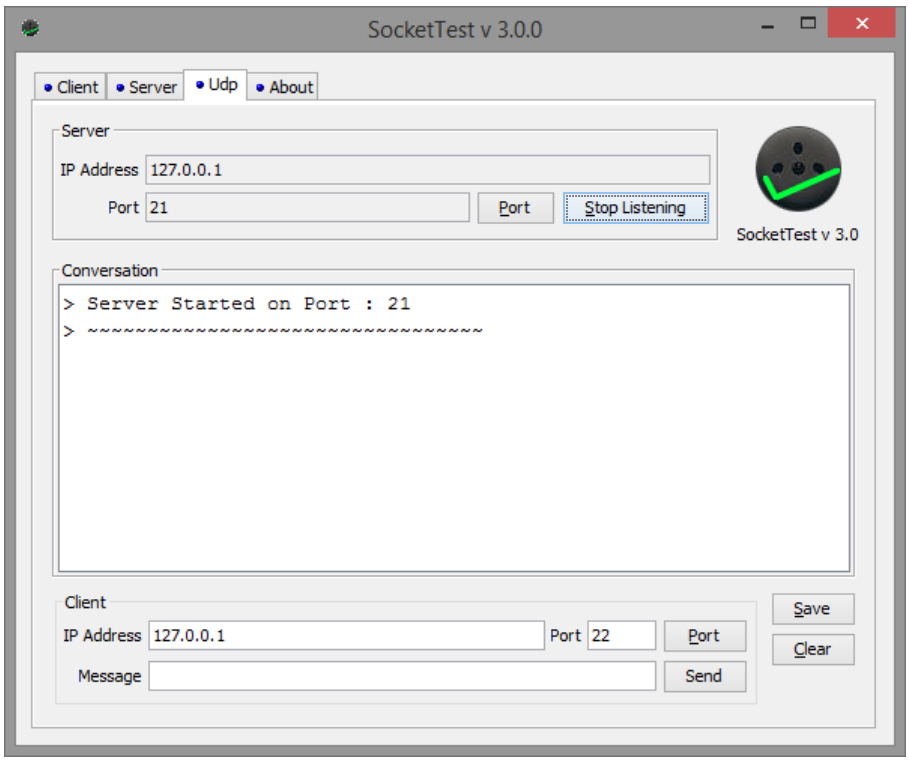
Para el servidor:

- Dirección IP: 127.0.0.1
- Puerto: 21

Para el cliente:

- Dirección IP: 127.0.0.1
- Puerto: 22

A continuación se debe iniciar la comunicación pulsando "Start Listening" y debe aparecer el siguiente mensaje:



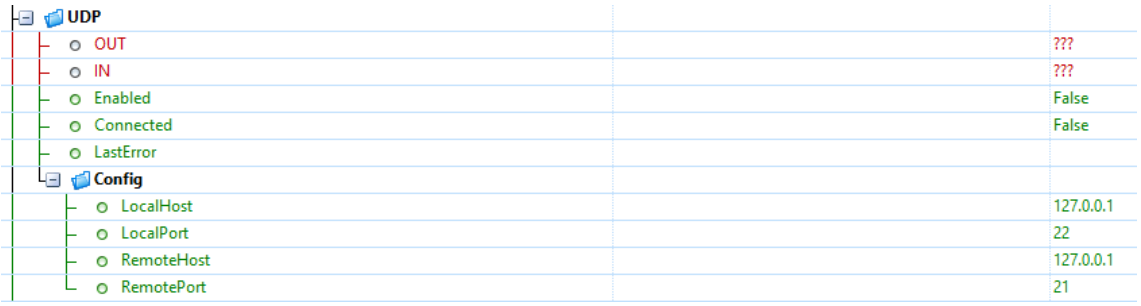
Se puede observar que en nuestro caso la dirección IP del servidor es la misma que la dirección IP del cliente, esto es debido a que estamos trabajando con un servidor local. Si tuviésemos un servidor externo en la dirección IP del servidor deberíamos introducir la dirección IP de dicho servidor.

En cuanto al puerto podemos configurar cualquier puerto que permita el protocolo UDP. UDP utiliza puertos para permitir la comunicación entre aplicaciones. El campo de puerto tiene una longitud de 16 bits, por lo que el rango de valores válidos va de 0 a 65.535. El puerto 0 está reservado, pero es un valor permitido como puerto origen si el proceso emisor no espera recibir mensajes como respuesta.

2.1.2 Configuración BMS Server Studio

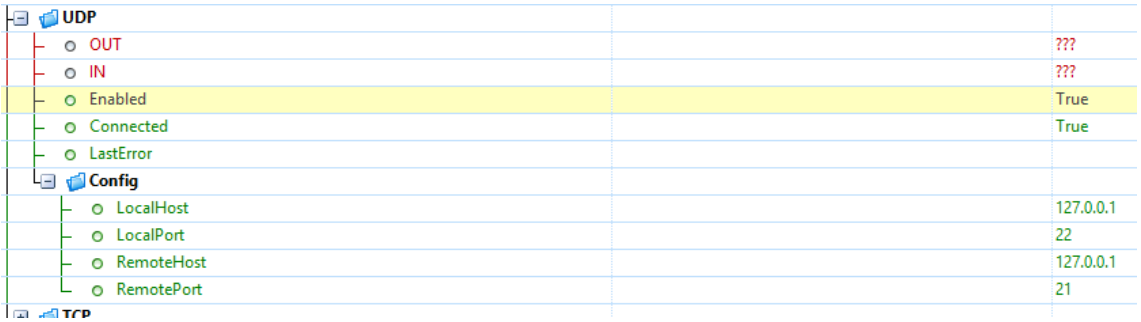
Una vez configurado el SocketTest, vamos a configurar los campos referidos al protocolo UDP del BMS Server Studio, que se encuentran en Item Tree, dentro del XCON.

Como vemos en la figura, se han configurado las direcciones IP de acuerdo a la configuración previa del Socket. Por tanto, para el dispositivo local se ha escogido el puerto 22, mientras que para el dispositivo remoto se ha escogido el 21, siendo las direcciones IP iguales en los dos dispositivos:

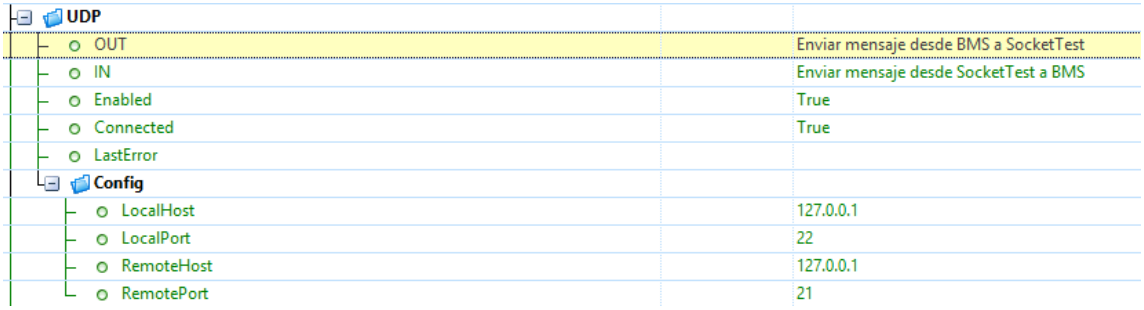


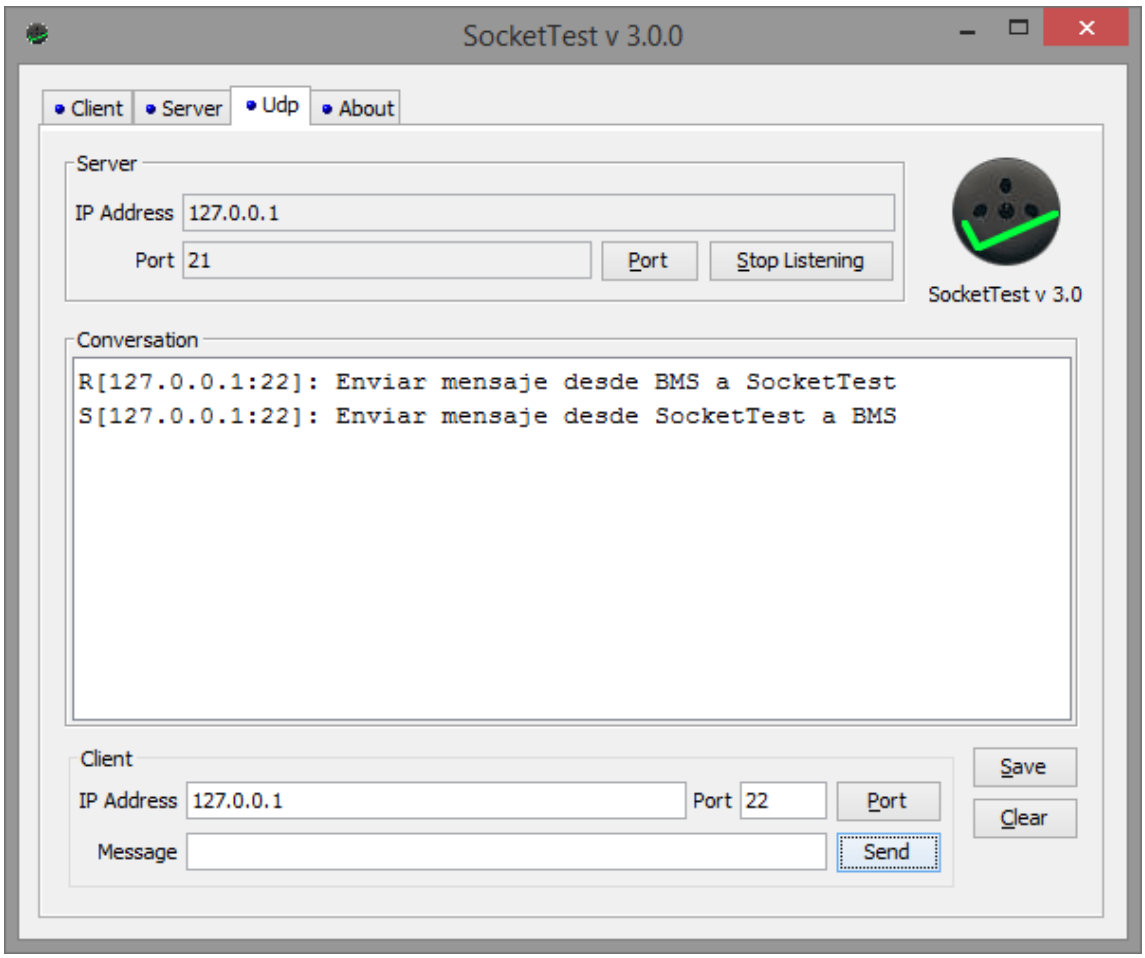
A continuación, para establecer la conexión se debe habilitar, escribiendo un “1” o “true” en la casilla *Enabled.*

Si la configuración se ha hecho correctamente, automáticamente la casilla *Connected* debe ponerse a *True o Verdadero.*



Llegados a este punto, la conexión está establecida, por lo que podemos enviar y recibir mensajes. Es decir, si escribimos un mensaje, en la casilla OUT (Ej: Enviar mensaje desde BMS a SocketTest), debe aparecer en la pantalla del Socket, o por el contrario, si escribimos cualquier cosa en la casilla Message de SocketTest, debe aparecer en la casilla IN (Ej: Enviar mensaje desde SocketTest a BMS) del BMS Server.





2.2 Configuración mediante Script

A continuación realizaremos un script mediante el cual configuraremos el UDP del BMS automáticamente cuando se inicie el BMS Server Studio.

Como ejemplo realizaremos un script que mide la temperatura de un motor contenida en un Holding Registers y envía por UDP el valor de este. Si el valor de esta temperatura es mayor de 60º se envía un mensaje de alarma y se apaga el motor. Además se podrá encender o apagar el motor remotamente desde el SocketTest.

Para ello crearemos un script donde introduciremos las funciones para la configuración y el envío de mensajes por UDP. El nombre del script es nxaUDP.lua. Dentro de este script hemos definido 4 funciones:

- config(): función para configurar el UDP en el BMS.
- iniciar(): se habilita la conexión UDP.
- mensaje(): Envía el valor de la temperatura del motor y si el motor está encendido o apagado.
- onoff(): Lee el valor de entrada del UDP (ON, OFF) y enciende o apaga el motor y además apaga el motor si se supera un umbral de 60º.

El Script creado es el siguiente:

```
function config()
        nxa.SetValue("NETx.XCON.UDP.Config.LocalHost", "127.0.0.1")
        nxa.SetValue("NETx.XCON.UDP.Config.LocalPort", "22")
        nxa.SetValue("NETx.XCON.UDP.Config.RemoteHost", "127.0.0.1")
        nxa.SetValue("NETx.XCON.UDP.Config.RemotePort", "21")
end

function iniciar()
        nxa.WriteValue("NETx.XCON.UDP.Enabled", "1", "1000")
end

function mensaje()
local d=60
if      (d > nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Holding Registers\\0") and
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0")==true)
then
        nxa.SetValue("NETx.XCON.UDP.OUT", "MOTOR ENCENDIDO, TEMPERATURA
        CORRECTA")
        nxa.SetValue("NETx.XCON.UDP.OUT",
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Holding Registers\\0"))
end

if (nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0")==false)
then
        nxa.SetValue("NETx.XCON.UDP.OUT", "MOTOR APAGADO, TEMPERATURA:")
        nxa.SetValue("NETx.XCON.UDP.OUT",
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Holding Registers\\0"))
end
end
```

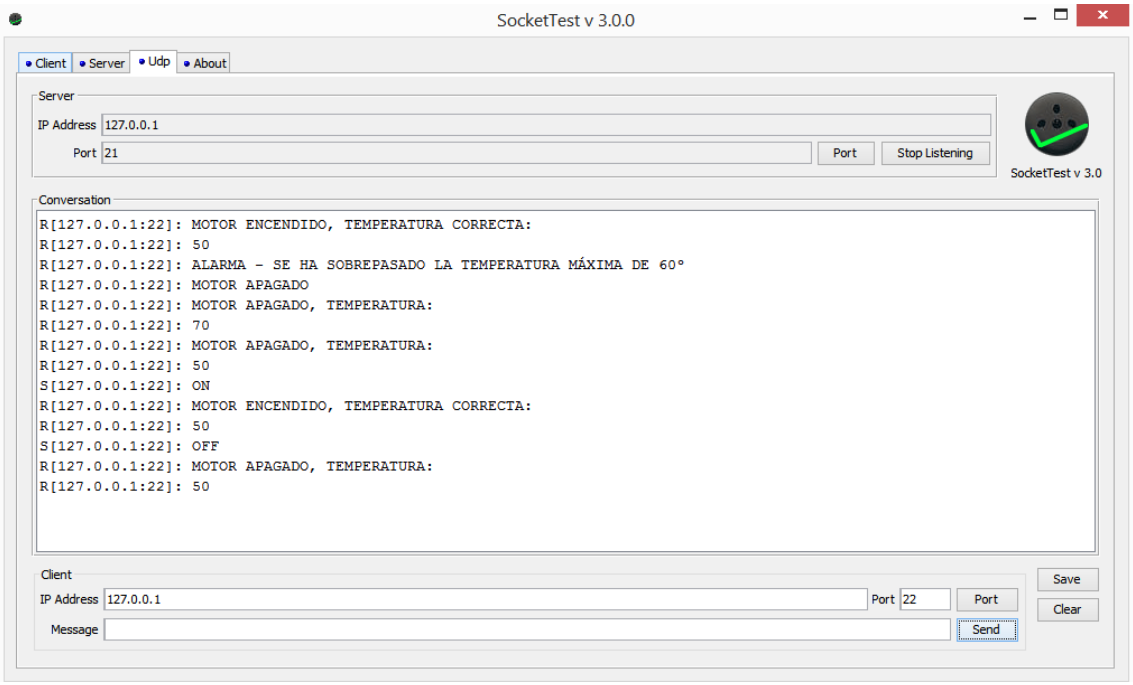
```
function onoff()
local d=60
if      (d < nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Holding Registers\\0") and
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0")==true)
then
        nxa.SetValue("NETx.XCON.UDP.OUT", "ALARMA - SE HA SOBREPASADO LA
        TEMPERATURA MÁXIMA DE 60º")
        nxa.SetValue("NETx.XCON.UDP.OUT", "MOTOR APAGADO")
        nxa.SetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0", "0")
end

if      (nxa.GetValue("NETx.XCON.UDP.IN")=="ON" and
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0")==false)
then
        nxa.SetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0", "1")
end

if      (nxa.GetValue("NETx.XCON.UDP.IN")=="OFF" and
        nxa.GetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0")==true)
then
        nxa.SetValue("NETx\\XIO\\Modbus\\Modbus1\\Coils\\0", "0")
end
end
```

Una vez definido el script se deben introducir las funciones dentro del script nxa.Definitions.lua, en este script se deben definir los siguientes parámetros para que se inicie el script:

```
...
require "nxaUDP"
...
function OnScriptStartEvent() -- Cuando se inicia el BMS se ejecutan estas funciones
        config()
        iniciar()
end
…
function OnSecondTimerEvent() -- La función se ejecuta cada segundo
        onoff()
end
...
function OnMinuteTimerEvent() -- La función se ejecuta cada minuto
        mensaje()
end
...
```

Definido el script e iniciado el BMS Server Studio y configurado el SocketTest, obtenemos los siguientes resultados de configuración:

Según el estado en el que se encuentre el motor y dependiendo de la temperatura recibiremos los siguientes mensajes en SocketTest:



Como podemos observar tenemos tres casos posibles:

1. Motor encendido y temperatura menor de 60º
   MOTOR ENCENDIDO, TEMPERATURA CORRECTA: 50
2. Motor encendido y temperatura mayor de 60º
   ALARMA - SE HA SOBREPASADO LA TEMPERATURA MÁXIMA DE 60º

   MOTOR APAGADO
3. Motor apagado
   MOTOR APAGADO, TEMPERATURA: 70

Con esos posibles casos podemos actuar de dos formas distintas enviando los comandos ON/OFF:

1. ON
   MOTOR ENCENDIDO, TEMPERATURA CORRECTA: 50
2. OFF
   MOTOR APAGADO, TEMPERATURA: 50