NETx BMS Server

Tutorial para el uso de **scripts LUA** en el BMS Server de NETxAutomation

NETx AUTOMATION

Versión 1.0 (Agosto de 2015)

Contenido

AGRADECIMIENTO

Este documento no sería posible sin la inestimable ayuda de mi colega y amigo Francisco Simón, profesor de la Universidad de Sevilla y de quien ya he aprendido más de lo que aún le he podido enseñar. Gracias a Paco y a sus alumnos por su esfuerzo e interés en el uso y divulgación de las soluciones que aporta el BMS Server de NETxAutomation.

INTRODUCCIÓN

Una de las ventajas de utilizar un BMS abierto como el de NETxAutomation es la versatilidad o capacidad de adaptación a cualquier necesidad. En este sentido, la decisión de utilizar un motor de scripts como compañero de viaje del resto de las funcionalidades del BMS Server fue siempre una necesidad más que una opción. El lenguaje elegido fue LUA, hasta ahora algo desconocido, pero hoy día con una implantación exponencial en múltiples sectores.

LUA es un lenguaje de programación imperativo, estructurado y bastante ligero que fue diseñado como un lenguaje interpretado con una semántica extendible. El nombre significa «luna» en portugués y su logo así lo refleja.

http://www.lua.org/

Imagen 1: Logo de LUA

Para profundizar en el uso de LUA y su aplicación práctica, os recomendamos la lectura (la compra es opcional, aunque siempre es bueno apoyar a los autores), de la biblia de LUA.

El presente tutorial pretende detallar las múltiples posibilidades de uso de LUA en el BMS Server de NETxAutomation. En el futuro se irá enriqueciendo con multiples ejemplos de uso, que agradeceremos también nos enviéis los partners.

PRIMEROS PASOS CON LUA

El primer acceso de que disponemos a los scripts de LUA en el BMS Server de NETxAutomation es el botón "Edit Script" del BMS Server Studio.

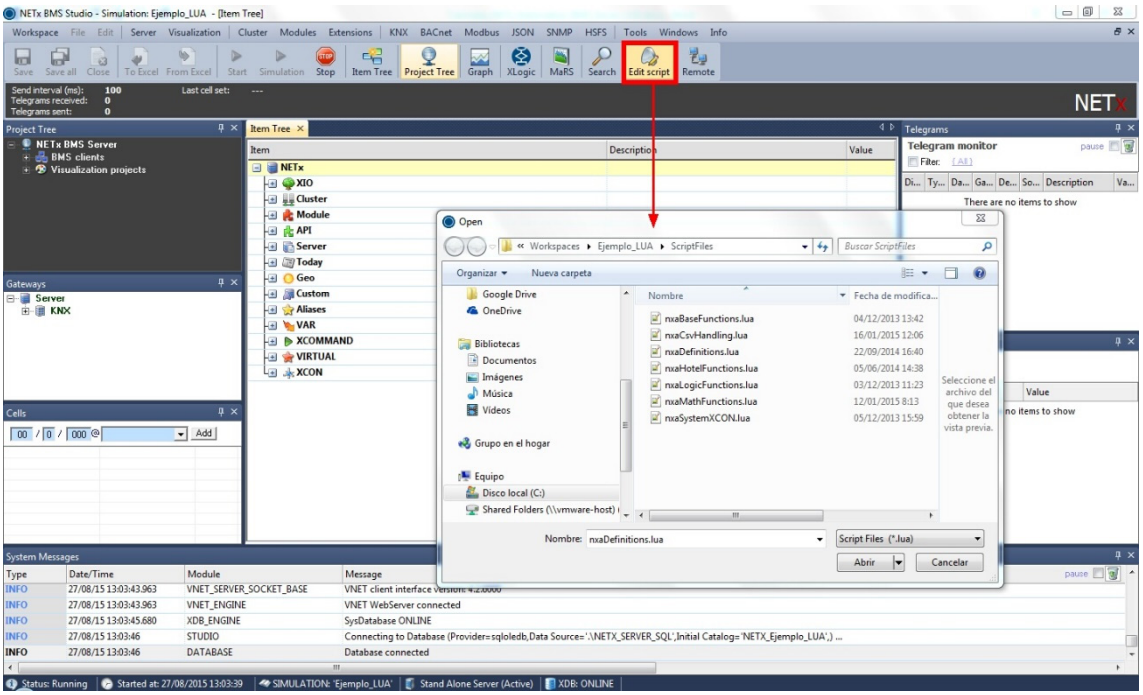


Imagen 2: Acceso a los scripts LUA del BMS Server

Los scripts se pueden crear y editar usando el botón "Edit script" del BMS Server, o desde cualquier otro editor. Por ejemplo, si se instala en el ordenador la distribución de LUA se instala con ella el editor SciTE (si quieres echarle un vistazo, busca el paquete gratuito "lua for windows").

Cuando se crea el script desde el BMS, este se guarda en el directorio ScriptFiles del espacio de trabajo creado por el BMS.

Podemos vincular un script a un evento de por ejemplo una tarea o un punto virtual y también podemos ejecutar el script directamente desde el menú Tools → Execute Script… del BMS.

Por defecto el BMS ejecuta el Script nxaDefinitions.lua residente en el directorio ScriptFiles.

Cualquier otro script que se quiera ejecutar debe estar instanciado en el script principal nxaDefinitions.lua. Para ello debemos incluirlo usando el comando "require" de LUA.

Por ejemplo, la sentencia: *require "nxaExample"* dentro del fichero nxaDefinitions, hace visible al BMS las funciones creadas dentro del fichero de scripts " nxaExample.lua". En concreto este fichero contiene la función HelloWorld().

Pero también, si abrimos en el BMS el menú "Tools → Execute Script", escribimos HelloWorld() y pulsamos el botón Execute, veremos en los mensajes del sistema el texto "hello world!".

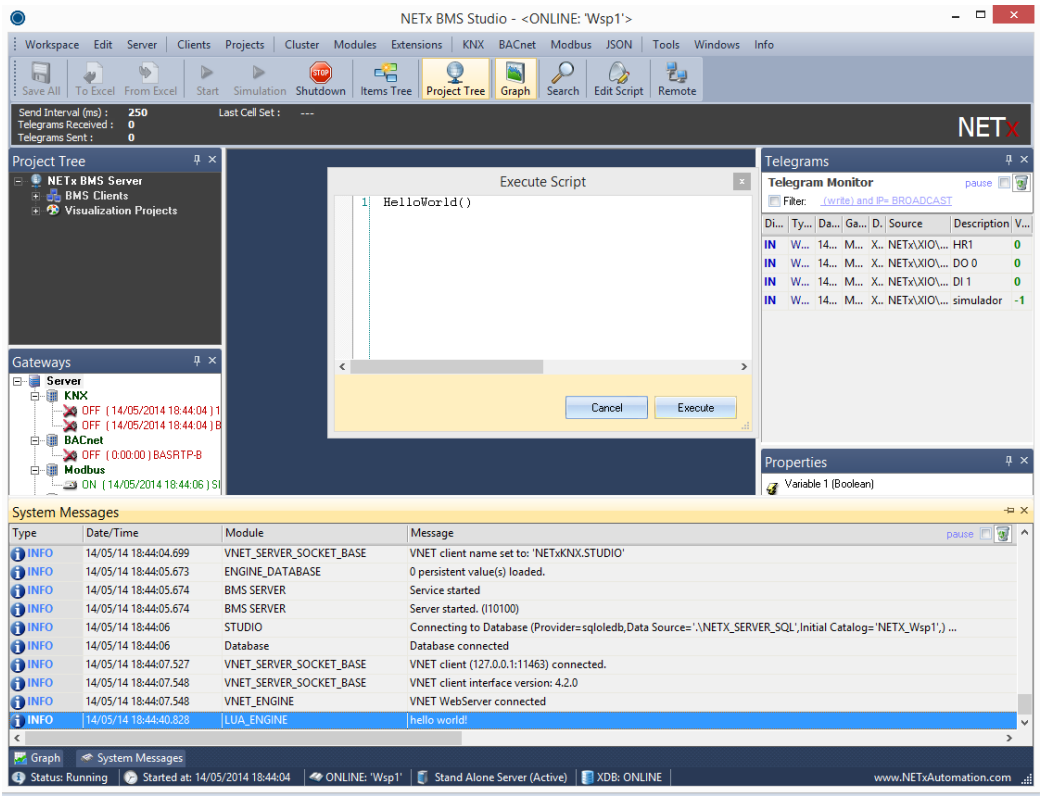


Imagen 3: Ejecución directa de un script desde el menú "Tools"

NOTA: Este procedimiento de vinculación de scripts en el BMS obliga a que la unidad mínima de ejecución de un script sea una función y además a que NO ESTÉ REPETIDO EL NOMBRE DE LA FUNCIÓN EN NINGÚN SCRIPT REFERENCIADO.

ALTERNATIVAS DE EJECUCIÓN DE LOS SCRIPTS

En muchas ocasiones el script es solo una secuencia de órdenes correlativas que no necesitan estar incluidas en una función. Otras veces tenemos el problema de reutilizar un script creado por otro usuario que colisiona con los nombres de las funciones que ya tenemos creadas.

Para resolver el problema lo que podemos hacer es no referenciar el script en el fichero nxaDefinitions y ejecutar nuestro script de manera independiente. Pongamos por ejemplo el script anterior nxaExample.lua El código del script es el siguiente:

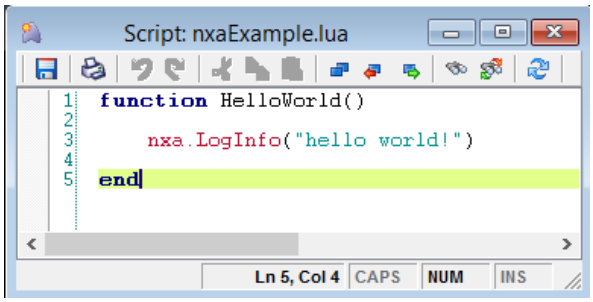


Imagen 5

Como vemos, solo hay una línea de código con la sentencia nxa.LogInfo encargada de enviar un texto al sistema de mensajes del BMS.
Una alternativa podría ser crear un Script solo con la sentencia nxa.LogInfo.
Lo creamos con el nombre nxaEx.lua y lo guardamos en el directorio c:\scripts

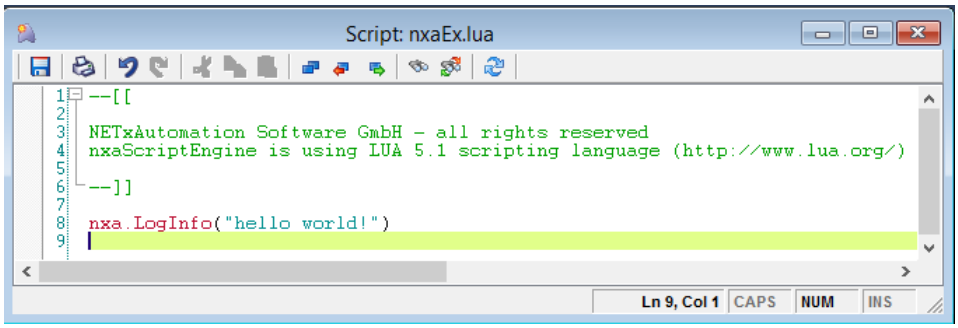


Imagen 6

Para ejecutar el script usaremos la **sentencia dofile** de LUA.

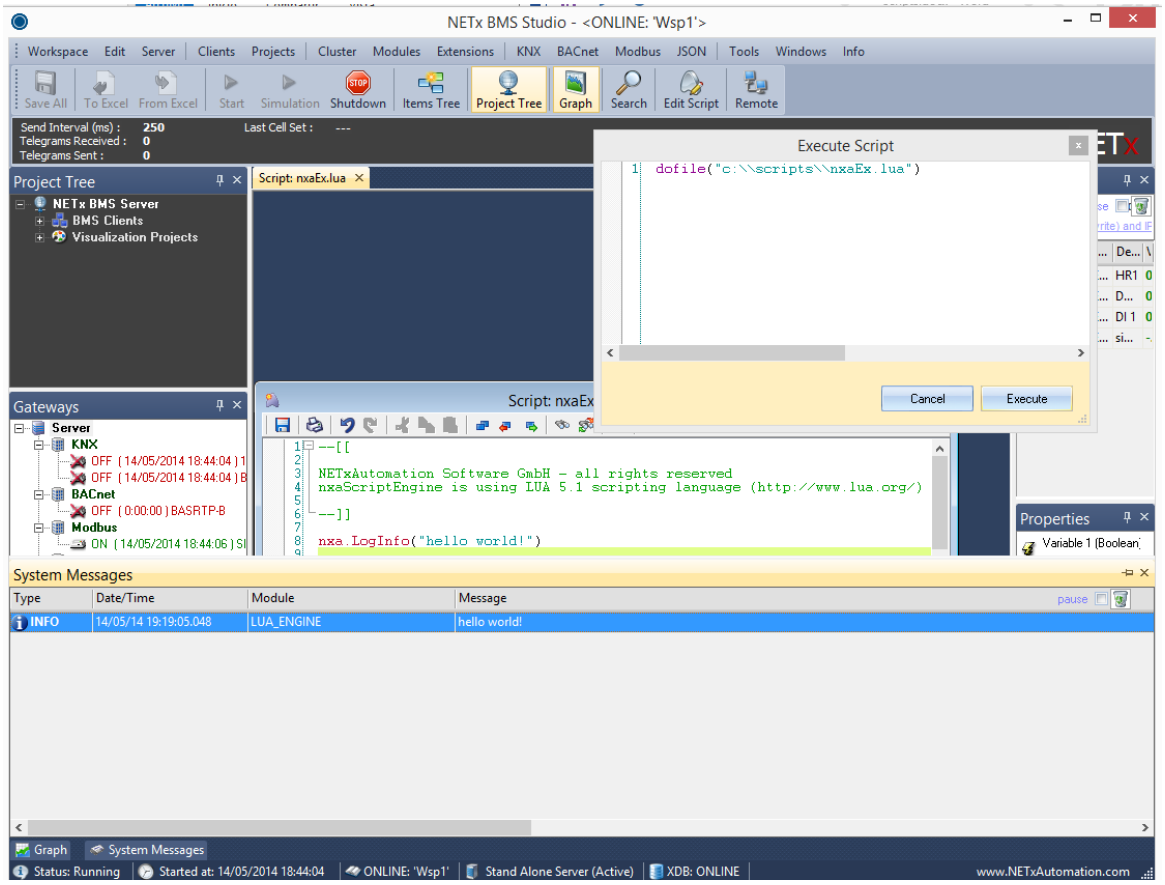


Imagen 7: Ejecución de un script mediante la sentencia dofile de LUA.

En este caso, es necesario especificar al script la ruta al archivo *(path)* como una cadena de caracteres sustituyendo el carácter '\' por "\\" y terminando el nombre del script con la extensión del fichero ".lua". Como vemos el script se ejecuta correctamente.

Si el script estuviera el directorio ScriptFiles y dado que el path es muy complicado de escribir, usaríamos la función **nxa.ScriptFilesPath()** del BMS para componer la cadena del nombre del script.

En este caso el comando a ejecutar sería:

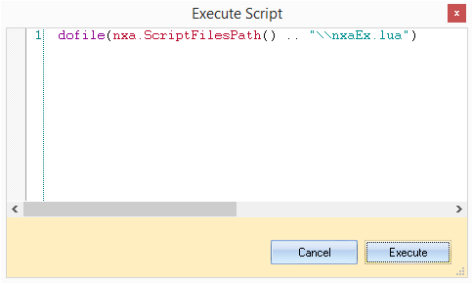


Imagen 8: ScriptFilesPath

NOTA: los dos puntos entre ambos campos es el operando de concatenación en LUA.

01 LUA CALLBACKS

Con el motor de scripts LUA que proporciona el BMS Server de NETxAutomation se pueden realizar múltiples funciones como crear ítems, inicializarlos, eliminarlos, crar enlaces entre los mismos,…

No obstante, es importante entender que, por ejemplo, la creación de un ítem ha de hacerse en un momento determinado del inicio del sistema, ya que no ha de haberse creado aún el árbol de ítems para poder añadirlo al mismo. Asimismo, si queremos inicializar un ítem con un valor dfeterminado, esto solo podremos hacerlo una vez que el árbol de ítems ya haya sido creado.

Este flujo de eventos (Inicio, arranque, finalización,…), viene definido por los llamados callbacks (por ejemplo, *OnInitEvent*).

Por tanto, como complemento a las múltiples funciones de LUA, la librería XNA LUA también provee de múltiples eventos de callback o auto-cargado en español, los cuales son invocados por el servidor en momentos específicos. Con estos callbacks, el usuario puede integrar Scripts de LUA que son ejecutados cuando el servidor invoca al callback correspondiente. Usando este modo, se puede implementar una funcionalidad de control que deberá estar disponible en momentos concretos de tiempo.

ONINITEVENT

Este callback es invocado o llamado durante la inicialización del servidor. En este callback podemos ejecutar tareas de inicialización, por ejemplo:

```
function OnInitEvent()
-- Aquí inicializamos una serie de ítems
-- Cambiamos el estado de una variable a true (podría ser un led que indica el estado del servidor)
        nxa.SetValue("NETx\\VAR\\Boolean\\Ítem1", true)
        nxa.WriteValue("NETx\\XIO\\KNX\\192.168.1.2\\03/0/001", true)
-- Añadimos funciones lógicas del módulo nxaLogicFunctions aquí
```

```
        ADD_OR ("NETx\\VAR\\Boolean\\Ítem15", "NETx\\VAR\\Boolean\\Ítem11",
        "NETx\\VAR\\Boolean\\Ítem12", "NETx\\VAR\\Boolean\\Ítem13",
        "NETx\\VAR\\Boolean\\Ítem14")
end
```

ONSTARTEVENT

Este callback es llamado durante la puesta en marcha del servidor. Una vez se haya ejecutado "OnStartEvent" y posteriormente también se haya ejecutado el evento "OnInitEvent", todos los elementos ya estarán disponibles y se pueden acceder a través de scripts LUA , por ejemplo:

```
function OnStartEvent()
  -- Cambiamos el estado de una variable a true (podría ser un led que indica el estado del servidor)
nxa.SetValue("NETx\\VAR\\Boolean\\Ítem1", true)
end
```

ONSTOPEVENT

Este callback es llamado durante el proceso de shutdown del servidor, por ejemplo:

```
function OnStopEvent()
  -- Cambiamos el estado de una variable a false (podría ser un led que indica el estado del servidor)
nxa.SetValue("NETx\\VAR\\Boolean\\Ítem1", false)
end
```

ONSECONDTIMEREVENT

Este callback es llamado cada segundo. Una de sus posibles utilidades es que puede ser usado para implementar una funcionalidad que tenga que ser ejecutada cada segundo, por ejemplo:

```
function OnSecondTimerEvent()
```

```
 -- Comprobamos cada segundo que la bomba (Ítem2) está funcionando, ya que de apagarse podría tener consecuencias nefastas
If nxa.GetValue("NETx\\VAR\\Boolean\\Ítem2", false)
        Then print ("Error: la bomba está apagada")
        end
end
```

ONMINUTETIMEREVENT

Este callback es llamado cada minuto. Una de sus posibles utilidades es que puede ser usado para implementar una funcionalidad que tenga que ser ejecutada cada minuto, por ejemplo:

```
function OnMinuteTimerEvent()
 -- Comprobamos cada minuto que la bomba (Ítem2) está funcionando, ya que de apagarse podría tener consecuencias nefastas
If nxa.GetValue("NETx\\VAR\\Boolean\\Ítem2", false)
        Then print ("Error: la bomba está apagada")
        end
end
```

ONHOURTIMEREVENT

Este callback es llamado cada hora. Una de sus posibles utilidades es que puede ser usado para implementar una funcionalidad que tenga que ser ejecutada cada hora, por ejemplo:

```
function OnHourTimerEvent()
 -- Comprobamos cada hora que la bomba (Ítem2) está funcionando, ya que de apagarse podría tener consecuencias nefastas
If nxa.GetValue("NETx\\VAR\\Boolean\\Ítem2", false)
        Then print ("Error: la bomba está apagada")
        end
end
```

ONKNXGATEWAYCONNECTEDEVENT

Este callback es llamado cada vez que una entrada de KNX ha sido conectada. El parámetro que se le pasa a la función puede ser usado para determinar que la acción se ejecute para una entrada determinada, por ejemplo:

Parámetros:

- string – Dirección IP de la entrada KNX.

Ejemplo:

```
function OnKNXGatewayConnectedEvent(ipaddress)
  -- Indicamos que ha sido conectado un dispositivo con IP = ipaddress
        print("Ha sido conectado un dispositivo en el sistema con IP = ".. ipaddress)
  -- Si conocemos el dispositivo, decimos cual es, por ejemplo climatización aula 2.11
        If ipaddress == variableLocalConIP
        Then print ("El dispositivo X ha sido conectado")
        end
end
```

ONKNXGATEWAYDISCONNECTEDEVENT

Este callback es llamado cada vez que una entrada de KNX ha sido desconectada. El parámetro que se le pasa a la función puede ser usado para determinar que la acción se ejecute para una entrada determinada, por ejemplo:

Parámetros:

- string – Dirección IP de la entrada KNX.

```
function OnKNXGatewayDisconnectedEvent(ipaddress)
  -- Indicamos que ha sido desconectado un dispositivo con IP = ipaddress
print ("Ha sido desconectado un dispositivo en el sistema con IP = ".. ipaddress)
  -- Si conocemos el dispositivo, decimos cual es, por ejemplo climatización aula 2.11
        If ipaddress == variableLocalConIP
        Then print("El dispositivo X ha sido desconectado")
        end
end
```

ONCLIENTCONNECTEDEVENT

Esta función es invocada cuando un cliente es conectado al Servidor BMS de NETx, por ejemplo:

Parámetros:

- clientType – Tipo de cliente (WEB, VNET, or OPC)
- clientName – Nombre de cliente
- source – De donde viene el cliente (e.g. Voyager.5.0, OPC Client name or IP address of BMS Client)
- IPAddress – Dirección IP del cliente
- user – Usuario que está actualmente online.

```
function OnClientConnectedEvent(clientType, clientName, source, IPAddress, user)
  -- Le damos la bienvenida al usuario que se ha conectado
        print ("Bienvenido "..clientName)
end
```

ONCLIENTDISCONNECTEDEVENT

Esta función es invocada cuando un cliente es desconectado del Servidor BMS de NETx, por ejemplo:

Parámetros:

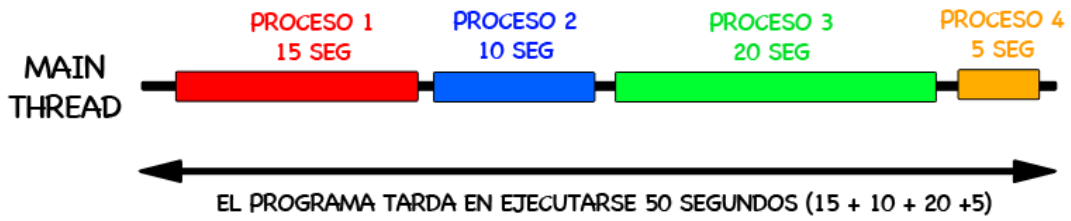
- clientType – Tipo de cliente (WEB, VNET, or OPC)
- clientName – Nombre de cliente
- source – De donde viene el cliente (e.g. Voyager.5.0, OPC Client name or IP address of BMS Client)
- IPAddress – Dirección IP del cliente
- user – Usuario que está actualmente online.

```
function OnClientDisconnectedEvent(clientType, clientName, source, IPAddress, user)
  -- Nos despedimos del usuario que se va a desconectar
        Print ("Hasta pronto "..clientName)
end
```

02 LUA APPS

Todas las funciones LUA son invocadas por el servidor de tareas, los eventos predefinidos (callbacks), las funciones propias del sistema así como las funciones creadas en los scripts LUA creados por los usuarios, son ejecutadas en el mismo contexto.

Esto quiere decir que las funciones se ejecutan en el mismo hilo, lo que provoca que se ejecuten de forma secuencial.



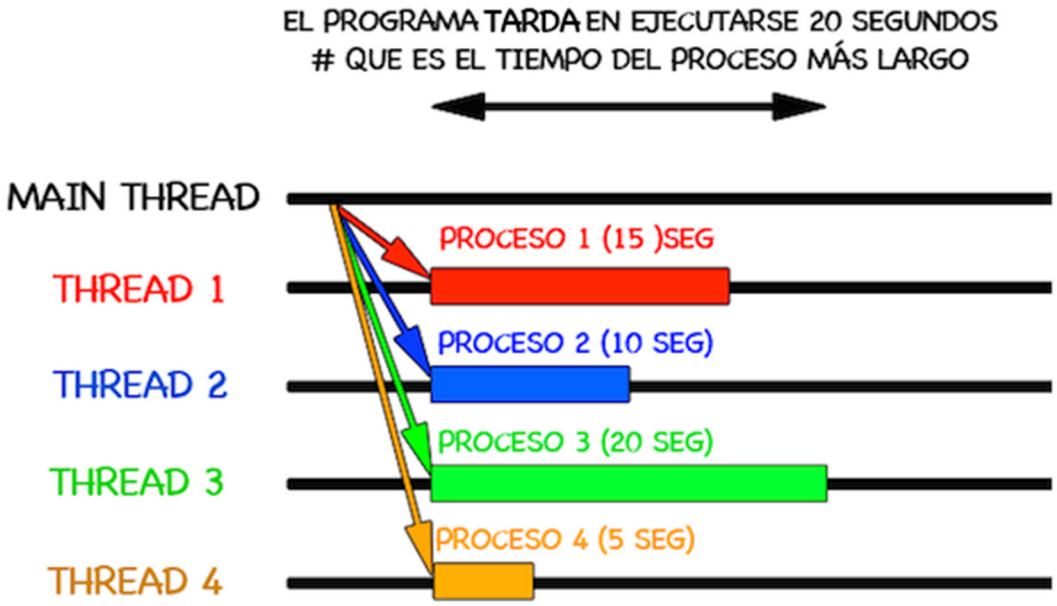
Esta forma de ejecución tiene el problema que para la realización de una tarea anteriormente tiene que haber terminado todas las anteriores. Esto en algunos momentos puede que no nos convenga.

Podemos querer realizar la ejecución de varias funciones en paralelo, para ello hacemos uso de la función de nxa startApp. Por ejemplo podríamos realizar en un hilo una tarea periódica sin que interfiriera en la ejecución de cualquier otra tarea.

Esta función tiene como entrada dos parámetros:

- Archivo LUA: Se le introduce el nombre del archivo LUA (.lua) donde esté definida la función que queremos ejecutar en un hilo independiente.
- Función LUA: Se le introduce la función que quiere ser invocada en un hilo independiente.

Cuando esta función es llamada se encarga automáticamente de crear un hilo independiente del resto de la ejecución, en dicho hilo ejecuta la función que le hayamos metido por parámetros. Este tipo de ejecución tiene la ventaja de no bloquear cualquier otra tarea, si no que ambas se ejecutan en paralelo.



Esta ejecución en paralelo se realiza en medida de lo posible. Hay que tener en cuenta que en cada paso no se realiza solo la ejecución de una tarea, si no que las tareas se van ejecutando en bloques dependiendo de la capacidad del dispositivo así como del tamaño de las funciones a tratar. Una vez terminada la función del hilo este se cierra automáticamente sin necesidad de "matar" el hilo por parte del usuario.

A continuación se mostrará un ejemplo de uso de esta función:
Creamos dos funciones. En **uno()** realizamos una espera activa la cual no hace nada, tan solo es para que espere.
La función **dos()** simplemente imprime por pantalla un mensaje.

```
function uno()
        nxa.LogInfo("funcion 1")
        for i=0, 5000000,1 do
        end
        nxa.LogInfo("funcion 1.1")
end

function dos()
        nxa.LogInfo("funcion 2")
end
```

Una vez creadas dichas funciones pasamos a ejecutar ambas en hilos independientes, para ello escribimos el siguiente código en la herramienta "Execute LUA script":

```
nxa.StartApp(uno())
nxa.StartApp(dos())
```

Primero creará un hilo con la función **uno()** y empezará a ejecutarse, al entrar en la espera activa (la cual no hace nada) y ser un hilo se ejecutara también la función **dos()** antes de la finalización de esta.
El resultado sería el siguiente:
"funcion 1"
"funcion 2"
"funcion 1.1"

En lugar de este sencillo ejemplo podemos utilizar estos hilos para realizar una espera activa a algún tipo de variable como puede ser un sensor de luminosidad. Uno de los hilos está continuamente comprobando el nivel de luz que nos proporciona el sensor, cuando el nivel de luz sea lo suficientemente bajo procederemos a ejecutar cualquier acción (encender una luz artificial). Al estar en otro hilo, la espera activa no interfiere en el comportamiento del resto del sistema, pudiendo realizar cualquier otra tarea que sea necesaria.

Dentro de estos hilos hay que tener cuidado con el tipo de funciones que se utilizan. Se debe tener en cuenta que algunas funciones son críticas por lo que **no pueden realizarse dentro de estos hilos**. Estas funciones son:

```
nxa.LogInfo
nxa.LogWarning
nxa.LogError
nxa.IsInitialized
nxa.IsRunning
nxa.IsSimulation
nxa.IsActiveServer
nxa.IsMainServer
nxa.IsBackupServer
nxa.GetValue | nxa.Value
nxa.SetValue
nxa.SetItemData
```

nxa.WriteValue
nxa.ReadValue
nxa.IsValidValue
nxa.WorkspaceName
nxa.RootPath
nxa.WorkspacePath
nxa.ScriptFilesPath
nxa.DataFilesPath
nxa.LogFilesPath
nxa.ProjectFilesPath
nxa.EventFilesPath
nxa.ConfigFilesPath
nxa.GetItemID
nxa.Sleep
nxa.Now
nxa.TimeOnly
nxa.DateOnly
nxa.Year
nxa.Month
nxa.Day
nxa.Hour
nxa.Minute
nxa.Second
nxa.DayOfWeek
nxa.MakeDate
nxa.MakeTimeOnly
nxa.AddDate
nxa.DiffDate
nxa.DateToString
nxa.SetDate
nxa.SetTimeOnly
nxa.GetVar
nxa.SetVar
nxa.ClearVar
nxa.AddVarTask
nxa.SourceVar
nxa.GetPropertyValue | nxa.PropertyValue | nxa.Property
nxa.SetPropertyValue | nxa.SetProperty

nxa.GetLastErrorText

nxa.GetLastErrorCode

nxa.LowByte

nxa.HiByte

nxa.LowWord

nxa.HiWord

nxa.IsBitSet

nxa.SetBit

nxa.ResetBit

nxa.SetLowByte

nxa.SetHiByte

nxa.SetLowWord

nxa.SetHiWord

nxa.And

nxa.Or

nxa.Xor

nxa.Tor

nxa.Not

03 FUNCIONES BÁSICAS DE LUA EN EL BMS SERVER

Las Funciones estándar **nxa** proporcionan las funcionalidades básicas que cualquier función LUA puede utilizar en cualquier momento durante la ejecución del BMS Server de NETxAutomation.

En general, estas funciones suelen cambiar valores de ítems en el servidor o que, al ser invocadas, recuperan el estado actual del workspace.

Las funciones estándar nxa son:

NXA.ISINITIALIZED()

Determina el estado del servidor, indicando si está encendido o no.
Parámetros: (No aplicable)
Devuelve: boolean es true si está inicializado o false si no.
Ej: nxa.LogInfo(nxa.IsInitialized())

NXA.ISRUNNING()

Indica si el servidor está funcionando
Parámetros: (No aplicable)
Devuelve: boolean a true si está funcionando o a false si no.
Ej: nxa.LogInfo(nxa.IsRunning())

NXA.ISSIMULATION()

Indica si el servidor está en modo simulación o no.
Parámetros: (No aplicable)
Devuelve: boolean a true si se cumple, o a false si no.
Ej: nxa.LogInfo(nxa.IsSimulation())

NXA.ISACTIVESERVER()

Determina si el servidor está activo.

Parámetros: (No aplicable)

Devuelve: boolean a true si está activo, o a false si no.

Ej: nxa.LogInfo(nxa.IsActiveServer())

NXA.ISMAINSERVER()

Determina si el servidor está definido como servidor principal.

Parámetros: (No aplicable)

Devuelve: boolean a true si se cumple, o a false si no.

Ej: nxa.LogInfo(nxa.IsMainServer())

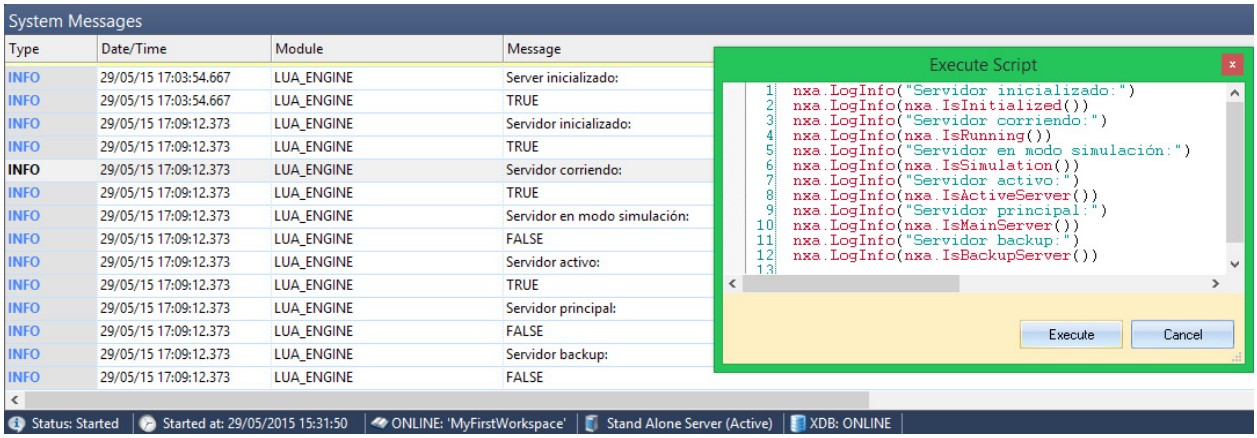
NXA.ISBACKUPSERVER()

Sirve para determinar si el servidor está definido como servidor backup o no.

Parámetros: (No aplicable)

Devuelve: boolean a true si es cierto, o a false si no.

Ej:nxa.LogInfo(nxa.IsBackupServer())



Se puede observar el uso de la función LogInfo, explicada más adelante

NXA.WORKSPACENAME()

Esta función devuelve el nombre actual del workspace.

Parámetros: (No aplicable)

Devuelve: string con el nombre del workspace.

Ej: nxa.LogInfo(nxa.WorkspaceName())

NXA.ROOTPATH()

La función devuelve la ruta donde se ha instalado el server.

Parámetros: (No aplicable)

Devuelve: string con la ruta.

Ej: nxa.LogInfo(nxa.RootPath())

NXA.WORKSPACEPATH()

La función devuelve la ruta donde se encuentra el workspace.

Parámetros: (No aplicable)

Devuelve: string con la ruta.

Ej: nxa.LogInfo(nxa.WorkspacePath())

NXA.SCRIPTFILESPATH()

Esta función retorna la ruta donde encuentra el directorio del script en el workspace actual.

Parámetros: (No aplicable)

Devuelve: string con la ruta.

Ej: nxa.LogInfo(nxa.ScriptFilesPath())

NXA.CONFIGFILESPATH()

Esta función devuelve la ruta al fichero de configuración del actual workspace.
Parámetros: (No aplicable)
Devuelve: string con la ruta.
Ej: nxa.LogInfo(nxa.ConfigFilesPath())

NXA.DATAFILESPATH()

La función devuelve la ruta al directorio del fichero de datos del actual workspace.
Parámetros: (No aplicable)
Devuelve: string con la ruta.
Ej: nxa.LogInfo(nxa.DataFilesPath())

NXA.LOGFILESPATH()

La función devuelve la ruta del directorio de historial del actual workspace.
Parámetros: (No aplicable)
Devuelve: string con la ruta.
Ej: nxa.LogInfo(nxa.LogFilespath())

NXA.PROJECTFILESPATH()

Devuelve la ruta al directorio del proyecto del actual workspace que contenga los archivos de proyecto Smart Voyager (*.vxf).
Parámetros: (No aplicable)
Devuelve: string con la ruta.
Ej: nxa.LogInfo(nxa.ProjectFilesPath())

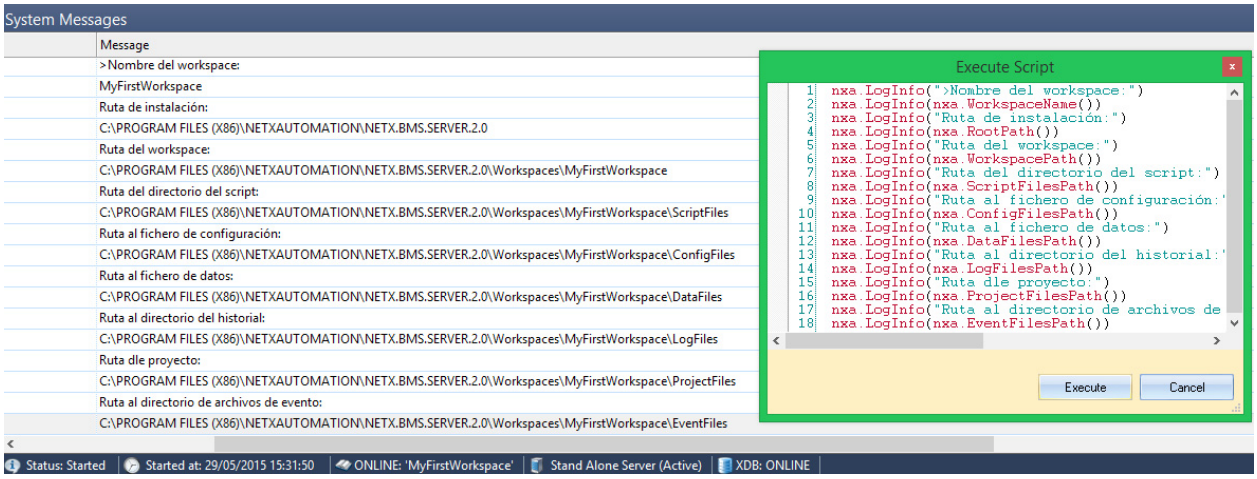
NXA.EVENTFILESPATH()

La función devuelve la ruta al directorio del archivo de evento del actual workspace.

Parámetros: (No aplicable)

Devuelve: string con la ruta.

Ej: nxa.LogInfo(nxa.EventFilesPath())



NXA.LOGINFO()

La función genera un mensaje de información almacenado en el historial del sistema.

Parámetros: string con el mensaje.

Devuelve: (No aplicable)

Ej: nxa.LogInfo("Mensaje de prueba")

NXA.LOGWARNING()

Genera un mensaje de warning que es almacenado en el historial del sistema.

Parámetros: string con el mensaje.

Devuelve: (No aplicable)

Ej: nxa.LogWarning("Mensaje de warning")

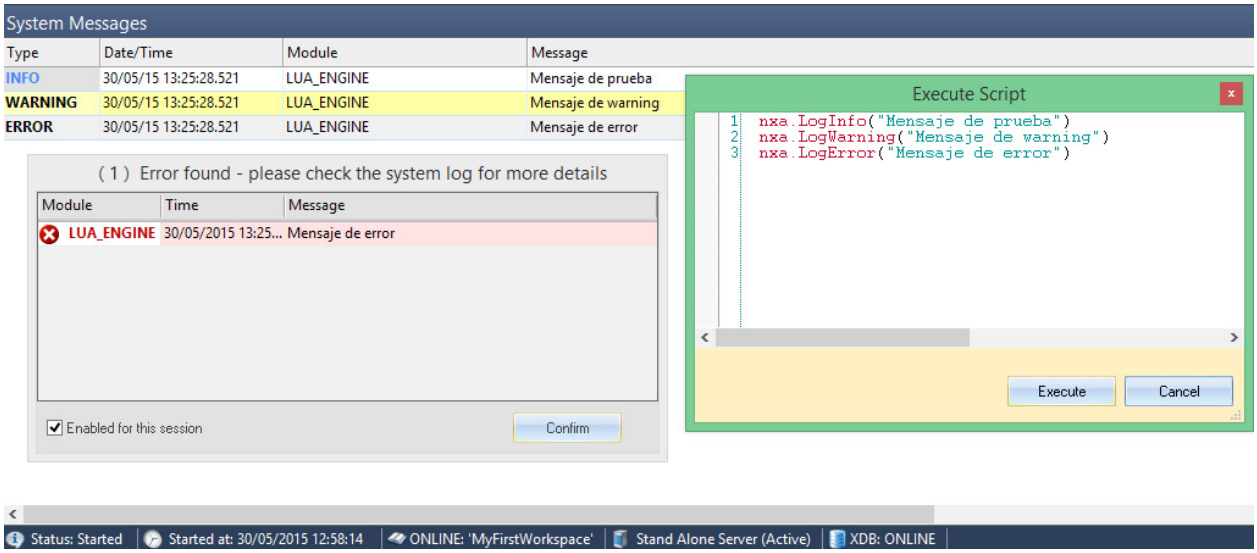
NXA.LOGERROR()

Esta función genera un mensaje de error que es almacenado en el historial del sistema.

Parámetros: string con el mensaje.

Devuelve: (No aplicable)

Ej: nxa.LogError("Mensaje de error")



NXA.SPINTELEGRAMLOG()

Esta función fuerza a almacenar el historial de mensajes actual y crea uno nuevo.

Parámetros: (No aplicable)

Devuelve: boolean que indica el éxito o fracaso de la operación.

Ej: if nxa.SpinTelegramLog() then

nxa.LogInfo("Exito")

end

NXA.SPINSYSTEMLOG()

Esta función fuerza a archivar el actual documento de historial y crea uno nuevo.

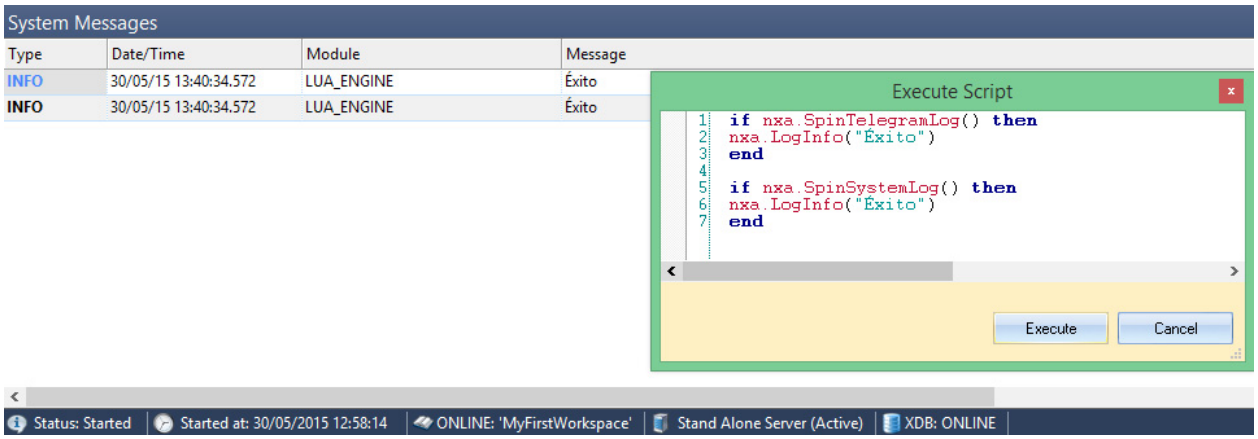
Parámetros: (No aplicable)

Devuelve: boolean que indica el éxito o fracaso de la operación.

Ej: if nxa.SpinTelegramLog() then

nxa.LogInfo("Exito")

end



NXA.GETLASTERRORTEXT()

La función devuelve el último texto de error como string.

Parámetros: (No aplicable)

Devuelve: string con el error.

Ej: nxa.LogInfo(nxa.GetLastErrorText())

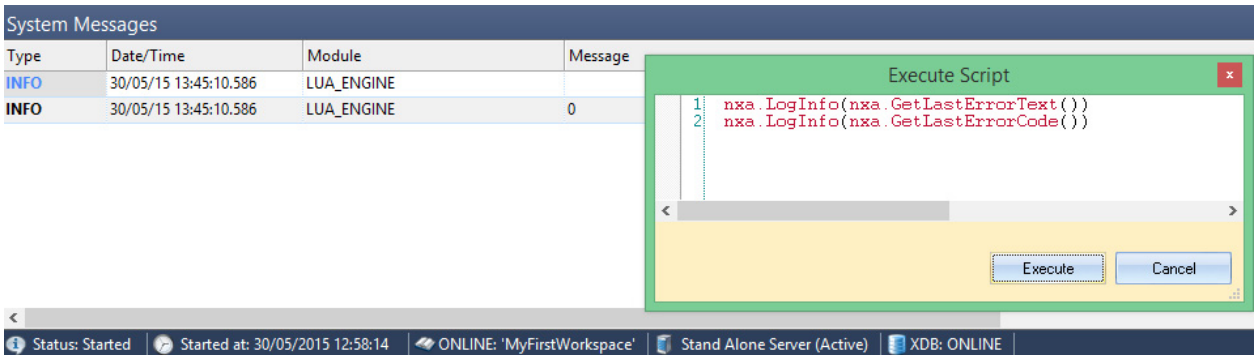
NXA.GETLASTERRORCODE()

La función devuelve el código del último error como integer.

Parámetros: (No aplicable)

Devuelve: string con el error.

Ej: nxa.LogInfo(nxa.GetLastErrorCode())



--Debido a la falta de errores, el primer método devuelve un string vacío y el segundo un 0--

NXA.GETVALUE() | NXA.VALUE()

Esta función solicita el valor actual de un Server Ítem. Si el estado del ítem es 'UNCERTAIN' (incierto), devolverá el valor por defecto.
Hay que destacar que la función lee el valor actual del ítem almacenado en el servidor (no enviará una petición de lectura para el campo del dispositivo requerido).

Parámetros: string con el ítemID del Server Ítem.
tipo con valor por defecto.

Devuelve: tipo con el actual valor del ítem.

Ej: nxa.LogInfo(nxa.GetValue("NETx\\VAR\\Boolean\\Ítem1",0))

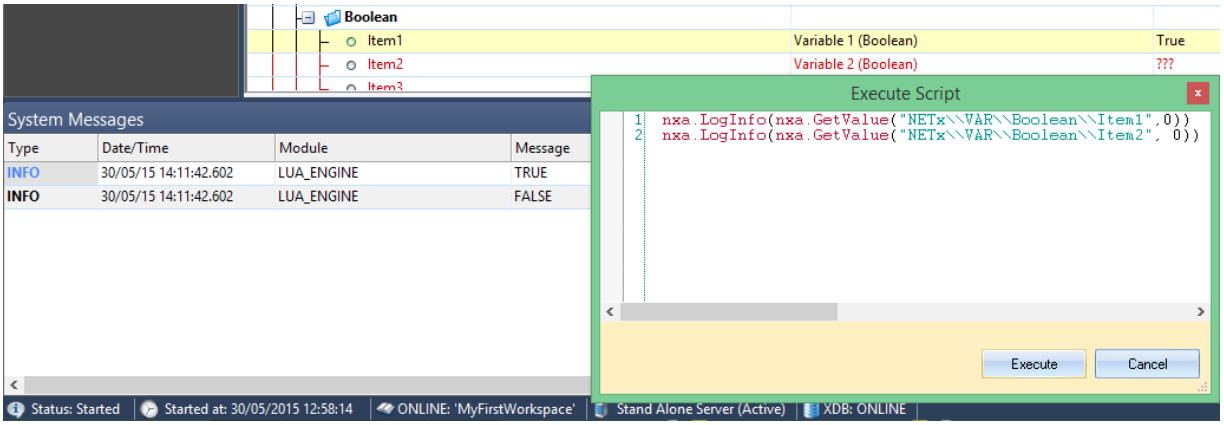
NXA.SETVALUE()

La función actualiza el valor del Server Ítem especificado. Se puede especificar un tiempo de retraso. El valor sólo se actualizará en el servidor (no lo hará en el campo del dispositivo requerido).

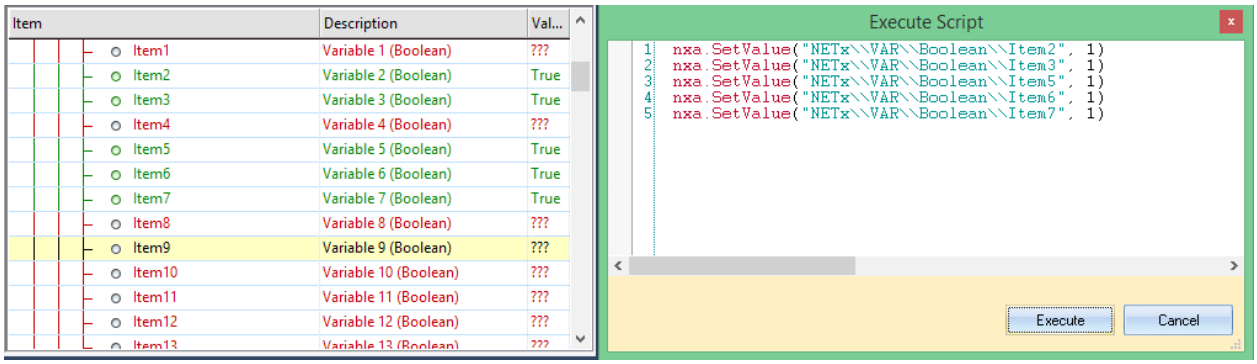
Parámetros: string con el ítemID del ítem a actualizar.
tipo con el valor que actualizará el ítem.
number con el retraso (en milisegundos).

Devuelve: (No aplicable)

Ej: nxa,LogInfo(nxa.SetValue("NETx\\VAR\\Boolean\\Item1", 1))



Debido a que el estado de la segunda variable es UNCERTAIN, el resultado obtenido es el valor por defecto, 0 (FALSE)



NXA.SETITEMDATA()

Función responsable de poner un valor especifico a un ítem en el servidor.

Similar a la Función nxa.SetValue, pero donde se puede configurar la información de origen y del tiempo.

Parámetros: string - ID del ítem que debe ser inicializado

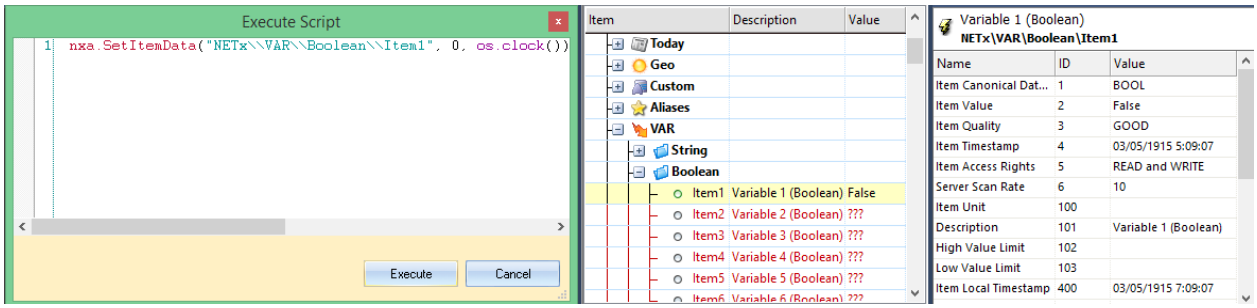
variable - nuevo valor que se debe inicializar

date - información temporal del valor. Grabado en el histórico de la BD

string - origen de la información

Devuelve: (No aplicable)

Ej: nxa.SetItemData("NETx\\VAR\\Boolean\\Item1", 0, os.clock(), "SYS:LUA;SRC:MyDevice")



os.clock() devuelve el tiempo que el programa lleva en ejecución, de ahí esta marca temporal (local timestamp) extraña

NXA.RESETITEMVALUE()

Función responsable de borrar un valor especifico de un ítem en el servidor.

Pone el valor a "UNCERTAIN".

Parámetros: string - ID del ítem que debe ser reiniciado

date - información temporal del valor. Grabado en el histórico de la BD string - origen de la información

Devuelve: (No aplicable)

Ej: nxa.LogInfo (nxa.ResetItemValue("NETx\\VAR\\Boolean\\Item1", os.clock(), "SYS:LUA ; SRC:MyDevice"))

NXA.READVALUE()

Función que hace activamente la lectura del valor de un ítem específico.

El valor leído solo envía la lectura del valor requerido para el dispositivo del campo.

Se puede añadir un retraso opcional elegible.

Parámetros: string - ID del ítem que debe ser leído

number - retraso en milisegundos

Devuelve: variable - valor actual del ítem

Ej: nxaLogInfo(nxa.ReadValue("NETx\\VAR\\Numeric\\Ítem1", 500))

NXA.WRITEVALUE()

Función que hace activamente la escritura del valor de un ítem específico.

El valor leído solo propaga la petición para el dispositivo de campo.

Se puede añadir un retraso opcional elegible.

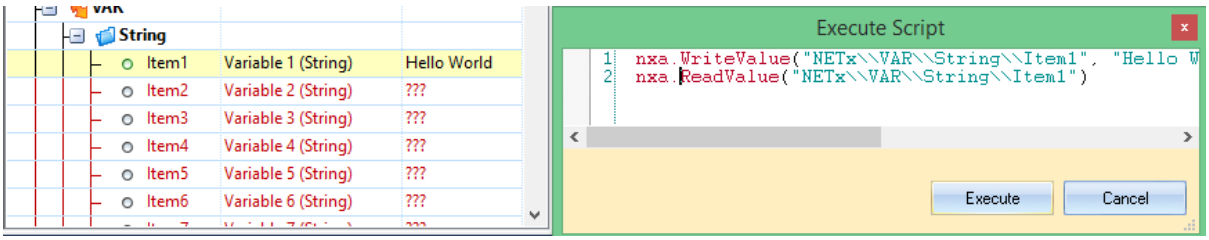
Parámetros: string - ID del ítem que debe ser leído

variable - valor del ítem a ser escrito

number - retraso en milisegundos

Devuelve: (No aplicable)

Ej:nxa.LogInfo(nxa.WriteValue("NETx\\VAR\\Numeric\\Ítem1", 123, 500)



NXA.ISVALIDVALUE()

Función que verifica si el ítem tiene un valor válido.

Parámetros: string - ID del ítem que debe ser testado

Devuelve: bool - indica si el valor es correcto o no

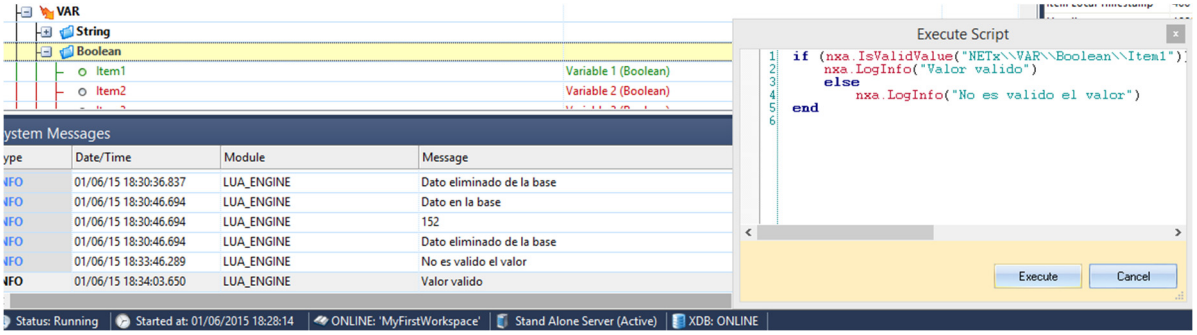
Ej: if (nxa.IsValidValue("NETx\\VAR\\Numeric\\Ítem1")) then

nxa.LogInfo("Valor valido")

else

nxa.LogInfo("El valor no es correcto.")

end



NXA.GETPROPERTYVALUE() | NXA.PROPERTYVALUE() | NXA.PROPERTY()

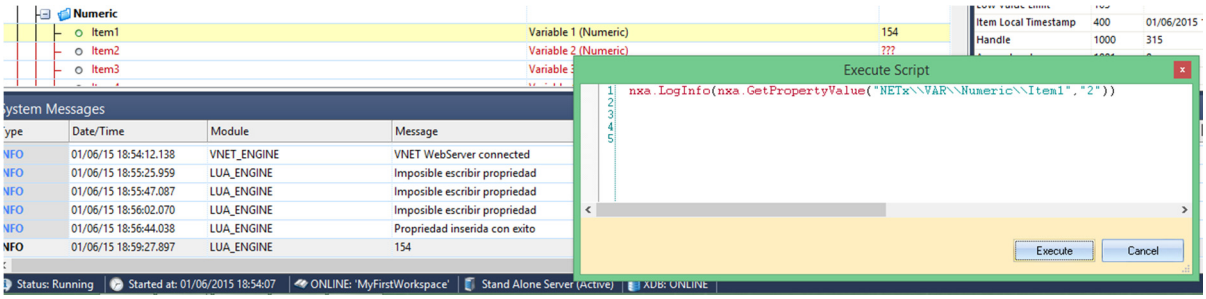
Función que lee las propiedades de un ítem en el servidor.

Parámetros: string - ID del ítem que contiene las propiedades

string - ID de la propiedad de que se lee

Devuelve: variable - valor actual de la propiedad

Ej: nxa.LogInfo(nxa.GetPropertyValue("NETx\\VAR\\Numeric\\Item1","2"))



NXA.SETPROPERTYVALUE() | NXA.SETPROPERTY()

Función que escribe un nuevo valor a las propiedades de un ítem en el servidor.

Si no es posible escribir, el resultado es false.

Parámetros: string - ID del ítem que contiene las propiedades

string - ID de la propriedad de que se desea escribir

variable - nuevo valor de la propriedad

Devuelve: bool - verdadero si la operación es exitosa

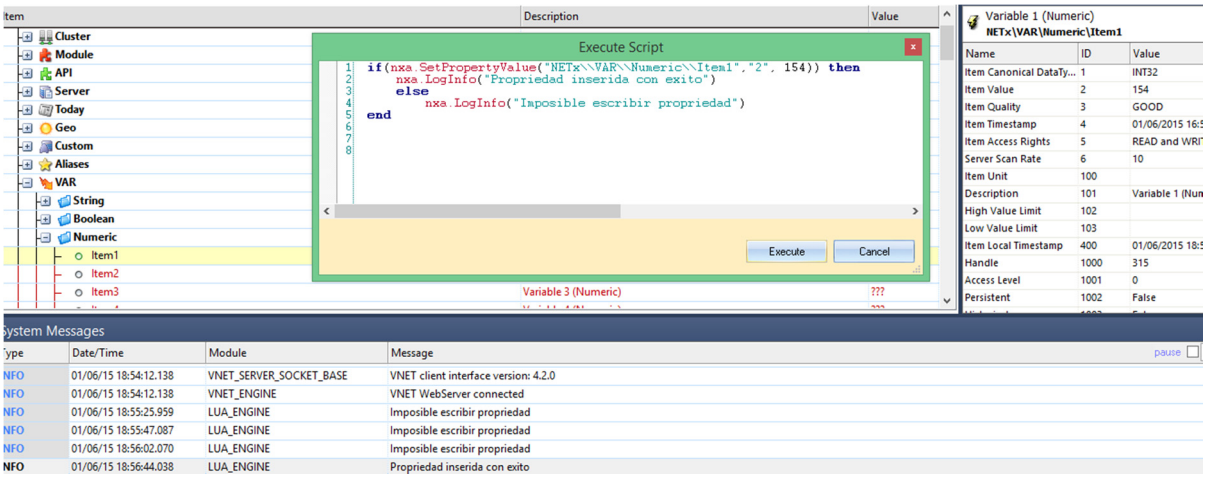
Ej: if (nxa.SetPropertyValue("NETx\\VAR\\Numeric\\Ítem1","2", 154)) then

nxa.LogInfo("Propriedad inserida con éxito")

else

nxa.LogInfo("Imposible escribir propiedad")

end



NXA.READVALUES()

Envía una petición KNX de lectura a todas las direcciones del grupo KNX o aquellas que además tienen activada la flag "ReadOnReconnect".

Parámetros: bool - Si es TRUE, el servidor envía la petición de lectura del KNX para el grupo KNX con flag "ReadOnReconnect"

Devuelve: (No Aplicable)

Ej: nxa.LogInfo(nxa.ReadValues(true))

NXA.GETITEMID() | NXA.ITEMID()

Función utilizada para recuperar un ItemID de un item en el servidor.

Es buscado por la dirección dada en KNX y la dirección de IP por KNX gateway.

Parámetros: string - dirección KNX

string - dirección IP del KNX gateway

Devuelve: string - ítemID del ítem especificado en la dirección KNX y la dirección de IP del gateway de KNX

Ej: nxa.LogInfo(nxa.GetÍtemID("NETx\\XIO\\KNX\\192.168.1.2","GATEWAY"))

XDB.SETDATA()

Función utilizada para almacenar cualquier tipo de datos persistentes.

Datos disponibles sólo después del reinicio del servidor.

Parámetros: string - Clave del valor

any - Valor almacenado de modo persistente en la base de datos

Devuelve: bool - Verdadero si lo ejecuta bien, falso si no

Ejemplo:

```
if (xdb.SetData("clave",152)) then
        nxa.LogInfo("Dato en la base")
        else
        nxa.LogInfo("Error")
 end
```

XDB.RESETDATA()

Función utilizada para quitar una pareja clave/valor de una base de datos persistente.

Parámetros: string - Clave del valor

Devuelve: bool - Verdadero si lo elimina bien, falso si no

Ejemplo:

```
 if (xdb.ResetData("clave")) then
        nxa.LogInfo("Dato eliminado de la base")
        else
        nxa.LogInfo("Error")
 end
```

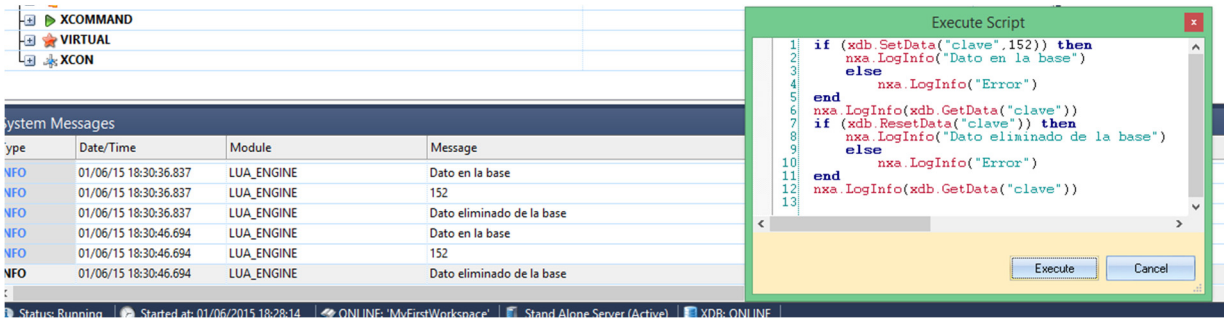
XDB.GETDATA()

Función utilizada para recuperar un dato almacenado en la base de datos persistente.
Datos disponibles sólo después del reinicio del servidor.

Parámetros: string - Clave del valor que se quiere recuperar

Devuelve: any - Valor almacenado conjuntamente con la clave introducida

Ejemplo: nxa.LogInfo(xdb.GetData("clave"))



NXA.GETVAR()

Función que recupera el dato de la variable global por su nombre.
El dato no es persistente y es perdido después del reinicio del servidor.

Parámetros: string - Nombre de la variable

Devuelve: any - Valor almacenado en la variable global según su clave

Ejemplo: nxa.LogInfo(nxa.GetVar("var"))

NXA.SETVAR()

Función que añade un valor a una variable global.
El dato no es persistente y es perdido después del reinicio del servidor.

Parámetros: string - Nombre de la variable
any - Valor a ser almacenado en la variable global

Devuelve: bool - Verdadero si añade bien, falso si no

Ejemplo:

```
if (nxa.SetVar("var",152)) then
        nxa.LogInfo("Valor en la variable")
        else
                nxa.LogInfo("Error")
 end
```

NXA.CLEARVAR()

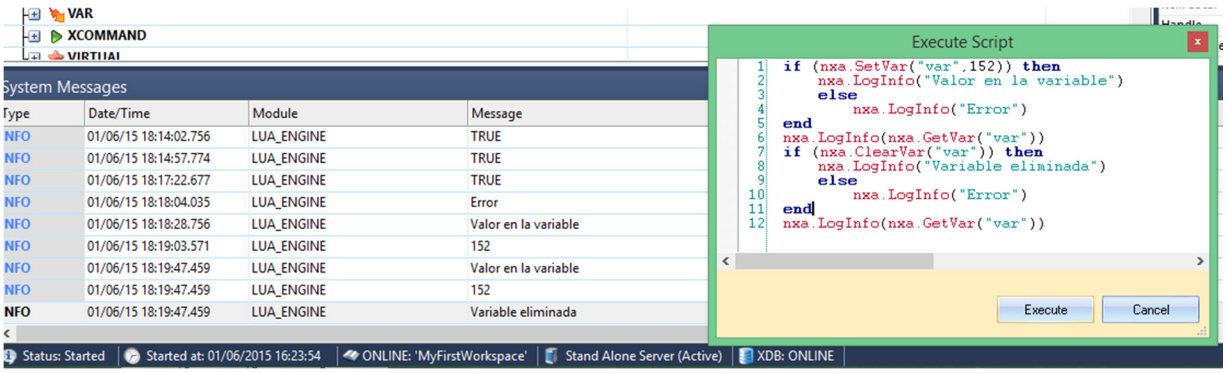
Función utilizada para quitar una pareja clave/valor de una variable.

Parámetros: string - Nombre de la variable

Devuelve: bool - Verdadero si elimina bien, falso si no

Ejemplo:

```
if (nxa.ClearVar("var")) then
        nxa.LogInfo("Variable eliminada")
        else
        nxa.LogInfo("Error")
 end
```



NXA.ADDDIRECTLINK()

Añade un enlace unidireccional entre la fuente y el elemento de destino.

Todas las peticiones (lectura, escritura y peticiones de asignación) se envían desde la fuente hasta el destino.

Parámetros: string - Origen del ÍtemID

string - Destino del ÍtemID

Devuelve: (No aplicable)

Ejemplo:

nxa.AddDirectLink("NETx\\VAR\\Boolean\\Item1","NETx\\VAR\\Boolean\\Item2")

NXA.REMOVEDIRECTLINK()

Elimina un enlace unidireccional entre dos ítems.

Parámetros: string - Origen del ItemID

string - Destino del ItemID

Devuelve: (No aplicable)

Ejemplo:

nxa.RemoveDirectLink("NETx\\VAR\\Boolean\\Item1","NETx\\VAR\\Boolean\\Item2")

NXA.ADDDIRECTBILINK()

Añade un enlace bidireccional entre la fuente y el elemento de destino.

Todas las peticiones (lectura, escritura y peticiones de asignación) se envían de un elemento a otro.

Parámetros: string - Ítem1 del ItemID

string - Ítem2 del ÍtemID

Ejemplo:

nxa.AddDirectBiLink("NETx\\VAR\\Boolean\\Item1","NETx\\VAR\\Boolean\\Item2")

NXA.REMOVEDIRECTBILINK()

Elimina un enlace bidireccional entre dos ítems.

Parámetros: string - Ítem1 del ÍtemID

string - Ítem2 del ÍtemID

Ejemplo:

nxa.RemoveDirectBiLink("NETx\\VAR\\Boolean\\Item1","NETx\\VAR\\Boolean\\Item2")

NXA.MAKEITEMSHADOWCOPY()

Crea un nuevo ítem que se comporta como una copia de la original.

Todas las solicitudes (lectura, escritura y asignación de valor) se reenvían a la copia y viceversa.

Parámetros: string - ItemID de la origen del ítem

string - El nuevo ítem de copia

Ejemplo:

```
nxa.MakeItemShadowCopy("NETx\\VAR\\Boolean\\Item1","NETx\\VAR\\Boolean\\Item2")
```

04 FUNCIONES NXA PARA CREAR ENLACES E ÍTEMS VIRTUALES

Estas funciones pueden ser usadas para crear ítems definidos por el usuario, también llamados Custom ítems. Estos ítems pueden ser organizados dentro de la jerarquía del servidor, permitiendo tener acceso completo a su funcionalidad. La diferencia es que estos Custom Ítems sólo están disponibles en el modelo de datos virtual del servidor, es decir que no hay un punto de datos físico donde estén conectados. Por lo tanto estos ítems pueden ser vistos como puntos de dato virtual.

Hay que tener en cuenta que los Custom ítems SOLO pueden ser creados y borrados cuando el servidor este iniciado o en marcha. (Ejemplo: dentro de la llamada a la función "OnInitEvent")

A continuación veremos las funciones para crear ítems definidos por el propio usuario, el uso y funcionalidad de cada una.

NXA.ADDCUSTOMITEM

Esta función nos permite añadir un nuevo ítem personalizado durante la fase de inicialización del servidor, y devuelve un ID, que identifica al ítem.

IMPORTANTE: Esta función sólo puede ser usada dentro de la llamada a la función "OnInitEvent" (ver la sección anterior de *callbacks*)

Parámetros:

1. Recibe como primer parámetro una cadena que indica el nombre que se le desea poner al elemento que estamos creando.
2. Recibe como segundo parámetro una cadena que describe el elemento personalizado.
3. Recibe como tercer parámetro del tipo nxa.access para indicar el tipo de acceso al ítem. Los accesos pueden ser de los siguientes tipos:
    - nxa.access.Readable : tipo de acceso de sólo lectura
    - nxa.access.Writeable: tipo de acceso de sólo escritura
    - nxa.access.All: tipo de acceso de lectura y escritura.

- Recibe como cuarto parámetro de tipo nxa.type para indicarle el tipo de dato del elemento. Los tipos a elegir son los siguientes:
    - nxa.type.Integer: tipo integer

- nxa.type.Real: tipo números reales
- nxa.type.Date: tipo fecha
- nxa.type.String: tipo cadena
- nxa.type.Boolean: tipo booleano, verdadero o falso.

Como parámetros opcionales tenemos:

- Tipo String "delimitador": que indica el delimitador que se utiliza para construir el identificador del ítem.
- Tipo String "path": para indicar la ruta que almacena el elemento personalizado. Se pueden añadir más de una ruta.

Esta función devuelve el identificador "ID" del elemento que hemos creado.

Implementación:

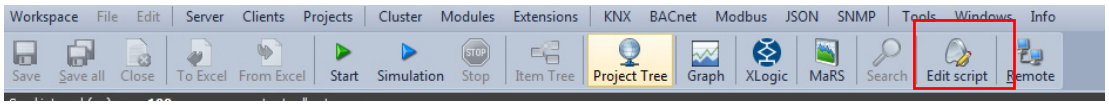
Nxa.AddCustomItem ("Elemento1", "descripción del elemento", nxa.access.All, nxa.type.Real, "/", "Ruta1", "Ruta2")

Para la ejecución de la función, el BMS Server nos proporciona unos scripts, que contiene las funciones a ejecutarse. El fichero principal que se va a utilizar es el "nxaDefinitions.lua", el cual contiene la función "OnInitEvent" donde vamos a llamar a las funciones de creación de ítems.

El fichero se encuentra en la siguiente ruta:

*C:\Program Files \NETxAutomation\NETx.BMS.Server.2.0\Workspaces**MiProyecto**\ScriptFiles*

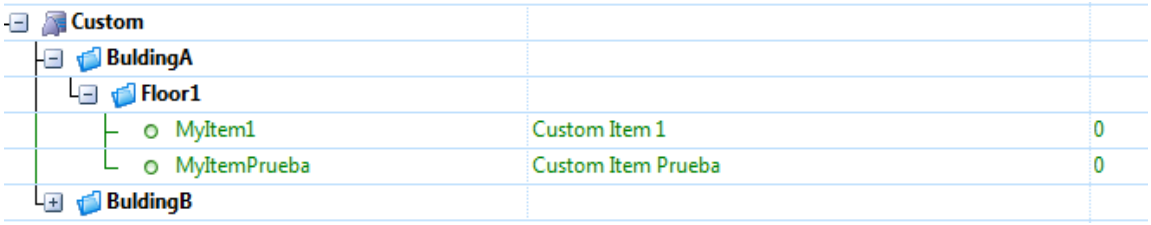
Desde el BMS Server tenemos la opción ***"Edit Script"*** → ***ScriptFiles***



Una vez abierto el archivo "*nxaDefinitions*" dentro de la función "*OnInitEvent*" creamos nuestro ítem de la siguiente forma:

```
function OnInitEvent()
  InitializeSysXCON()
  Example()

  -- Add your custom items here
  nxa.AddCustomItem("MyItem1",
   "Custom Item 1",
    nxa.access.All,
    nxa.type.Real,"/",
    "BuldingA",
    "Floor1")

  end
```

En este ejemplo hemos creado un nuevo Custom Ítem llamado "MyItem1" que se encuentra dentro del directorio: "Custom/BuldingA/Floor1/MyItem1"
Para comprobar que se ha creado el Custom Ítem en el árbol de ítems podemos verlo dentro de la opción Custom, de la siguiente forma:



NXA.ADDEXTCUSTOMITEM

Esta función añade un nuevo elemento personalizado durante la fase de inicialización del servidor y devuelve el identificador del elemento creado, de la misma forma como lo hace la anterior función. Pero a diferencia de la anterior "*Nxa.AddCustomÍtem*" se puede añadir una bandera, la cual se puede configurar como "Persistente", "Histórico" y "Sincronizado". El parámetro "**Persistent**" nos indicara si el valor del ítem es persistente, es decir que es restaurado desde la base de datos después de la puesta en marcha del servidor; el parámetro "**Historical**" especifica si los valores históricos del ítem se almacenan en la base de datos o no; y el parámetro "**Synchronize**" indica que el valor se sincroniza entre el servidor principal y el de respaldo si se establece como "TRUE".

Parámetros de entrada:

- Recibe como primer parámetro una cadena que indica el nombre que se le desea poner al ítem que estamos creando.
- Recibe como segundo parámetro una cadena que describe el Custom ítem.
- Recibe como tercer parámetro del tipo nxa.access para indicar el tipo de acceso al ítem.
- Recibe como cuarto parámetro de tipo nxa.type para indicarle el tipo de dato del elemento.
- Recibe un quinto parámetro un tipo booleano (true o false), que indica si es "Persistent" o no.
- Un sexto parámetro de tipo booleano (true o false), para establecer si es de tipo "Historical"
- Un séptimo parámetro de tipo booleano (true o false), para establecer si es de tipo "Synchronize"

Como parámetros opcionales tenemos:

- Tipo String "delimitador": que indica el delimitador que se utiliza para construir el Identificador del elemento.
- Tipo String "path": para indicar la ruta que almacena el elemento personalizado. Se pueden añadir más de una ruta.

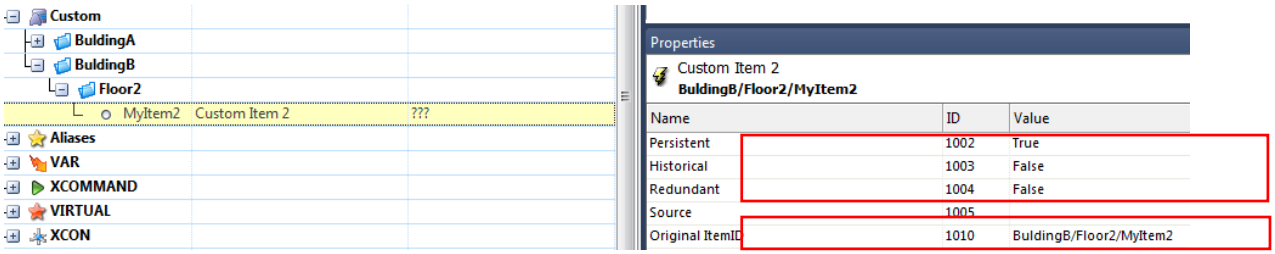
Implementación:

Para implementar esta función, igual que la anterior, dentro de la función "*OnInitEvent*" que se encuentra dentro del script *nxaDefinitions* la añadimos de la siguiente forma:

```
nxa.AddExtCustomItem("MyItem2",
 "Custom Item 2",
 nxa.access.All,
 nxa.type.Real,
 true, false,
 false,"/",
 "BuldingB", "Floor2")
```

En este ejemplo se ha creado un nuevo ítem llamada MyÍtem2 el cual solo tiene la bandera activada a "Persistent" y las demás desactivas.

Para comprobarlo podemos verlo en el árbol de ítems



También es posible ver las propiedades de los ítems, haciendo click derecho sobre el ítem creado y en la opción "*show properties*", y ver el ítemID que nos devuelve, que en este caso es:

BuldingB/Floor2/MyItem2

Otra forma de ver el ítemID es haciendo "*copy ítem path to clipboard*" (botón derecho).

NXA.ADDSYSCUSTOMITEM

Esta función permite añadir un nuevo ítem del sistema durante la fase de inicialización del servidor y devuelve un identificador del elemento (ID). En comparación con los anteriores, con esta función un Custom ítems del sistema nuevo se puede colocar en cualquier lugar del árbol de ítems (Ítem Tree).

En general esta función debe ser usada en la llamada a la función "*OnInitEvent*"

Parámetros:

- Recibe como primer parámetro una cadena que indica el nombre que se le desea poner al ítem que estamos creando.
- Recibe como segundo parámetro una cadena que describe el Custom ítem.
- Recibe como tercer parámetro del tipo nxa.access para indicar el tipo de acceso al ítem.
- Recibe como cuarto parámetro de tipo nxa.type para indicarle el tipo de dato del ítem.
- Recibe un parámetro de tipo cadena que describe las unidades del Custom ítem.
- Recibe un parámetro de tipo number donde se añade el valor mínimo permitido para el Custom ítem. (opcional)
- Recibe un parámetro tipo number que describe el máximo valor permitido para el Custom ítem (opcional)

Como parámetros opcionales tenemos:

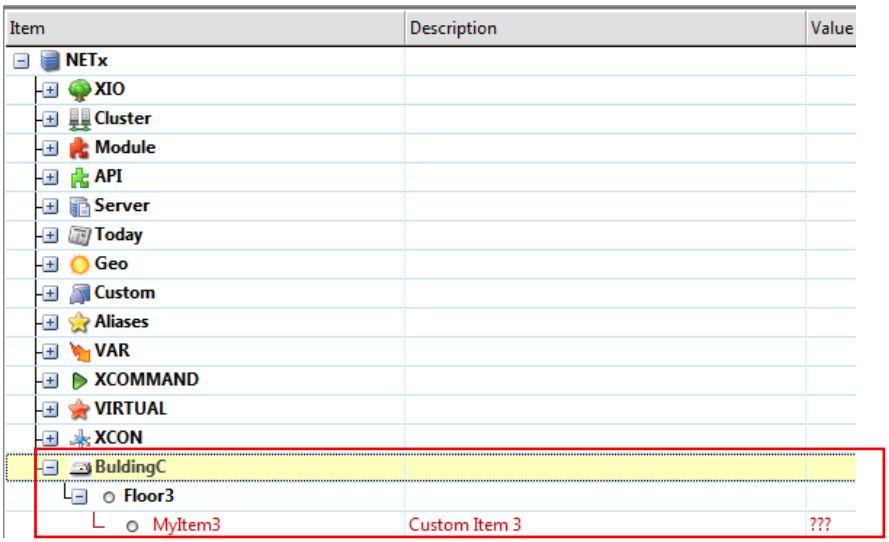
- Tipo String "delimitador": que indica el delimitador que se utiliza para construir el Identificador del ítem.
- Tipo String "path": para indicar la ruta que almacena el Custom ítem. Se pueden añadir más de una ruta.

Implementación:

Dentro de la función OnInitEvent añadimos lo siguiente:

```
nxa.AddSysCustomItem("MyItem3",
 "Custom Item 3",
  nxa.access.All,
  nxa.type.Real,
  "uno",1,3,"/",
  "BuldingC", "Floor3")
```

Y podemos comprobar dentro de la jerarquía de nuestro árbol de ítems que se ha añadido un nuevo ítem:



NXA.ADDEXTSYSCUSTOMITEM

Esta función permite añadir un nuevo elemento del sistema durante la fase de inicialización del servidor y devuelve el identificador correspondiente a ese elemento (ID). A diferencia del anterior, en esta función podemos configurar las banderas como "Persistente", "Histórica", y "Sincronizada."

Parámetros de entrada:

- Recibe como primer parámetro una cadena que indica el nombre que se le desea poner al ítem que estamos creando.
- Recibe como segundo parámetro una cadena que describe el Custom ítem.
- Recibe como tercer parámetro del tipo nxa.access para indicar el tipo de acceso al ítem.
- Recibe como cuarto parámetro de tipo nxa.type para indicarle el tipo de dato del ítem.
- Recibe un quinto parámetro un tipo booleano (true o false), que indica si es "Persistent" o no.
- Un sexto parámetro de tipo booleano (true o false), para establecer si es de tipo "Historical"
- Un séptimo parámetro de tipo booleano (true o false), para establecer si es de tipo "Synchronize

Como parámetros opcionales tenemos:

- Recibe un parámetro de tipo cadena que describe las unidades del Custom ítem.

- Recibe un parámetro de tipo number donde se añade el valor mínimo permitido para el Custom ítem. (opcional)
- Recibe un parámetro tipo number que describe el máximo valor permitido para el Custom ítem (opcional)
- Tipo String "delimitador": que indica el delimitador que se utiliza para construir el Identificador del elemento.
- Tipo String "path": para indicar la ruta que almacena el elemento personalizado. Se pueden añadir más de una ruta.

Implementación:

Se implementa de la misma forma que el anterior pero con las siguientes modificaciones:

```
nxa.AddExtSysCustomItem ("MyItemSys",
 "Custom Item Sys",
  nxa.access.All,
  nxa.type.Real,
  true, false, false,
  "uno",1,3,"/",
  "BuldingD", "Floor2")
```

NXA.ADDITEMLINK

Esta función enlaza un ítem origen a un ítem destino. De tal forma que el valor del elemento origen es enviado al ítem destino con un retraso dado.

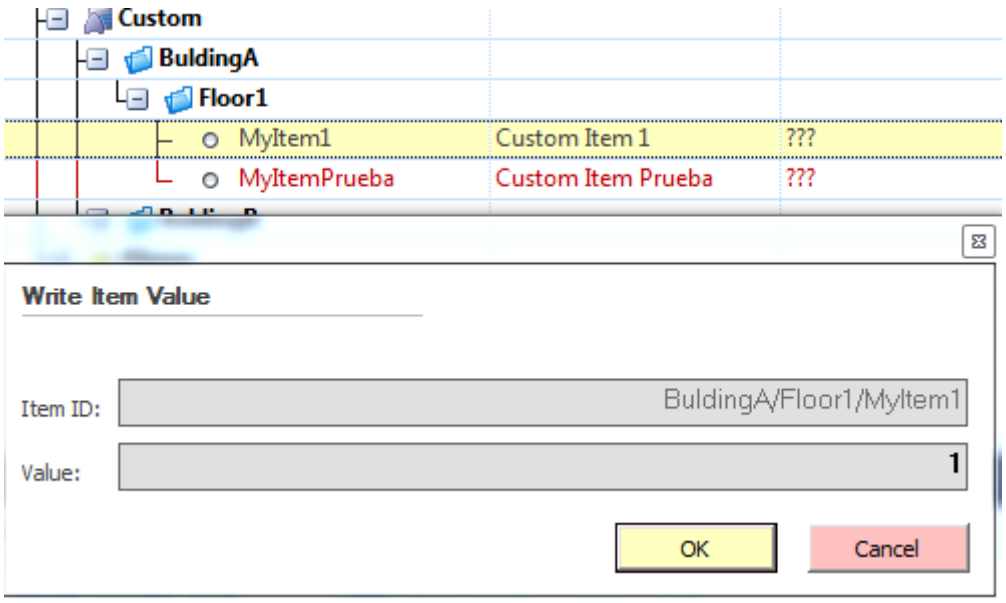
Parámetros de entrada:

- El primer parámetro indica el identificador del ítem origen.
- El segundo parámetro indica el identificador del ítem destino.
- El tercer parámetro indica el retraso en milisegundos de enviar desde origen a destino.

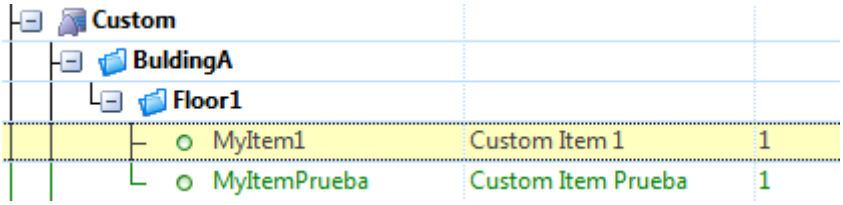
Implementación:

```
function OnInitEvent()
  InitializeSysXCON()

 nxa.AddItemLink("BuldingA/Floor1/MyItem1","BuldingA/Floor1/MyItem2",100)

end
```

Para comprobar que funciona, dentro del árbol de ítems, escribimos un valor en el ítem origen y se escribe ese valor al ítem destino de la siguiente forma



Y ahora se muestra el valor en los dos ítems:



NXA.ADDDIRECTLINK

Esta función enlaza de forma directa un ítem origen con uno destino, es decir que el valor del ítem destino cambia en el mismo instante el que el del ítem origen es cambiado.

Parámetros de entrada:

- El primer parámetro indica el identificador del elemento origen.
- El segundo parámetro indica el identificador del elemento destino.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()

 nxa.AddDirectLink("BuldingA/Floor1/MyItem1","BuldingA/Floor1/MyItem2")

end
```

NXA.REMOVEDIRECTLINK

Esta función nos permite borrar un enlace directo entre un elemento origen y uno destino.

Parámetros de entrada:

- El primer parámetro indica el idenficador del elemento origen.
- El segundo parámetro indica el identificador del elemento destino.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()
 nxa.RemoveDirectLink("BuldingA/Floor1/MyItem1","BuldingA/Floor1/MyItem2")

end
```

NXA.ADDDIRECTBLINK

Esta función nos permite hacer un enlace bidireccional, es decir que el valor se puede transmitir de origen a destino y viceversa.

Parámetros de entrada:

- El primer parámetro indica el idenficador del elemento origen.
- El segundo parámetro indica el identificador del elemento destino.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()
nxa.AddDirectBiLink("BuldingA/Floor1/MyItem1","BuldingA/Floor1/MyItem2")

end
```

NXA.REMOVEDIRECTBILINK

Esta función nos permite borrar los enlaces bidireccionales entre ítems origen y destino.

Parámetros de entrada:

- El primer parámetro indica el idenficador del elemento origen.
- El segundo parámetro indica el identificador del elemento destino.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()
nxa.RemoveDirectBiLink("BuldingA/Floor1/MyItem1","BuldingA/Floor1/MyItem2")

end
```

NXA.ADDITEMEVENT

Esta función permite añadir un evento, el cual se activara cada vez que el elemento cambie.

Parámetros de entrada:

- El primer parámetro indica el identificador del elemento origen, el cual activara el evento.
- El segundo parámetro dependiendo del valor que le demos-false, true-el ítem será activado o no si ha recibido una petición de lectura.
- El tercer parámetro dependiendo del valor que le demos-false, true-el ítem será activado o no si escribimos en él.
- El cuarto parámetro dependiendo del valor que le demos-false, true-el ítem será activado o no si se ha modificado el valor.
- El quinto parámetro nos indica el tiempo que tarda en ejecutarse el evento una vez se realice la acción sobre el ítem. Está en milisegundos.
- En el último parámetro ponemos la función que ejecuta el evento. Debe tener en cuenta que como este parámetro es de tipo string debe ir entre comillas.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()
nxa.AddItemEvent("BuldingA/Floor1/MyItem1",false,true,false,10000,"HelloWorld()")

end
```

Una vez que cambiemos el valor en el ítem se ejecuta el script del evento que hemos creado, en este caso nuestro script es HelloWorld y va a escribir por consola HelloWorld.

NXA.ADDITEMTEMPLATE

Esta función nos permite crear una nueva plantilla para los ítems personalizados que tenemos creados, y los añade a la lista de plantillas dentro del servidor.

Implementación:

```
1  function template()
2
3      return  nxa.LogInfo(nxa.AddItemTemplate())
4
5  end
```

Devuelve el identificador de la plantilla, es decir el ID que identifica de forma exclusiva la plantilla.

NXA.ADDPROPERTYTEMPLATE

Una vez tengamos creada la plantilla con la función vista anteriormente, con esta función podemos añadir nuevas propiedades a la plantilla.

Parámetros:

- El primer parámetro que recibe esta función es el identificador de la plantilla de tipo numérico, creado anteriormente con la función anterior.
- El segundo parámetro le pasamos el Id que identifica la propiedad dentro de la plantilla. El rango se comprende entre 7000-7999
- El tercer parámetro indica el nombre de la propiedad, por tanto es de tipo cadena.
- El cuarto parámetro indica el tipo de la propiedad. De tipo nxa.type.
- El último parámetro indica el valor por defecto de la propiedad.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()

 nxa.AddPropertyTemplate(nxa.AddItemTemplate(),7001,"template1",nxa.type.Real,11)
 end
```

NXA.ATTACHTEMPLATE

Con esta función asigna una plantilla a un custom ítem.

Parámetros:

- En el primer parámetro indica el identificador del ítem a enlazar.
- En el segundo parámetro ponemos el identificador del template que queremos enlazar.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()

 nxa.AttachTemplate("BuldingA/Floor1/MyItem1",39)
 end
```

NXA.DETACHTEMPLATE

Esta función separa una plantilla unida a un ítem.

Parámetros:

Recibe un único parámetro de tipo string que indica el identificador del ítem en la plantilla que se le ha sido asignada.

Implementación:

```
function OnInitEvent()
  InitializeSysXCON()

 nxa.DetachTemplate("BuldingA/Floor1/MyItem1")
 end
```

05 FUNCIONES PARA EL MANEJO DE TAREAS (TASK HANDLING)

Éste conjunto de funciones son usadas en combinación con las tareas. Algunas funciones proveen funcionalidad de ayuda que sólo pueden ser llamadas dentro de las funciones LUA que son ejecutadas por las tareas (Sección 4.5.1 del manual del BMS Server). Además, hay funciones incluidas que pueden ser usadas para añadir o eliminar definiciones de tareas directamente desde scripts LUA. Usarlas nos ofrece la oportunidad de crear dinámicamente tareas que no estás listadas en el archivo de definición de tareas. NOTA: la agregación o la eliminación de tareas puede ser sólo realizado durante el arranque del servidor (en el callback "OnInitEvent").

NXA.SOURCEITEMID | NXA.SOURCEVAR

Esta función puede ser utilizada para preguntar el ID del objeto de origen que lanzó la tarea correspondiente. Sólo puede ser usada en las funciones LUA que son invocadas por tareas.

Devuelve:

- string – ID del objeto de origen que lanzó la tarea.

NXA.DESTINATIONITEMID

Esta función puede ser utilizada para preguntar el ID del objeto de destino que está configurado para actuar con la tarea correspondiente. Sólo puede ser usada en las funciones LUA que son invocadas por tareas.

Devuelve:

- string – ID del objeto de destino relacionado con la tarea.

NXA.SETDESTINATIONVALUE

Esta función puede ser usada para establecer un nuevo valor para el objeto de destino que está configurado en la correspondiente tarea. Sólo puede ser usada en funciones LUA invocadas por tareas. NOTA: el nuevo valor sólo será modificado en el

servidor (no llega al dispositivo).

Devuelve:

- variant – nuevo valor del objeto de destino.

NXA.WRITEDESTINATIONVALUE

Esta función puede ser usada para escribir un nuevo valor para el objeto de destino que está configurado en la correspondiente tarea. Comparado a poner un valor, un valor escrito se propagará también al campo del dispositivo. Sólo puede ser usada en funciones LUA invocadas por tareas.

Devuelve:

- variant – nuevo valor del objeto de destino.

NXA.READDESTINATIONVALUE

Esta función puede ser usada para leer el valor actual del objeto de destino que está configurado en la correspondiente tarea. Esta función puede ser usada sólo en funciones LUA que son invocadas por tareas.

Devuelve:

- variant – valor actual del objeto de destino

NXA.INPUTVALUE | NXA.SOURCEVALUE

Esta función puede ser usada para consultar el valor del objeto de origen que ha lanzado la tarea correspondiente. Esta función sólo puede ser usada por funciones LUA que son lanzadas por tareas.

Devuelve:

- variant – valor actual del objeto de origen

NXA.EXECUTEDELAYEDSCRIPT

Esta función ejecuta un script LUA especificado como un parámetro String. La ejecución puede ser opcionalmente retrasada.

Parametros:

- string – nombre del script LUA
- number – retraso en milisegundos (opcional)

NOTA: la agregación o la eliminación de tareas puede ser sólo realizado durante el arranque del servidor (OnInitEvent()).

NXA.ADDWRITETASK

Esta función añade una nueva definición para una tarea de 'escritura' durante la ejecución.

Parámetros:

- string – ID del objeto de origen del servidor que lanza la tarea.
- string – ID del objeto de destino del servidor que es cambiado por la tarea.
- bool – corresponde al parámetro "OnReceive" de la definición de la tarea.
- bool – corresponde al parámetro "OnSend" de la definición de la tarea.
- bool – corresponde al parámetro "OnSet" de la definición de la tarea.
- number – intervalo de tiempo en milisegundos que la tarea será retrasada cuando se lance.
- variant – si es usado, este valor será usado en vez del valor del objeto de origen (opcional).

NXA.ADDREADTASK

Esta función añade una nueva definición para una tarea de 'lectura' durante el periodo de ejecución.

Parámetros:

- string – ID del objeto de origen del servidor que lanza la tarea.
- string – ID del objeto de destino del servidor que es cambiado por la tarea.
- bool – corresponde al parámetro "OnReceive" de la definición de la tarea.
- bool – corresponde al parámetro "OnSend" de la definición de la tarea.
- bool – corresponde al parámetro "OnSet" de la definición de la tarea.
- number – intervalo de tiempo en milisegundos que la tarea será retrasada cuando se lance.

NXA.ADDSETTASK

Esta función añade una nueva definición para una tarea de 'establecer' durante el periodo ejecución.

Parámetros:

- string – ID del objeto de origen del servidor que lanza la tarea.
- string – ID del objeto de destino del servidor que es cambiado por la tarea.
- bool – corresponde al parámetro "OnReceive" de la definición de la tarea.
- bool – corresponde al parámetro "OnSend" de la definición de la tarea.
- bool – corresponde al parámetro "OnSet" de la definición de la tarea.
- number – intervalo de tiempo en milisegundos que la tarea será retrasada cuando se lance.
- variant – si es usado, este valor será usado en vez del valor del objeto de origen (opcional).

NXA.ADDSCRIPTTASK

Esta función añade una nueva definición para una tarea de 'script' durante el periodo ejecución.

Parámetros:

- string – ID del objeto de origen del servidor.
- string – ID del objeto de destino del servidor.
- bool – corresponde al parámetro "OnReceive" de la definición de la tarea.
- bool – corresponde al parámetro "OnSend" de la definición de la tarea.
- bool – corresponde al parámetro "OnSet" de la definición de la tarea.
- number – intervalo de tiempo en milisegundos que la tarea será retrasada

cuando se lance.

- variant – si es usado, este valor será usado en vez del valor del objeto de origen (opcional).

NXA.ADDVARTASK

Esta función puede ser usada para añadir una tarea que es lanzada si una variable cambia su valor.

Parámetros:

- string – nombre de la variable que será el lanzador de la tarea.
- string – nombre de la función LUA que será lanzada.
- number – retraso opcional para la función.

Devuelve:

- bool – cierto en caso de éxito, falso en cualquier otro caso.

TUTORIAL DE USO

Se pueden añadir tareas de distintas formas, la principal es a través del menú Extensions → [Live] Task definitions. La alternativa a esto es mediante la programación de scripts mediante LUA, a través de la opción Edit Script, seleccionando 'nxaDefinitions.lua'.

Para añadir las distintas tareas (nxa.AddWriteTask, nxa.AddReadTask, nxa. AddSetTask…) se debe hacer en la función 'OnInitEvent', es la función que se ejecuta al iniciar el servidor.

Ejemplo:

```
function OnInitEvent()
InitializeSysXCON()
-- función que cada vez cambia los segundos del programa, se lee el Item1 de las variables String
nxa.AddReadTask(("NETx\\Today\\Seconds","NETx\\VAR\\String\\Item1", true, true, true, 0)
--  función que cada vez que cambia los segundos del sistema, se guardan en el Item1 de las variables String el valor de origen (que lanza la tarea)
  nxa.AddWriteTask("NETx\\Today\\Seconds","NETx\\VAR\\String\\Item1", true, true, true, 0, nxa.SourceValue)
--función idéntica al write pero la variación no llega al dispositivo
 nxa.AddSetTask("NETx\\Today\\Seconds","NETx\\VAR\\String\\Item3", true, true, true, 100, nxa.SourceValue)
end
```

Además, se pueden crear scripts .lua aparte donde podemos crear nuestras funciones y poder hacer un tratamiento de datos más exhaustivos, que luego invocaremos con la función nxa.AddScriptTask dentro de la función 'OnInitEvent'.

Ejemplo:

```
require 'script'
function OnInitEvent()
InitializeSysXCON()
nxa.AddScriptTask("NETx\\String\\Item1", "NETx\\String\\Item2", true, true, true, 0, scriptPrint())
end
```

Con la misma idea, se puede hacer el mismo procedimiento pero dependiendo de variables, lo que hace que cada vez que cambie la variable se ejecuta el script.

Ejemplo:

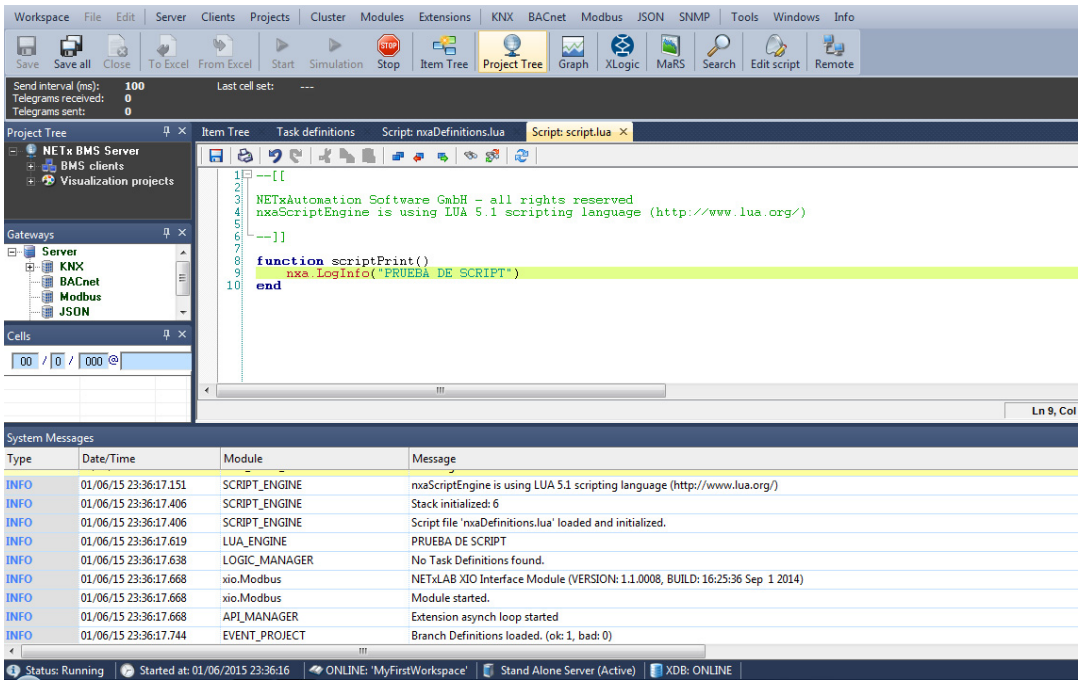
```
require 'script'
function OnInitEvent()
InitializeSysXCON()
nxa.AddVarTask("NETx\\String\\Item1", scriptPrint(), 0)
end

-- script.lua
function scriptPrint()
-- muestra en la consola el ID del objeto de origen, se puede utilizar cualquier función aquí
nxa.LogInfo(nxa.SourceItemID())
end
```

Las funciones se pueden subdividir en dos grupos:

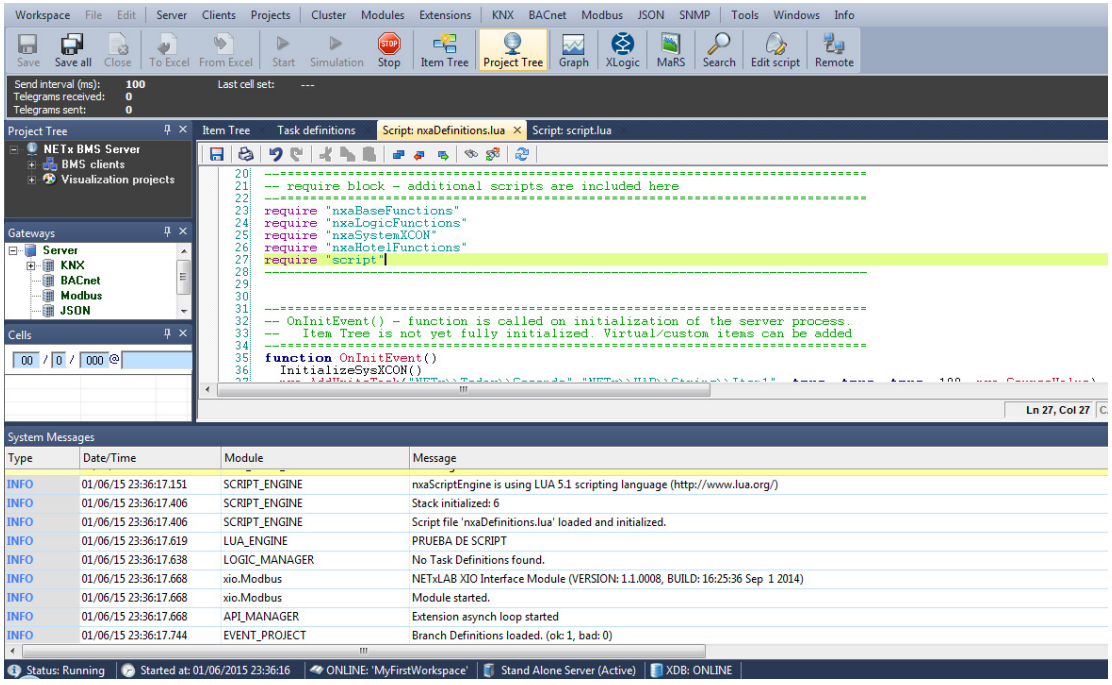
- Funciones para crear tareas
- Funciones que puede leer o modificar los valores de los parámetros de entrada de las tareas.

Las primeras se deben definir en la función 'OnInitEvent' cómo ya se ha comentado. El resto puede llamarse dentro de un script que lance una tarea, o incluso como uno de los parámetros de entrada de las funciones de creación.

Se escribe un nuevo fichero con las funciones que deseemos.

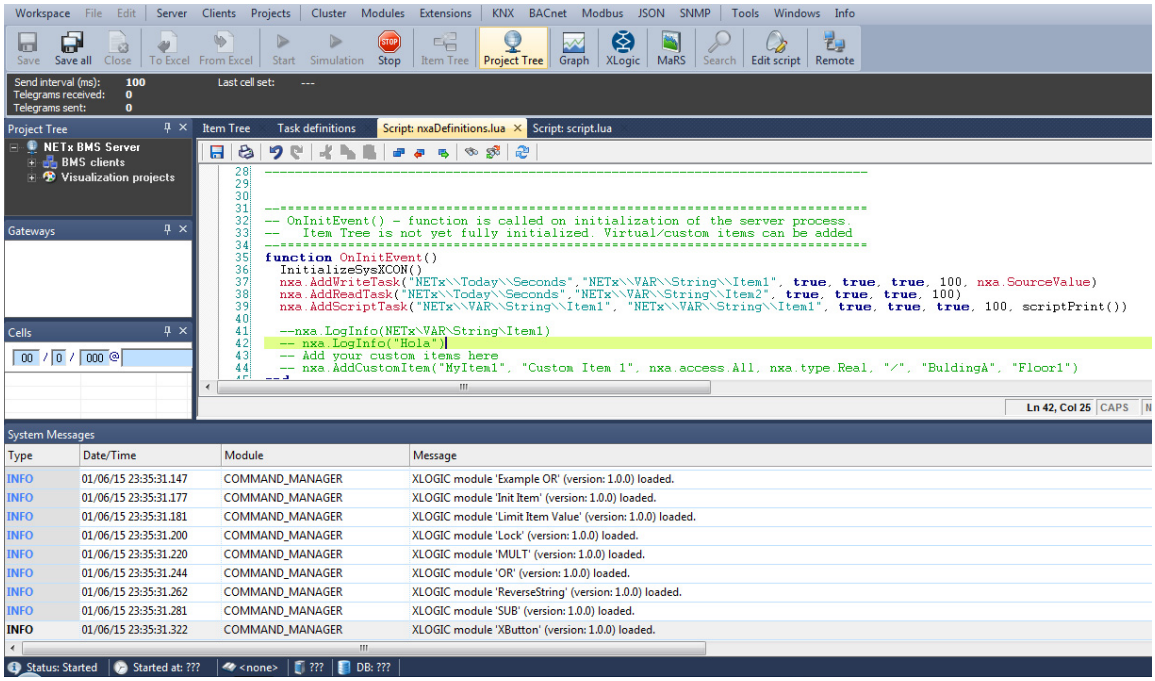


En el archive 'nxaDefinitions.lua' añadimos el script anterior con 'require', de ésta manera ya podemos llamar las funciones creadas.

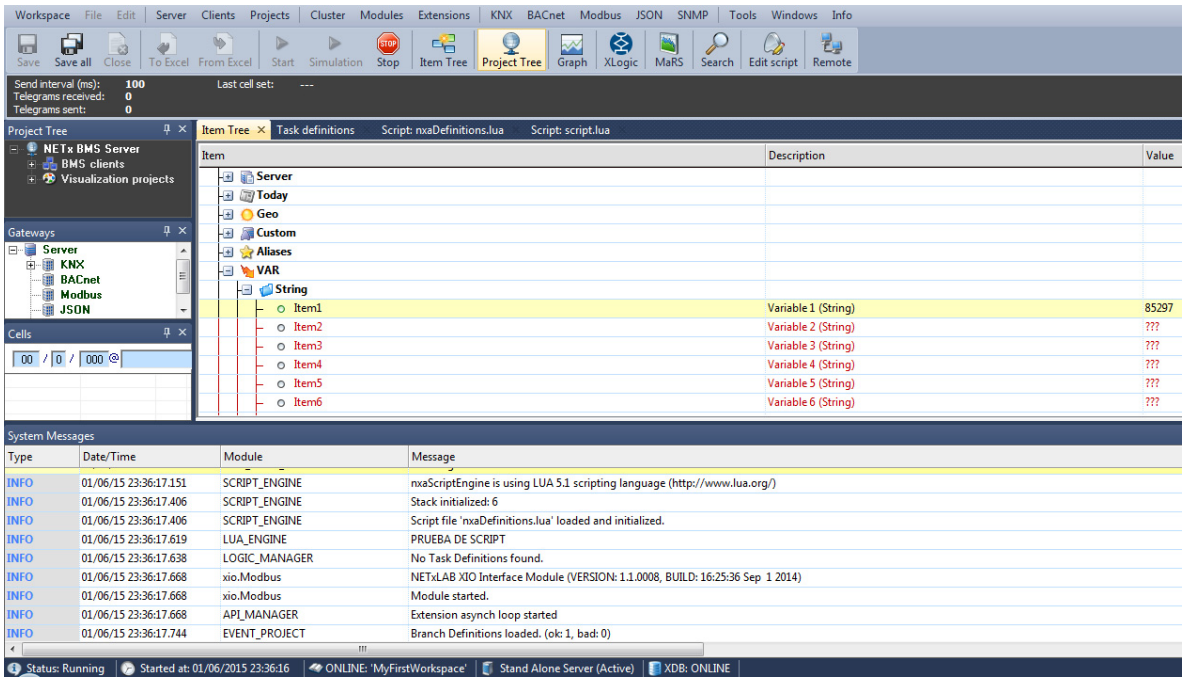


En 'nxaDefinitions' definimos la creación de tareas. Tanto en éstas funciones como en el script adicional podemos utilizar el resto de las funciones para obtener los parámetros de las funciones y realizar las operaciones que queramos, además de las

funciones del resto del sistema.



Aquí se puede ver el resultado de una función que escribe en el Item1 de String los segundos que cuenta el servidor, el objeto 'Seconds' del sistema lanza la tarea como se ha definido en la tarea y se escribe su valor que se recoge mediante 'nxa.SourceValue'.



06 FUNCIONES DATE AND TIME

En esta sección se repasan las funciones de gestión de datos relativos a la fecha y la hora.

NXA.NOW

Devuelve el día y la hora.

Script de ejemplo:

```
function Now()
nxa.LogInfo(nxa.Now())
end
```

Ejemplo de llamada: Now()

Resultado obtenido: 42156.748229167

NXA.TIMEONLY

Devuelve la hora actual.

Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function TimeOnly(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.TimeOnly(date))
end
```

Ejemplo de llamada: TimeOnly(2015, 10, 5, 6, 15, 48)

Resultado obtenido: 0.260972222

NXA.DATEONLY

Devuelve el día actual.

Recibe como parámetros el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function DateOnly(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
```

nxa.LogInfo(nxa.DateOnly(date))
end
Ejemplo de llamada: DateOnly(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 42282

NXA.YEAR

Devuelve el año.
Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.

Script de ejemplo:
function Year(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Year(date))
end
Ejemplo de llamada: Year(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 2015

NXA.MONTH

Devuelve el mes.
Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:
function Month(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Month(date))
end
Ejemplo de llamada: Month(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 10

NXA.DAY

Devuelve el día.
Recibe como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function Day(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Day(date))
end
```

Ejemplo de llamada: Day (2015, 10, 5, 6, 15, 48)
Resultado obtenido: 5

NXA.HOUR

Devuelve la hora.
Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function Hour(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Hour(date))
end
```

Ejemplo de llamada: Hour(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 6

NXA.MINUTE

Devuelve los minutos.
Recibe como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function Minute(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Minute(date))
end
```

Ejemplo de llamada: Minute(2015, 10, 5, 6, 15, 48)

Resultado obtenido: 15

NXA.SECOND

Devuelve los segundos.
Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function Second(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.Second(date))
end
```

Ejemplo de llamada: Second (2015, 10, 5, 6, 15, 48)
Resultado obtenido: 48

NXA.DAYOFWEEK

Devuelve el día de la semana.
Le pasamos como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function DayOfWeek(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.DayOfWeek(date))
end
```

Ejemplo de llamada: DayOfWeek(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 1

NXA.MAKEDATE

Crea una variable de tipo fecha.
Recibe como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function MakeDate(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(date)
end
```

Ejemplo de llamada: MakeDate (2015, 10, 5, 6, 15, 48)
Resultado obtenido: 42282.260972222

NXA.MAKETIMEONLY

Crea una variable de tipo fecha que solo contiene la información de la hora.
Le pasamos como parámetros la hora, los minutos y los segundos.
Script de ejemplo:

```
function MakeTimeOnly(hour,minute,second)
local date = nxa.MakeTimeOnly(hour,minute,second)
nxa.LogInfo(date)
end
```

Ejemplo de llamada: MakeDateOnly(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 0.2609722222

NXA.ADDDATE

Añade una fecha a otra.
Le pasamos como parámetros dos fechas. Ambas fechas tendrán como parámetros el año, mes, día, hora, minutos y segundos.
Script de ejemplo:

```
function
AddDate(year,month,day,hour,minute,second,year2,month2,day2,hour2,minute2,seco
nd2)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
local date2 = nxa.MakeDate(year2,month2,day2,hour2,minute2,second2)
nxa.LogInfo(nxa.AddDate(date,date2))
end
```

Ejemplo de llamada: MakeTimeOnly(6, 15, 48)
Resultado obtenido: 885.07620373037

NXA.DIFFDATE

Resta una fecha a otra fecha.
Recibe como parámetros dos fechas. Ambas fechas tendrán como parámetros el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function DiffDate(year,month,day,hour,minute,second,year2,month2,day2,hour2,minute2,second2)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
local date2 = nxa.MakeDate(year2,month2,day2,hour2,minute2,second2)
nxa.LogInfo(nxa.DiffDate(date,date2))
end
```

Ejemplo de llamada: DiffDate(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 885.07662037037

NXA.DATETOSTRING

Convierte una fecha en una cadena.
Recibe como parámetros el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function DateToString(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.DateToString(date))
end
```

Ejemplo de llamada: DateToString(2015, 10, 5, 6, 15, 48)
Resultado obtenido: 05/10/2015 6:15:48

NXA.STRINGTODATE

Convierte una cadena en una fecha.
Le pasamos como parámetro una fecha.
Script de ejemplo:

```
function StringToDate(date)
nxa.LogInfo(nxa.StringToDate(date))
end
```

Ejemplo de llamada: StringToDate("2013, 10, 5, 6, 15, 48")
Resultado obtenido: 06:15:48.998

NXA.SETDATE

Modifica una fecha dada.

Le pasamos como parámetro la fecha, que incluye el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function setDate(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.SetDate(date))
end
```

Ejemplo de llamada: SetDate(2015, 10, 5, 6, 15, 48)

ResultadoObtenido: TRUE

NXA.SETTIMEONLY

Modifica una hora dada.

Recibe como parámetro la fecha, que incluye el año, mes, día, hora, minutos y segundos.

Script de ejemplo:

```
function setTimeOnly(year,month,day,hour,minute,second)
local date = nxa.MakeDate(year,month,day,hour,minute,second)
nxa.LogInfo(nxa.SetTimeOnly(date))
end
```

Ejemplo de llamada: SetTimeOnly(2015, 10, 5, 6, 15, 48)

Resultado obtenido: TRUE

NXA.EXTTIME

Devuelve la hora actual con el formato “HH:MM:SS:MMM”

Script de ejemplo:

```
function ExtTime()
nxa.LogInfo(nxa.ExtTime())
end
```

Ejemplo de llamada: ExtTime()

07 FUNCIONES CON BITS

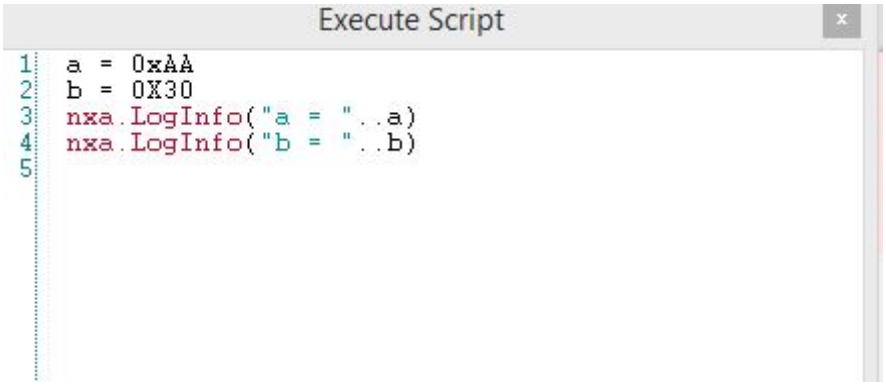
A continuación veremos algunos ejemplos de uso de funciones lógicas y de manejo de bits. Con la función nxa.LogInfo("String".. variableAConsultar), pondremos el string que queramos que se vea, y la variable donde está guardado el resultado a mostrar.

Para ello, deberá ir precedido de dos puntos(..) y seguidamente, la variable a visualizar: Hay que tener en cuenta que todos los valores que nos devuelvan estas funciones son decimales, es decir, nosotros usamos números en hexadecimales y el valor mostrado será en decimal.

PRIMER EJEMPLO

Mediante la función nxa.LogInfo("x = "..x), visualizaremos el valor de la variable x, como veremos a continuación:

Teniendo dos números a y b son número hexadecimales:



Y nuestras variables contienen lo siguiente:

System Messages

Type	Date/Time	Module	Message
INFO	02/06/15 15:50:00.311	LUA_ENGINE	NOT (a) = 170
INFO	02/06/15 15:50:00.311	LUA_ENGINE	NOT (b) = 48
INFO	02/06/15 15:51:02.560	LUA_ENGINE	a = 170
INFO	02/06/15 15:51:02.560	LUA_ENGINE	b = 48

FUNCIÓN AND

La función nxa.And(x,y), devolverá la AND de dos números, siendo **c** un entero en decimal:

Teniendo dos números a y b hexadecimales:

Execute Script

```
1  a=0xAA
2  b=0x30
3  c=nxa.And(a,b)
4  nxa.LogInfo("And(a,b)= "..c)
5
6
7
```

Y el resultado es lo marcado en recuadro negro.

ERROR	15/06/15 16:27:34.864	XDB_ENGINE	SysDatabase OFFLINE. Error :
INFO	15/06/15 16:28:08.950	LUA_ENGINE	And(a,b)= 32

FUNCIÓN OR

La función nxa.OR(x,y), devolverá la OR de los números, siendo **f** un entero en decimal

Teniendo dos números a y b, de tipo entero:

Execute Script

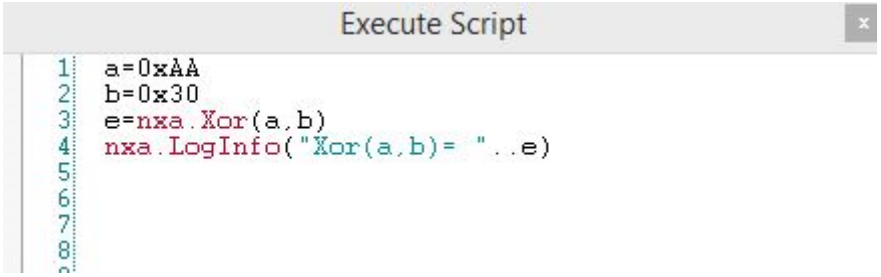
```
1  a=0xAA
2  b=0x30
3  f=nxa.Or(a,b)
4  nxa.LogInfo("Or(a,b)= "..f)
5
6
```

Y el resultado es lo marcado en recuadro negro.

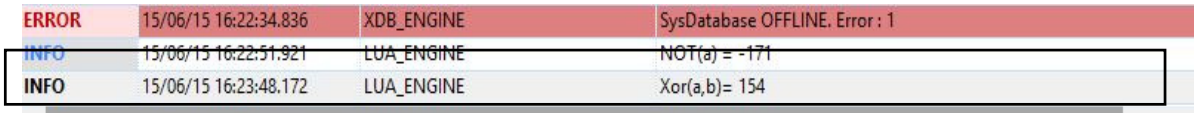
INFO	15/06/15 16:22:51.921	LUA_ENGINE	NOT(a) = -171
INFO	15/06/15 16:23:48.172	LUA_ENGINE	Xor(a,b)= 154
INFO	15/06/15 16:24:46.715	LUA_ENGINE	Or(a,b)= 186

FUNCIÓN XOR

La función nxa.Xor(x,y), devolverá la Xor de dos números, siendo **e** un entero en decimal:
Teniendo dos números a y b hexadecimales:



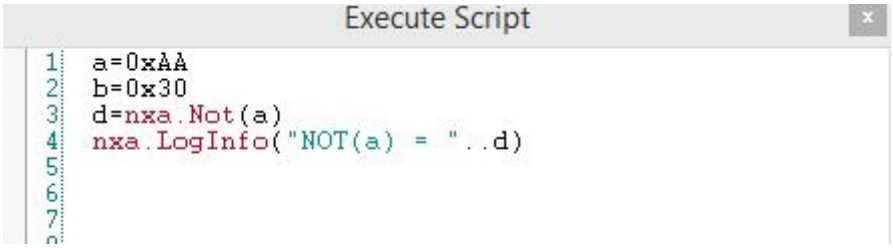
Y el resultado es lo marcado en recuadro negro.



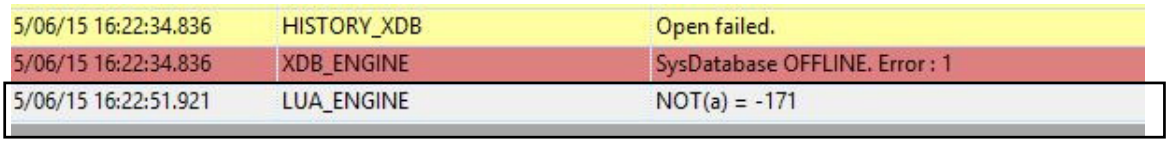
FUNCIÓN NOT

La función nxa.Not(x), devolverá un número negado, siendo **d** un entero en decimal.

Teniendo dos números a y b hexadecimales:



Y el resultado es lo marcado en recuadro negro.



08 FUNCIONES LUA PARA LA EXTRACCIÓN DE DATOS

NXA.LOWBYTE

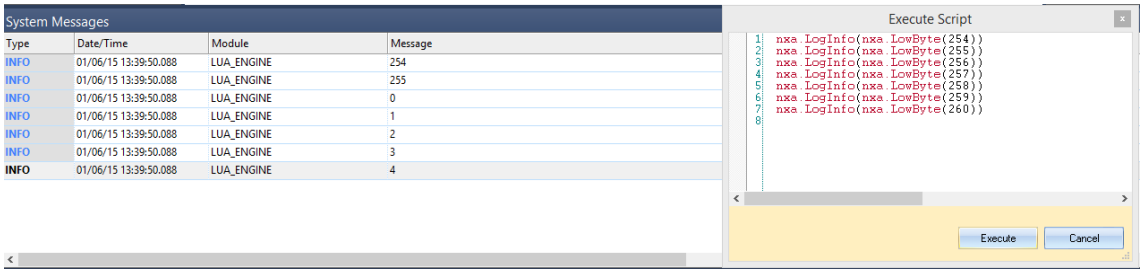
Permite obtener el byte menos representativo del valor de entrada.

Parámetros:

Número: valor de entrada

Devuelve:

Número: byte menos representativo del valor de entrada



Como podemos ver en el ejemplo, en los 2 primeros casos son valores de menos de 256 (valores que se pueden representar con 1 byte), y en el siguiente te devuelve 0, ya que necesita otro bit para representarlo y los primeros 8 bits (el primer byte) son cero. A partir de este vemos que devuelve el byte menos representativo, que sería el valor menos 256 o múltiplos de este.

NXA.HIBYTE

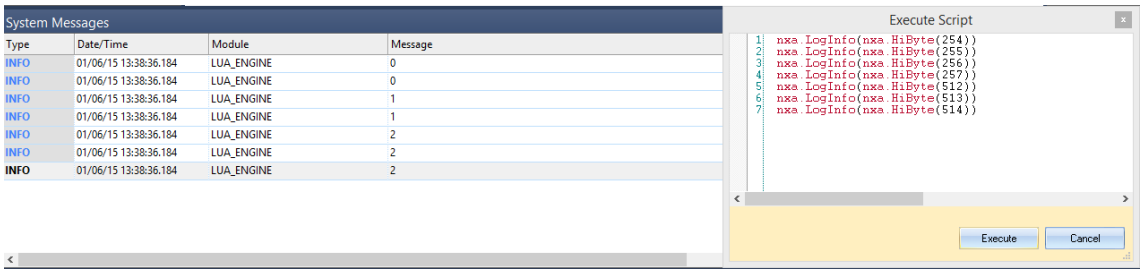
Devuelve el byte más representativo del valor de entrada.

Parámetros:

Número: valor de entrada.

Devuelve:

Número: byte más representativo del valor de entrada.



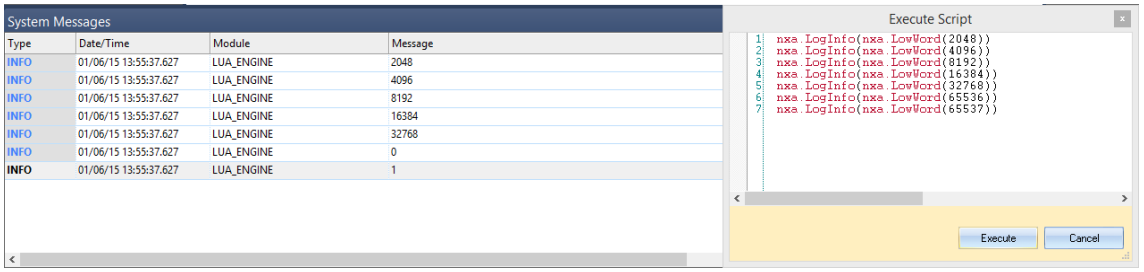
En el ejemplo podemos ver como hasta 256 el valor se representa con 1 solo byte, pero cuando pasa de 256 devuelve 1, ya que el bit que necesita para representarlo ya pertenece al byte más representativo, el cual queremos mostrar. Vemos que ocurre lo mismo al pasar de 512, múltiplo de 256, y se pone a dos.

NXA.LOWWORD

Devuelve la palabra (16 bits) menos representativa dentro del valor de entrada.

Parámetros:
Número: valor de entrada

Devuelve:
Número: la palabra (16 bits) más baja dentro del valor de entrada.



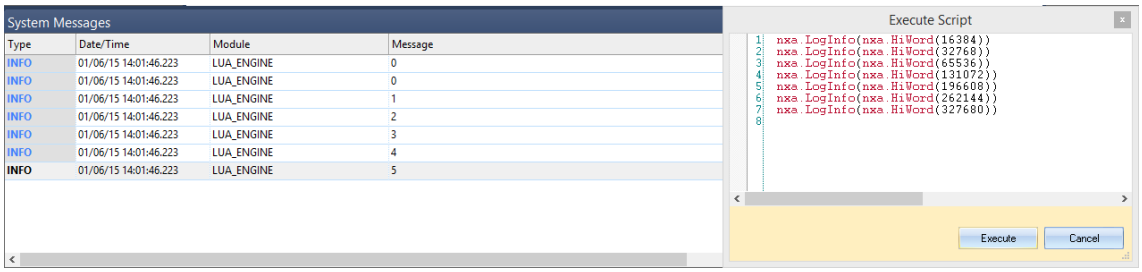
El ejemplo es similar al método **nxa.LowByte**, pero en este caso en vez de un byte es una palabra de 16 bits, por lo que ocurre lo mismo pero a partir del valor 65536. Vemos que con este valor se pone a 0 y a partir de este devuelve el valor de entrada menos 65536 o múltiplos del mismo.

NXA.HIWORD

Devuelve la palabra (16 bits) más significativa del valor de entrada.

Parámetros:
Número: valor de entrada

Devuelve:
Número: Palabra (16 bits) más significativa del valor de entrada.



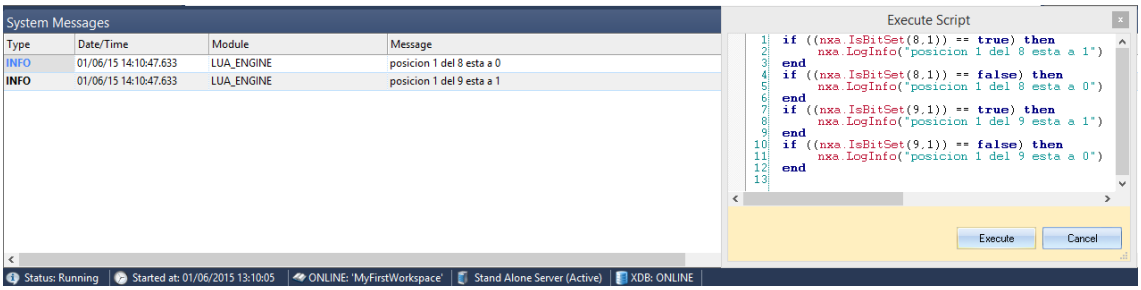
El ejemplo es similar a **nxa.HiByte**, donde vemos que cuando pasa de 65536 devuelve 1, ya que el bit que necesita para representarlo ya pertenece al byte más representativo, el cual queremos mostrar. Vemos que, como se explica el ejemplo de **nxa.HiByte**, va cambiando con los múltiplos de 65536.

NXA.ISBITSET

Comprueba si el bit de una posición del valor de entrada está a 1.

Parametros:
Número: valor de entrada
Número: posición del bit

Devuelve:
Boolean: Devuelve verdadero i el valor del bit está a 1, de lo contrario devuelve 0.



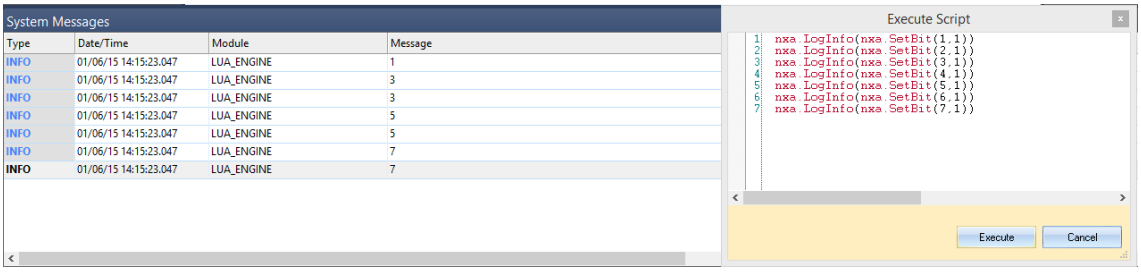
Podemos ver en este sencillo ejemplo donde comprobamos de los valores 8 (1000) y 9 (1001) la posición 1. Como vemos, comprobamos si con el método nos devuelve verdadero o falso, si nos devuelve verdadero nos devuelve "posición 1 del X está a 1", y si nos devuelve falso "posición 1 del X está a 0". Vemos que con 8, al ser el bit en la posición 1 un 0, este nos va a devolver falso, por ello nos devuelve "posición 1 del 8 está a 0", sin embargo, el 9 al ser el bit en la posición 1 un 1, nos devuelve "posición 1 del 9 está a 1".

NXA.SETBIT

Coloca un bit a 1 en una posición del valor de entrada.

Parámetros:
Número: valor de entrada
Número: posición del bit

Devuelve:
Número: resultado al cambiar el bit



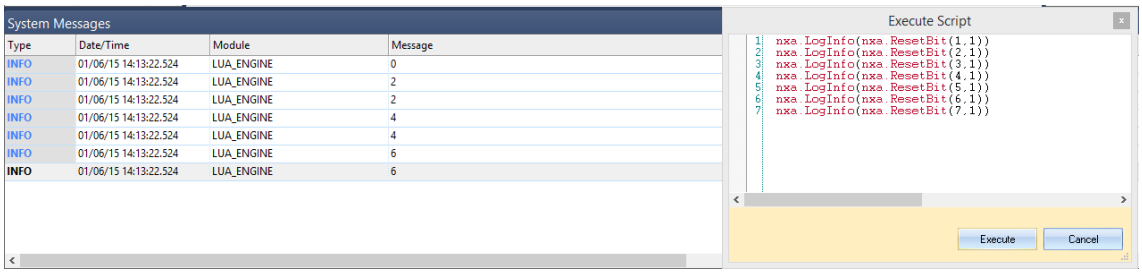
Como podemos ver en este ejemplo, hemos cambiado el bit 1 de los valores del 1 al 7. Como se puede comprobar, solo cambia los valores 2, 4 y 6, y esto sucede porque los valores 1, 3, 5 y 7 ya tienen el bit 1 a 1, y no tienen ningún cambio.

NXA.RESETBIT

Coloca un bit a 0 en una posición del valor de entrada.

Parámetros:
Número: valor de entrada
Número: posición del bit

Devuelve:
Número: resultado al cambiar el bit

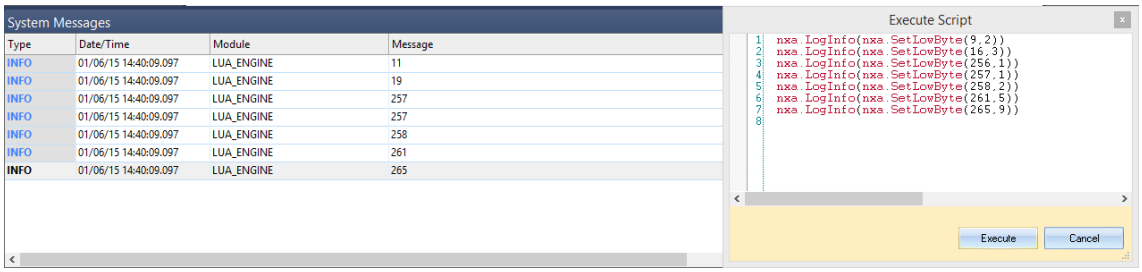


El ejemplo es similar el del método **nxa.SetBit**, pero en este caso en vez de poner el bit a 1, lo ponemos a 0. Vemos que en este caso los que cambian son los valores 1, 3, 5, y 7 ya que son los que tienen el bit 1 a 1 y los pone a 0. Los valores 2,4 y 6 se quedan igual ya que el bit 1 ya está a 0.

NXA.SETLOWBYTE

Aplica la operación lógica OR del byte menos representativo del valor de entrada con otro byte de entrada.

Parámetros:
Número: valor de entrada.
Número: valor del byte para aplicar al valor de entrada.
Devuelve:
Número: resultado del método OR de los dos valores.

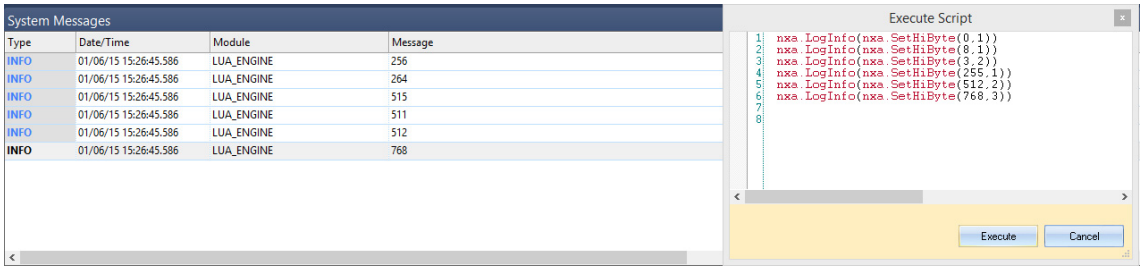


Como podemos ver en el ejemplo, al principio puede parecer una suma de valores ya que 9 (1000) al aplicarle este método con 2(10) te sale la suma de ambos, 11(1010). Pero si vemos en los siguientes ejemplos podemos comprobar que si le aplicamos un byte X, a un número cuyo byte más bajo es este mismo valor, el número no cambia. Hace hace un OR de los dos bytes, y si son iguales no va a cambiar.

NXA.SETHIBYTE

Aplica la operación lógica OR del byte más representativo del valor de entrada con otro byte de entrada.

Parámetros:
Número: valor de entrada.
Número: valor del byte para aplicar al valor de entrada.
Devuelve:
Número: resultado del método OR de los dos valores.

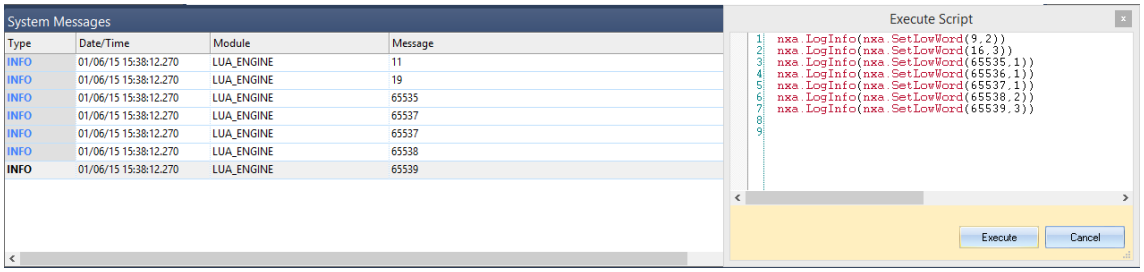


Como vemos en el ejemplo, el método es igual que **nxa.SetLowByte**, pero en este caso se hace con el byte más representativo del valor. Como podemos ver en el ejemplo, en el caso de 512 (1000000000), al aplicarle el método con el 2 (10), el byte más significativo es 10, por lo que el valor no cambia.

NXA.SETLOWWORD

Aplica la operación lógica OR de la palabra (16 bits) menos representativo del valor de entrada con otro byte de entrada.

Parámetros:
Número: valor de entrada.
Número: valor de la palabra para aplicar al valor de entrada.
Devuelve:
Número: resultado del método OR de los dos valores.

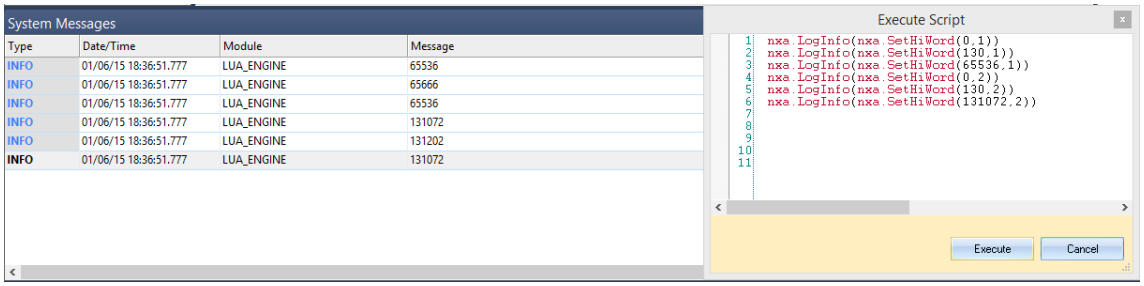


Como podemos ver, el ejemplo es similar a **nxa.SetLowByte**, solo que en vez de 1 byte (8 bits) es una palabra (16 bits). De hecho podemos ver que hemos usado los mismos ejemplos al principio, y puede parecer una suma de valores ya que 9 (1000) al aplicarle este método con 2(10) te sale la suma de ambos, 11(1010).
Si vemos en los siguientes ejemplos podemos comprobar que si le aplicamos una palabra X, a un número cuya palabra menos significativa es este mismo valor, el número no cambia. Hace un OR de las dos palabras al igual que **nxa.SetLowByte** con los bytes.

NXA.SETHIWORD

Aplica la operación lógica OR de la palabra (16 bits) más representativo del valor de entrada con otro byte de entrada.

Parámetros:
Número: valor de entrada.
Número: valor de la palabra para aplicar al valor de entrada.
Devuelve:
Número: resultado del método OR de los dos valores.



Como vemos en el ejemplo, el método es igual que **nxa.SetLowWord**, pero en este caso se hace con la palabra más representativa del valor. Como podemos ver en el ejemplo, en el caso de 131072 (100…..00), al aplicarle el método con el 2 (10), la palabra más significativa es 10, por lo que el valor no cambia.

09 LUA Y XCON

En este apartado, vamos a usar los scripts de LUA, para crear comunicaciones de cara al exterior. Se podrán crear diferentes tipos, como por ejemplo crear sockets para conexiones TCP o UDP, COM, RSS, o poder crear un HTML o enviar emails a diferentes personas.

A lo largo de este documento se irá detallando los distintos métodos que hay, el código para poder ejecutarlo y si fuese posible, indicar un ejemplo de uso. Por lo que vamos a empezar a desarrollar los diferentes métodos.

Antes de nada, hay que destacar que el código que vamos a indicar, a continuación es el TEORICO, siguiendo el tutorial, pero ese código no sirve, antes de poder ejecutar este comando, hay que hacer una estructura completa, que es el comando

IntializeSysXCON()

Sin la estructura que hace ese comando, primeramente no podemos ver si hemos abierto los sockets, y no lo podremos ver en el ItemTree, por lo que cuando se terminen de crear los distintos sockets, explicaremos la estructura que se sigue verdaderamente para que el sistema funcione correctamente.

XCON.CREATE

Este método de creación de Sockets mediante LUA, es el más genérico, ya que permite, mediante un parámetro poder crear, cualquier tipo de sockets indicado anteriormente. Este método requiere de varios parámetros de entrada para poder crearse correctamente y posteriormente, realizar acciones sobre el socket correspondiente. Los parámetros de entrada son:

- String -> Manipulador que identifica la conexión mediante LUA
- Number -> Mediante un entero, se indica el tipo de socket que se va a crear, pudiendo ser los siguientes valores:
    - 1 para crear nxa.xcon.UDP
    - 2 para crear nxa.xcon.TCPClient
    - 4 para crear nxa.xcon.HTTP
    - 5 para crear nxa.xcon.RSS

- 6 para crear nxa.xcon.COM

- String -> La IP local, si se dejase vacío lo detecta automáticamente
- Number -> El puerto local para esta conexión, si se dejase el valor a 0, el sistema lo genera automáticamente
- String -> La IP destino a la que conectarse
- Number -> Puerto destino a la que conectarse

El código de uso sería el siguiente:

```
function OnInitEvent()
  --InitializeSysXCON()
  xcon.Create("__SysUDP",1,"",9091,"",9092)

end
```

Se ha incluido en OnInitEvent para que se ejecute cuando arranque el sistema. De esta forma tendremos un socket UDP abierto con esos datos.

XCON.CREATEUDP

Con este método vamos a crear un socket UDP. Los parámetros de entrada son similares al anterior caso, lo único que cambia, es que no hay que indicarle el número, quedando por tanto así:

- String -> Manipulador que identifica la conexión mediante LUA
- String -> La IP local, si se dejase vacío lo detecta automáticamente
- Number -> El puerto local para esta conexión, si se dejase el valor a 0, el sistema lo genera automáticamente
- String -> La IP destino a la que conectarse
- Number -> Puerto destino a la que conectarse

```
function OnInitEvent()
  --InitializeSysXCON()
    xcon.CreateUDP("__Sys_UDP","",9091,"127.0.0.1",9092)
end
```

XCON.CREATETCP

Con este método vamos a crear un socket TCP. Los parámetros de entrada son similares al anterior caso:

- String -> Manipulador que identifica la conexión mediante LUA
- String -> La IP local, si se dejase vacío lo detecta automáticamente
- Number -> El puerto local para esta conexión, si se dejase el valor a 0, el sistema lo genera automáticamente
- String -> La IP destino a la que conectarse
- Number -> Puerto destino a la que conectarse

```
function OnInitEvent()
  --InitializeSysXCON()
    xcon.CreateTCP("__Sys_TCP","",9091,"127.0.0.1",9092)
end
```

XCON.CREATECOM

Con este método vamos a crear una conexión Serie con LUA. Los parámetros a utilizar son los siguientes:

- String -> Manipulador que identifica al puerto Serie
- String -> Nombre del COM del sistema
- Number -> Indica la velocidad de transmisión (Baudrate)
- Number -> La cantidad de bits de datos del puerto serie
- Number -> La paridad usada
- Number -> La cantidad de bits de stop
- Number -> El protocolo de handshake utilizado
- Number -> El evento de carácter usado

```
function OnInitEvent()
  InitializeSysXCON()
  xcon.CreateCOM("__SysCOM","COM1",9600,8,0,0,0,0)
end
```

XCON.CREATEHTTP

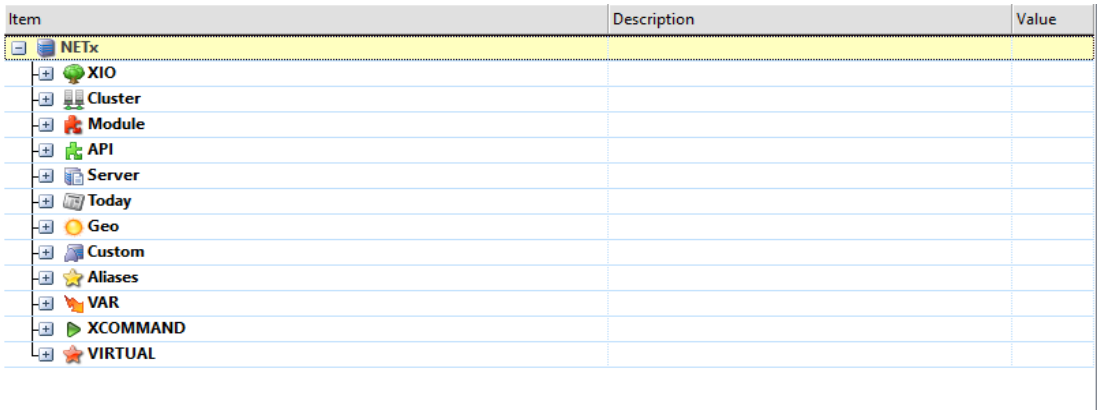
Con este método, vamos a poder crear una conexión HTTP con LUA. Para ello, solo es necesario el String correspondiente al manipulador de la conexión HTTP

XCON.CREATERSS

Con este método, vamos a poder crear un RSS con LUA, siguiente con el mismo caso que antes, solo es necesario el String correspondiente al manipulador del RSS

ACLARACIÓN SOBRE LA CREACIÓN DE LOS DIFERENTES SOCKETS

Como hemos indicado anteriormente, esto por sí solo no funciona. Además si lo intentamos crear así podemos ver lo siguiente en el árbol de Ítems:



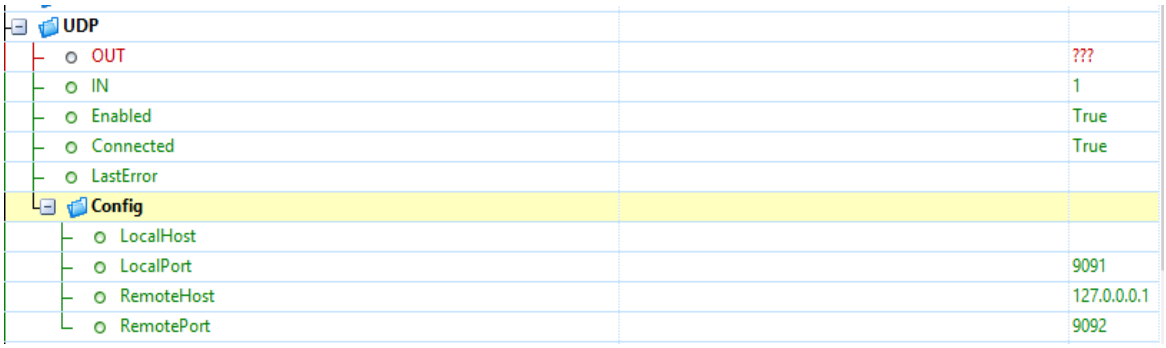
Como vemos, en este árbol de elementos, no aparece la rama XCON. Esto es debido a que no se ha creado la estructura para las comunicaciones que son las que se cargan con el elemento que indicábamos anteriormente que estaba conectado. Un ejemplo de una estructura creada para UDP sería la siguiente:

```
71  -- UDP Subsystem
72  --------------------------------------------------------------------------------
73  function Create__SysUDP()
74    local CfgCngFunc = "__SysUDPConfigChanged()"
75    local SndFunc    = "__SysUDPSend()"
76    local EnbFunc    = "__SysUDPEnabled()"
77    sysUDP           = {}
78    sysUDP[IsCON]    = false
79    sysUDP[NM]       = "__SysUDP"
80    sysUDP[DP]       = "UDP"
81
82    sysUDP[OUT] = nxa.AddSysCustomItem(OUT, EptDESC, nxa.access.All,      nxa.type.String,
83    sysUDP[IN]  = nxa.AddSysCustomItem(IN,  EptDESC, nxa.access.Readable, nxa.type.String,
84    sysUDP[ENB] = nxa.AddSysCustomItem(ENB, EptDESC, nxa.access.All,      nxa.type.Boolean,
85    sysUDP[CON] = nxa.AddSysCustomItem(CON, EptDESC, nxa.access.Readable, nxa.type.Boolean,
86    sysUDP[LE]  = nxa.AddSysCustomItem(LE,  EptDESC, nxa.access.All,      nxa.type.String,
87
88    sysUDP[LH]  = nxa.AddExtSysCustomItem(LH, EptDESC, nxa.access.All, nxa.type.String,  tr
89    sysUDP[LP]  = nxa.AddExtSysCustomItem(LP, EptDESC, nxa.access.All, nxa.type.Integer, tr
90    sysUDP[RH]  = nxa.AddExtSysCustomItem(RH, EptDESC, nxa.access.All, nxa.type.String,  tr
91    sysUDP[RP]  = nxa.AddExtSysCustomItem(RP, EptDESC, nxa.access.All, nxa.type.Integer, tr
92
93    nxa.SetValue(sysUDP[ENB], false)
94    nxa.SetValue(sysUDP[CON], false)
95    nxa.SetValue(sysUDP[LE], "")
96    nxa.SetValue(sysUDP[LH], "")
97    nxa.SetValue(sysUDP[LP], 9091)
98    nxa.SetValue(sysUDP[RH], "")
99    nxa.SetValue(sysUDP[RP], 9092)
100
101   nxa.AddScriptTask(sysUDP[LH],  "", true, true, true, 100, CfgCngFunc)
102   nxa.AddScriptTask(sysUDP[LP],  "", true, true, true, 100, CfgCngFunc)
103   nxa.AddScriptTask(sysUDP[RH],  "", true, true, true, 100, CfgCngFunc)
104   nxa.AddScriptTask(sysUDP[RP],  "", true, true, true, 100, CfgCngFunc)
105   nxa.AddScriptTask(sysUDP[OUT], "", true, true, true, 100, SndFunc)
106   nxa.AddScriptTask(sysUDP[ENB], "", true, true, true,   0, EnbFunc)
107 end
```

Como se ve, crea una estructura inicializada con unos valores por defectos, y sobre los campos de esta estructura es donde podemos trabajar. Es decir, en este caso por ejemplo, sysUDP. Un ejemplo puede ser el siguiente:

```
function OnInitEvent()

  InitializeSysXCON()
  nxa.SetValue(sysUDP[ENB], true)
  nxa.SetValue(sysUDP[CON], true)
  nxa.SetValue(sysUDP[RH], "127.0.0.0.1")

end
```

Partiendo de la imagen de la estructura, hemos hecho, que cuando se cree, modifique la IP destino, y active el enable y el connected para poder realizar comunicaciones, y para probarlo hemos enviado un mensaje mediante UDP, resumiéndose en lo siguiente:



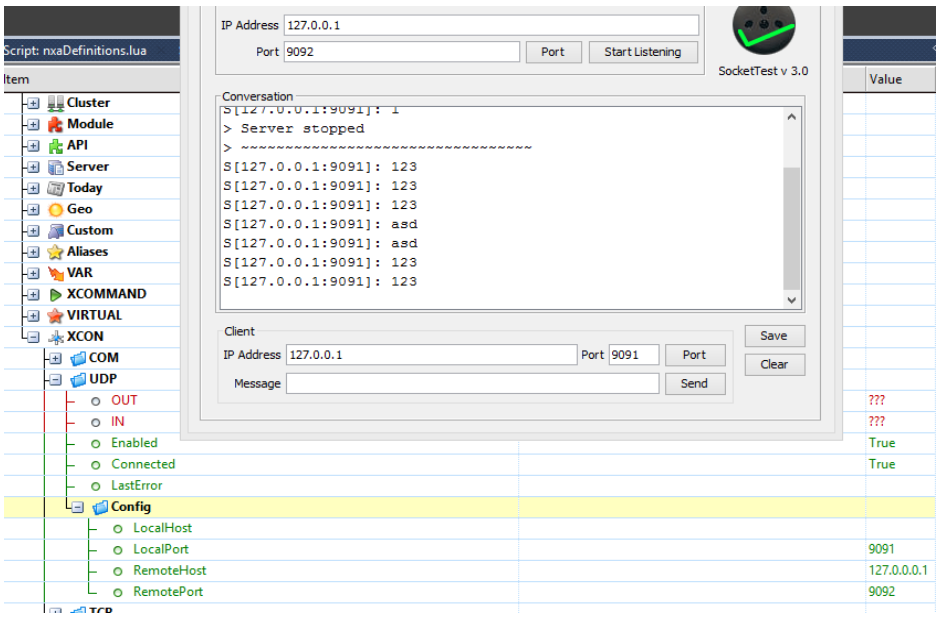
Como vemos lo hemos activado correctamente y hemos recibido un mensaje, en este caso el número 1 mediante una aplicación UDP conectada al puerto 9091.
Por lo que partiendo de este ejemplo, vamos a seguir continuando con el resto de la documentación.

XCON.CLOSE

Con este método, lo que vamos a hacer es cerrar la conexión, por lo que de parámetro solo necesita un String que se corresponda con el manipulador. En este caso, vamos a mantener los valores de antes, pero, aun manteniéndose en enable y connected, es imposible recibir ningún mensaje.

```
-----------------------------------------
function OnInitEvent()

   InitializeSysXCON()
   nxa.SetValue(sysUDP[RH], "127.0.0.0.1")
   nxa.SetValue(sysUDP[ENB], true)
   nxa.SetValue(sysUDP[CON], true)
   xcon.Close(sysUDP[NM])


end
-----------------------------------------
```

Obteniendo lo siguiente:



Vemos, que nunca se actualiza el valor de entrada, por muchos intentos que le hagamos.

XCON.SEND

Con este método lo que pretendemos es enviar un mensaje a través de una conexión establecida, siguiendo la misma pauta anterior, lo vamos a comprobar con UDP. Para ello, son necesarios los siguientes parámetros:

- String -> Manipulador del socket por el que enviaremos el mensaje
- String -> Mensaje a enviar
- Number -> Cantidad de caracteres a enviar

Además, este método devuelve un parámetro:

- Boolean -> True si el servidor está inicializado o False si no lo está

Se podría ejecutar de la siguiente forma:

```
function OnInitEvent()

  InitializeSysXCON()
  xcon.Send(sysUDP[NM],"Holaa mundooo",6)

end
```

XCON.SENTTEXT

Esta función sirve para enviar un texto a través de una conexión establecida. Lo vamos a comprobar con UDP. Para ello, son necesarios los siguientes parámetros:

- String -> Manipulador del socket por el que enviaremos el mensaje
- String -> Mensaje a enviar

Además, este método devuelve un parámetro:

- Boolean -> True si el servidor está inicializado o False si no lo está

Se podría ejecutar de la siguiente forma:

```
function __SysUDPSend()
  if (sysUDP[IsCON] == false) then
    nxa.SetValue(sysUDP[LE], "Not connected")
  else
    xcon.SendText(sysUDP[NM], nxa.InputValue())
  end
end
```

XCON.SENDTO

Esta función, permite enviar un mensaje a un socket destino. Se necesitan los siguientes parámetros:

- String -> Manipulador de la conexión a usar
- String -> Mensaje a enviar
- Number -> Cantidad de caracteres a enviar
- String -> IP destino
- Number -> Puerto destino

Además, este método devuelve un parámetro:

- Boolean -> True si el servidor está inicializado o False si no lo está

XCON.SENDTEXTTO

Esta función, permite enviar un mensaje a un socket destino. Se necesitan los siguientes parámetros:

- String -> Manipulador de la conexión a usar
- String -> Mensaje a enviar
- String -> IP destino
- Number -> Puerto destino

Además, este método devuelve un parámetro:

- Boolean -> True si el servidor está inicializado o False si no lo está

```
function OnInitEvent()
  InitializeSysXCON()
  xcon.SendTextTo(sysUDP[NM],"Hola","127.0.0.1",9092)
end
```

XCON.ISCONNECTED

Esta función, nos permite saber el estado de la conexión, por lo que le pasamos el manejador de la conexión y nos devuelve un booleano con el estado de la conexión.

XCON.SENDEMAILTO

Este método se usa para enviar un email a través de LUA. Para los 4 tipos de email, es necesario haber configurado previamente, correctamente el servicio de correo en la configuración del sistema.

Se necesitan los siguientes parámetros:

- String -> correo destino
- String -> Asunto del correo
- String -> Cuerpo del correo

Con el siguiente código, se puede enviar un correo a un destino:

```
function OnInitEvent()
  InitializeSysXCON()
  xcon.SendEmailTo("luzxor@gmail.com","Prueba 1", "email")
end
```

XCON.SENDADMINEMAIL

Este método se usa para enviar un email al administrador del sistema a través de LUA. Se necesitan los siguientes parámetros:

- String -> Asunto del correo
- String -> Cuerpo del correo

XCON.SENDSYSTEMEMAIL

Este método se usa para enviar un email al correo del sistema a través de LUA. Se necesitan los siguientes parámetros:

- String -> Asunto del correo
- String -> Cuerpo del correo

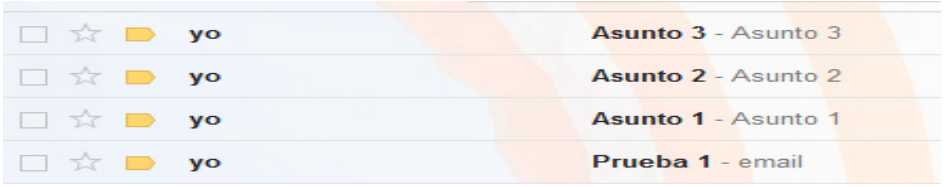
XCON.SENDUSEREMAIL

Este método se usa para enviar un email a un usuario previamente configurado en el sistema a través de LUA. Se necesitan los siguientes parámetros:

- String -> Asunto del correo
- String -> Cuerpo del correo

Como resultado de los tres últimos casos, se puede ver lo siguiente:

```
function OnInitEvent()

  InitializeSysXCON()

  xcon.SendAdminEmail("Asunto 1","Asunto 1")
  xcon.SendSystemEmail("Asunto 2","Asunto 2")
  xcon.SendUserEmail("Asunto 3","Asunto 3")

end
```

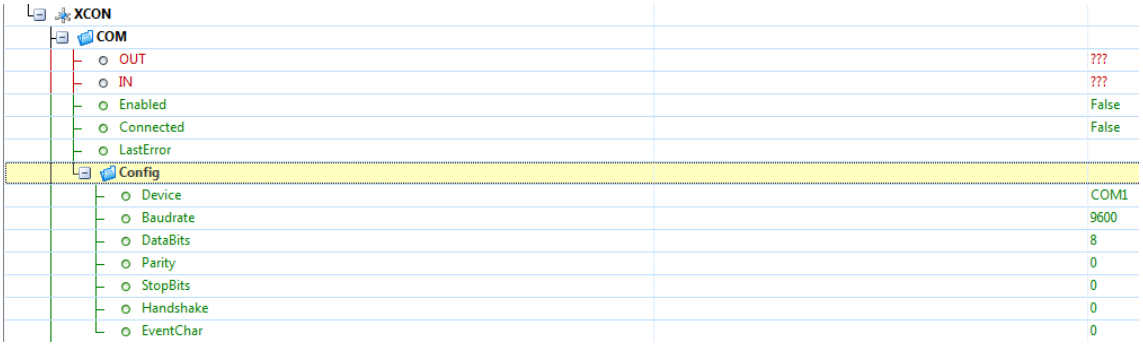


10 XCON: INTERFAZ COM

La COM INTERFACE es una interfaz de comunicaciones usada para establecer una comunicación a través de una interfaz serie. Se compone de *5 entradas de comunicación* y una rama de *configuración*. Esta interfaz debe ser habilitada antes de poder ser utilizada.

DESCRIPCIÓN DE LOS ÍTEMS EN INTERFAZ GRÁFICA XCON.

La siguiente gráfica muestra cada uno de los ítems que podemos encontrar en la interfaz gráfica COM.



A continuación procederemos a especificar la funcionalidad y características de cada uno de los ítems:

10.1 PARÁMETROS DE COMUNICACIÓN:

OUT

Data Type: STRING

Default value: None

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.OUT

Item ID: NETx.XCON.COM.OUT

Description: El buffer de bytes para enviar a la interfaz COM se escribe en este ítem.

IN

Data Type: STRING

Default value: None

Access Rights: Read only

Standard Path: NETx.XCON.<COM#>.IN

Item ID: NETx.XCON.COM.IN

Description: El buffer de bytes que recogemos desde la interfaz COM será escrito en este item.

ENABLE

Data Type: BOOL

Default value: 0 (False)

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Enabled

Item ID: NETx.XCON.COM.Enabled

Description: A través de este item habilitamos o deshabilitamos la interfaz COM. Este item es el encargado tanto de crear una conexión como de eliminarla.

CONNECTED

Data Type: BOOL

Default value: 0 (False)

Access Rights: Read only

Standard Path: NETx.XCON.<COM#>.Connected

Item ID: NETx.XCON.COM.Connected

Description: Muestra si la conexión se ha establecido o no.

LAST ERROR

Data Type: STRING

Default value: empty string

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.LastError

Item ID: NETx.XCON.COM.LastError

Description: Si ocurre un error, este será escrito en este item.

10.2 PARÁMETROS DE CONFIGURACIÓN

DEVICE

Data Type: STRING

Default value: "COM1"

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.Device

Item ID: NETx.XCON.COM.Config.Device

Description: Determina el Puerto COM que será direccionado en la conexión.

BAUDRATE

Data Type: INT4

Default value: 9600

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.Baudrate

Item ID: NETx.XCON.COM.Config.Baudrate

Description: Este item especifica el BaudRate de la interfaz de comunicaciones.

La siguiente table muesta los valores de baudrate estándar:

110	300	600
1200	2400	4800
9600	14400	19200
28800	38400	56000
57600	115200	

DATABITS

Data Type: INT4

Default value: 8

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.DataBits

Item ID: NETx.XCON.COM.Config.DataBits

Description: Determina los databits de cada carácter, los valores 5,7 y 8 son los mas communes mientras que 6 y 9 son raramente usados.

PARITY

Data Type: INT4

Default value: 0 (False)

Access Rights: Read only

Standard Path: NETx.XCON.<COM#>.Config.Parity

Item ID: NETx.XCON.COM.Config.Parity

Description: Especifica la paridad del mensaje. Si paridad es distinto de cero, se añadirá un bit de paridad con la comprobación especificada por ese número; es decir, si parity=1 se añadirá un último bit de paridad impar y se comprobará que es correcta con el fin de establecer mecanismos de seguridad y comprobación del envío y recepción correcta de mensajes.

La siguiente tabla muestra los distintos métodos de paridad disponibles:

Value	Meaning
0	No parity
1	Odd parity
2	Even parity
3	Mark parity
4	Space parity

STOPBITS

Data Type: INT4

Default value: 0

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.StopBits

Item ID: NETx.XCON.COM.Config.StopBits

Description: Determina el número de bits de parade al final de cada carácter, es usado para detector el final de cada carácter para recibir y resincronizar. Los dispositivos electromecánicos lentos pueden demander hasta dos bits de parade.

La siguiente tabla muestra los bits de parada permitidos:

Value	Meaning
0	1.0 stop bits
1	1.5 stop bits
2	2.0 stop bits

HANDSHAKE

Data Type: INT4

Default value: 0

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.Handshake

Item ID: NETx.XCON.COM.Config.Handshake

Description: Control de flujo por handshake. Valor a 1 para activarlo.

EVENTCHAR

Data Type: INT4

Default value: 0

Access Rights: Read and Write

Standard Path: NETx.XCON.<COM#>.Config.EventChar

Item ID: NETx.XCON.COM.Config.EventChar

Description: Determina el carácter a usar para indicar el final de una línea (p.ej. CHR(10) para Linefeed).

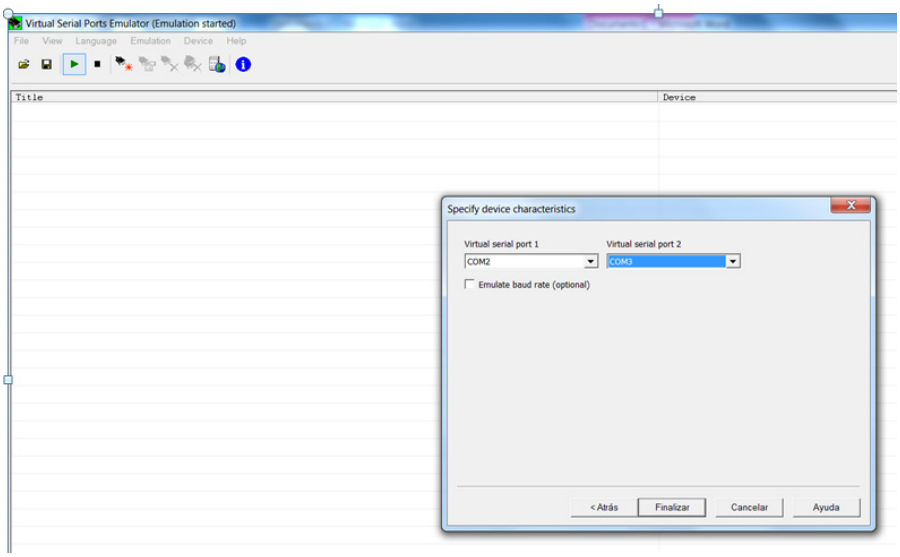
10.3 EJEMPLO DE PRUEBA

Para mostrar la funcionalidad del sistema, a continuación se desarrollará un ejemplo a través de una serie de pasos en el que mostraremos la potencialidad del COM INTERFACE.

CONFIGURACIÓN DE LAS COMUNICACIONES

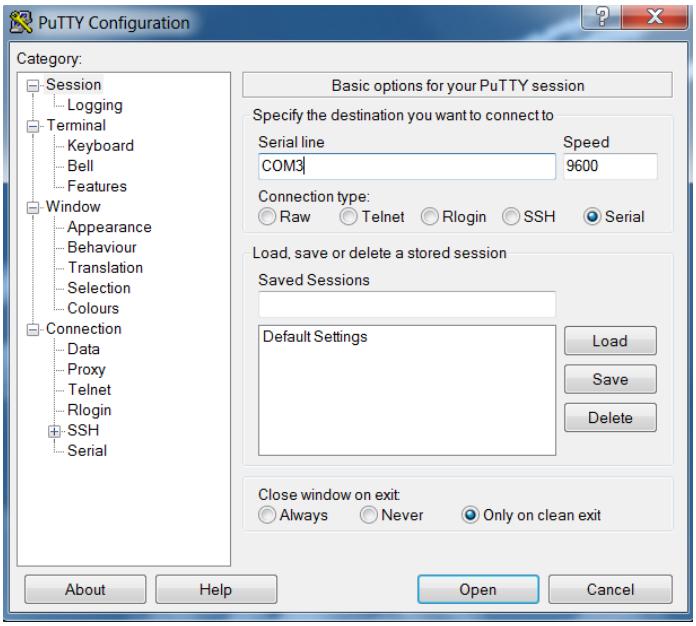
Para emparejar dos puertos de comunicación serie dentro del mismo equipo utilizaremos un emulador de puertos virtuales como es el Virtual Serial Ports Emulator (VSPE). De este modo conseguimos una conexión virtual entre dos puertos serie dentro de un mismo equipo.

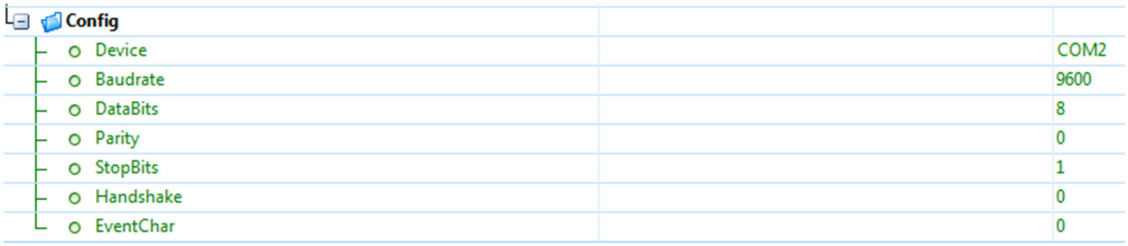
Con el VSPE emparejaremos los puertos COM2 y COM3 de nuestro equipo de forma que se establezca un enlace virtual entre los dos puertos actuando como si estuvieran vinculados físicamente.



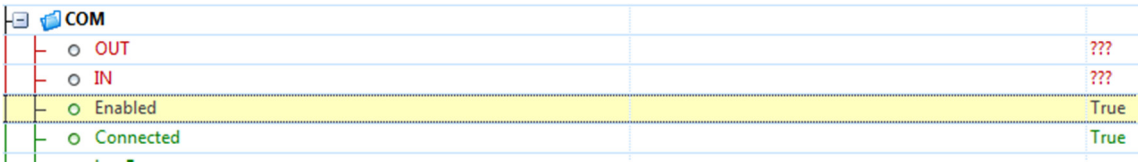
Como cliente serie utilizaremos un software cliente de comunicación como puede ser Putty.

En este punto del ejemplo enlazaremos el puerto COM2 a COM de BMS Server y COM3 al cliente Putty. Los parámetros de comunicación son los siguientes: 9600,8,N,1 (Baudrate, Databits, Parity, StopBits).





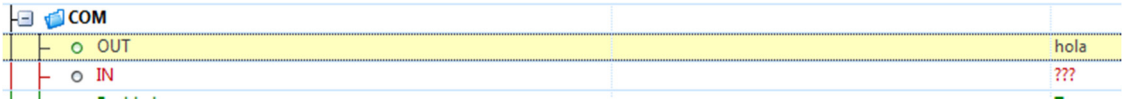
Una vez están enlazados los puertos de comunicación y configurados los parámetros de comunicación procederemos a realizar un sencillo envío de mensajes entre dichos puertos para ilustrar la funcionalidad del COM INTERFACE.

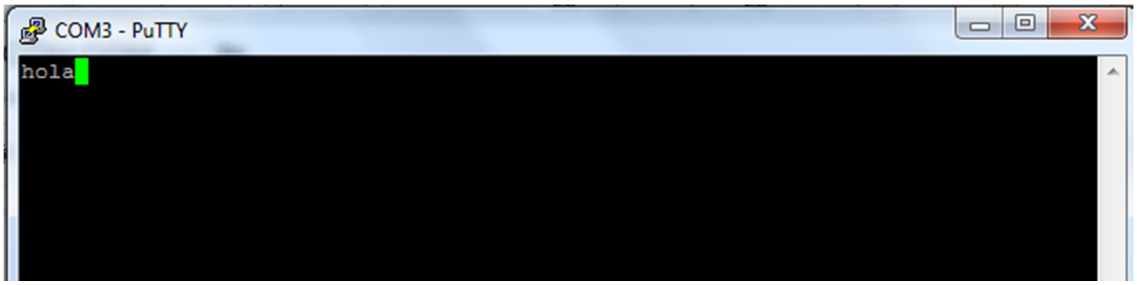


ENVÍO DEL MENSAJE DESDE EL BMS SERVER.

Importante: Antes de comenzar a enviar mensajes, será necesario habilitar el item Enable escribiendo 1 sobre el ítem correspondiente.

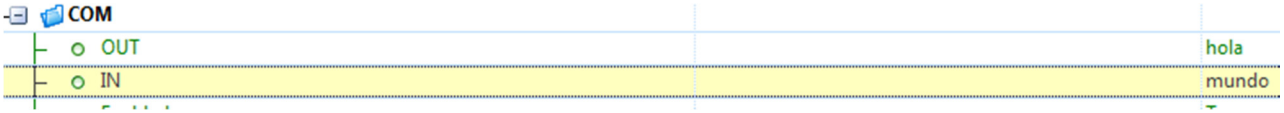
Para el envío de mensajes, basta con escribir sobre el campo OUT un valor de salida, de modo que si escribimos "hola", en el terminal del cliente nos aparecerá el mensaje.





ENVÍO DEL MENSAJE DESDE EL TERMINAL CLIENTE (PUTTY).

El envío de mensajes de cliente se realiza tan sólo con introducir el mensaje por el terminal, de forma que si escribimos "mundo" por el terminal, en la casilla "IN" de COM deberá aparecer el mensaje.



10.4 MANEJO DE FUNCIONES Y CREACIÓN DE SCRIPTS EN LUA

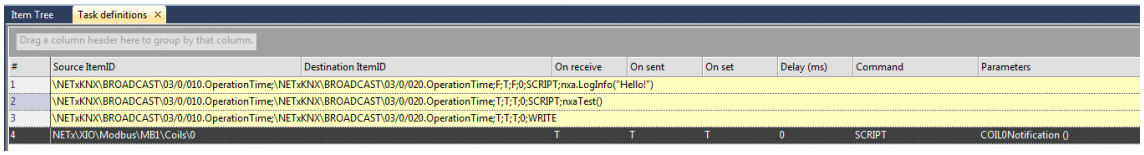
El objetivo del uso de scripts en BMS es el de asignar a una tarea o evento una serie de acciones pudiendo ser más o menos complejas. De esta forma se puede aprovechar todas sus funcionalidades de un modo más personalizado.

En este apartado, se va a asociar un script a una tarea determinada, a modo de ejemplo. Para ello imaginemos una situación en la que tenemos un dispositivo modbus cuya funcionalidad es la de una alarma en una puerta de una máquina frigorífica.

El dispositivo tendría un registro de configuración (no se explicará al detalle en este ejemplo) donde se podrían definir parámetros de temporización y temperatura, de forma que si se deja la puerta abierta y pasa el tiempo programado, se disparará la alarma, cambiando el estado de un coil determinado.

PROCEDIMIENTO

En primer lugar, debemos definir la tarea específica, en nuestro caso se reproduce el script cuando se modifica el valor del coil tanto en envío como en recepción, la función que se ejecuta es COIL0Notification().



A continuación, debemos crear la función **COIL0Notification()**, la cual será la encargada de enviar la notificación a través del puerto serie de comunicaciones.

Para ello, podemos crear la función en un archivo ya existente o crear un archivo "*.lua" nuevo en la misma ruta que los scripts del sistema que por defecto se encuentran en:

C:\Program Files (x86)\NETxAutomation\NETx.BMS.Server.2.0\Workspaces\Workspace\ScriptFiles

En nuestro caso se ha creado un archivo .lua nuevo llamado *nxaCOMExample.lua*

Importante: Recuerda que al crear un nuevo archivo de script, debemos añadirlo en *nxaDefinitions.lua* de la siguiente forma: *require "nxaCOMExample"* La siguiente imagen muestra cómo debería estar escrito y su localización en el archivo de definiciones.

Item Tree | Task definitions | Script: nxaDefinitions.lua*

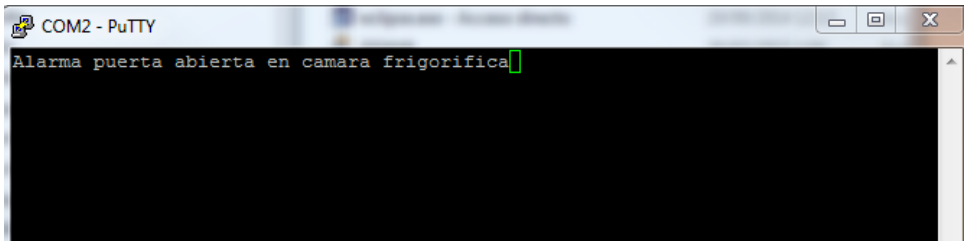
```
--[[============================================================================
company:      NETxAutomation
author:       Paul Furtak
date:         03.12.2013
version:      BMS Server 2.0.7200
filename:     nxaDefinitions.lua
description:  main scripting file of the NETx BMS Server 2.0
------------------------------------------------------------------------------
history:
24.02.2012 - new
02.12.2013 - added libraries nxaBaseFunctions and nxaLogicFunctions
18.09.2014 - added new client events
============================================================================
IMPORTANT:
USE OF SCRIPT ENGINE FUNCTIONALITY IS AT YOUR OWN RISK. BE CAREFULLY WITH IT.
============================================================================]]
package.path = nxa.ScriptFilesPath() .. "\\?.lua"

----------------------------------------------------------------------------
-- require block - additional scripts are included here
----------------------------------------------------------------------------
require "nxaBaseFunctions"
require "nxaLogicFunctions"
require "nxaSystemXCON"
require "nxaHotelFunctions"
------------------------------------------------------------------------------
--require "nxaExample"
require "nxaCOMExample"
----------------------------------------------------------------------------
-- OnInitEvent() - function is called on initialization of the server process.
--    Item Tree is not yet fully initialized. Virtual/custom items can be added
----------------------------------------------------------------------------
function OnInitEvent()
  InitializeSysXCON()

  -- Add your custom items here
  -- nxa.AddCustomItem("MyItem1", "Custom Item 1", nxa.access.All, nxa.type.Real, "/", "BuldingA", "Flc
end
------------------------------------------------------------------------------
```

Una vez hecho esto, procedemos a crear la función. Aquí se configura el puerto cada vez que se produce el evento, de esta forma nos aseguramos que la transmisión se realizará correctamente.

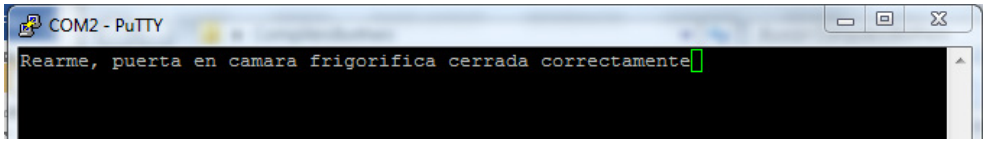
Item Tree | Task definitions | Script: nxaDefinitions.lua* | Script: nxaCOMExample.lua

```

--nxa.SetValue(itemID, value [,delay_in_ms])
--nxa.GetValue(itemID [,defaultvalue]) As value
--nxa.WriteValue(itemID, value [,delay_in_ms])
--nxa.ReadValue(itemID [,delay in ms])


function COIL0Notification ()
--nxa.SetValue("NETx.XCON.<COM3>.OUT","MUNDO")
--configura comunicaciones

nxa.SetValue("NETx.XCON.COM.Config.Device","COM3")
nxa.SetValue("NETx.XCON.COM.Config.Baudrate","9600")
nxa.SetValue("NETx.XCON.COM.Config.DataBits","8")
nxa.SetValue("NETx.XCON.COM.Config.Parity","0")
nxa.SetValue("NETx.XCON.COM.Config.StopBits","1")
--resto de parámetros por defecto

--Habilita comunicaciones
nxa.SetValue("NETx.XCON.COM.Enabled","1")

--Envia mensaje COIL1

if (nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Coils\\0")==true) then--si se activa la alarma...
    nxa.SetValue("NETx.XCON.COM.OUT","Alarma puerta abierta en camara frigorifica")--COIL0=1
else nxa.SetValue("NETx.XCON.COM.OUT","Rearme, puerta en camara frigorifica cerrada correctamente")--COIL0=0
    end
end
```

Con una configuración como la del ejemplo anterior, los mensajes que se mostrarían serían como sigue:



Si se cierra la puerta, se vuelve a cambiar el estado del coil, con lo cual se envía lo siguiente:



SCRIPT DE EJEMPLO

```
function COIL0Notification ()

nxa.SetValue(Item ID,Valor)

nxa.GetValue(Item ID)

--configura comunicaciones

nxa.SetValue("NETx.XCON.COM.Config.Device","COM3")

nxa.SetValue("NETx.XCON.COM.Config.Baudrate","9600")

nxa.SetValue("NETx.XCON.COM.Config.DataBits","8")

nxa.SetValue("NETx.XCON.COM.Config.Parity","0")

nxa.SetValue("NETx.XCON.COM.Config.StopBits","1")

--resto de parámetros por defecto

--Habilita comunicaciones

nxa.SetValue("NETx.XCON.COM.Enabled","1")

--Envía mensaje COIL1

if (nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Coils\\0")==true) then--si se activa la alarma...

nxa.SetValue("NETx.XCON.COM.OUT","Alarma puerta abierta en camara frigorifica")--COIL0=1

else nxa.SetValue("NETx.XCON.COM.OUT","Rearme, puerta en camara frigorifica cerrada correctamente") --COIL0=0

end

end
```

11 XCON: INTERFAZ UDP

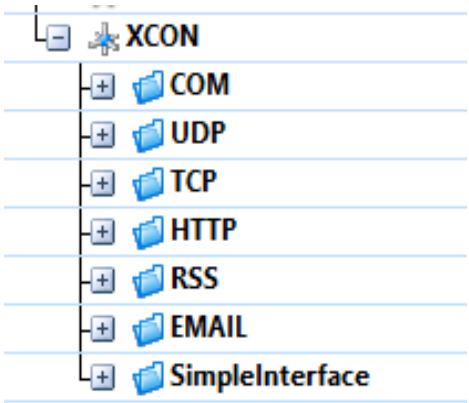
En este apartado explicaremos el funcionamiento del XCON del NETx BMS Studio, donde se encuentran los diferentes elementos que se pueden utilizar para la comunicación con otros sistemas o servicios de red (Puerto COM, UDP, TCP, HTTP, correo electrónico). En concreto nos centraremos en el protocolo UDP.

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.

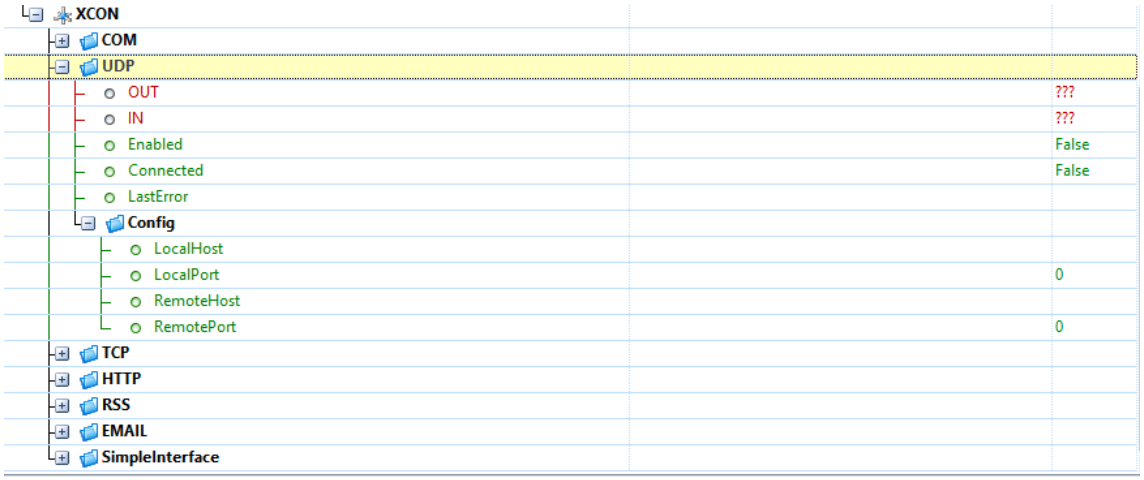
En la familia de protocolos de Internet UDP proporciona una sencilla interfaz entre la capa de red y la capa de aplicación. UDP no otorga garantías para la entrega de sus mensajes (por lo que realmente no se debería encontrar en la capa 4) y el origen UDP no retiene estados de los mensajes UDP que han 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.



11.1 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.

11.2. EJEMPLO

A continuación configuraremos la interfaz UDP y realizamos una prueba de comunicaciones con Socket Test. Para realizar la prueba usaremos la dirección IP local.

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.

CONFIGURACIÓN DEL 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:

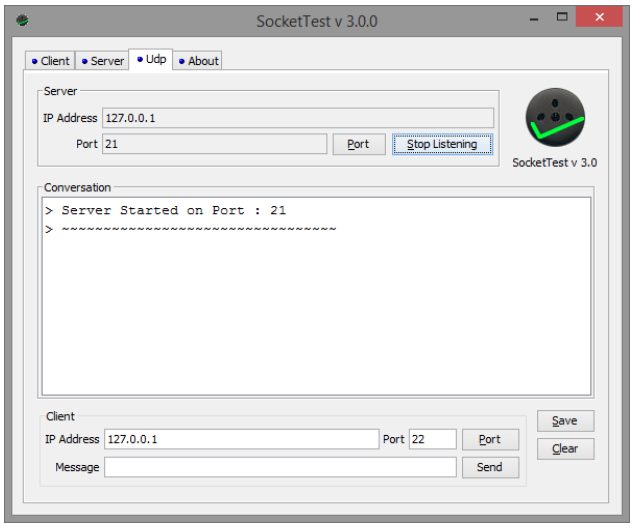
Para el servidor:

- 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:



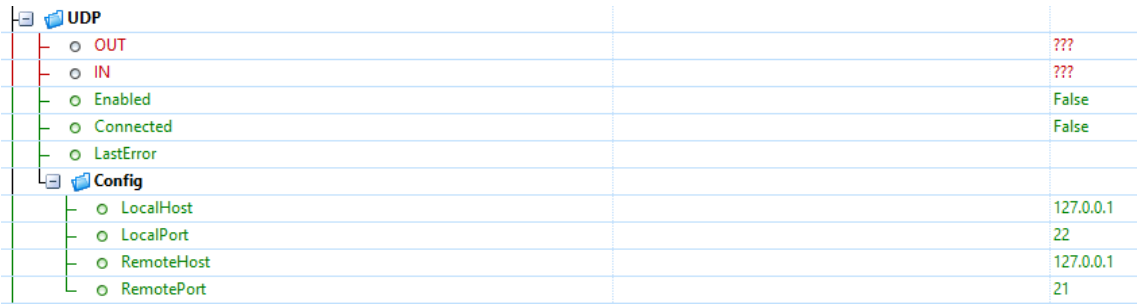
Se puede observar que en nuestro caso la dirección IP del servidor es la misma que la dirección IP del cliente, esto es debido a que estamos trabajando con un servidor local. Si tuviésemos un servidor externo en la dirección IP del servidor deberíamos introducir la dirección IP de dicho servidor.

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.

CONFIGURACIÓN DEL BMS SERVER STUDIO

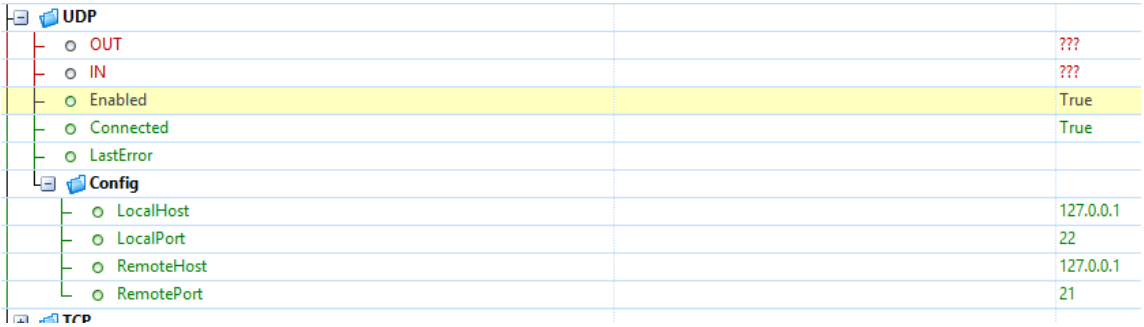
Una vez configurado el SocketTest, vamos a configurar los campos referidos al protocolo UDP del BMS Server Studio, que se encuentran en Item Tree, dentro del XCON.

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:

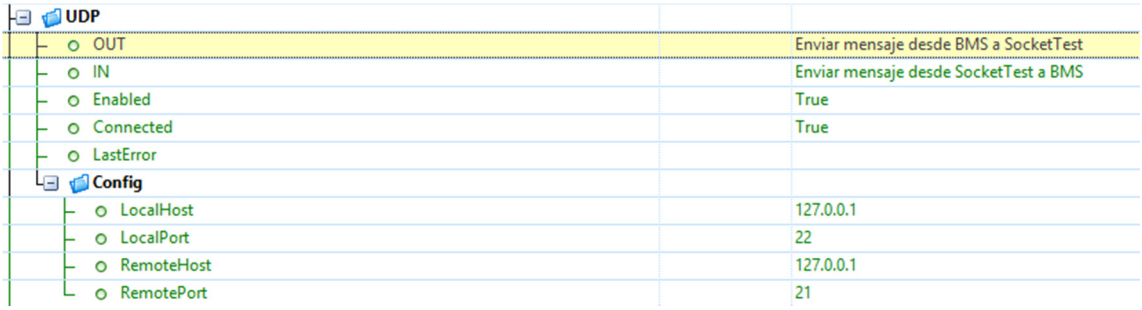


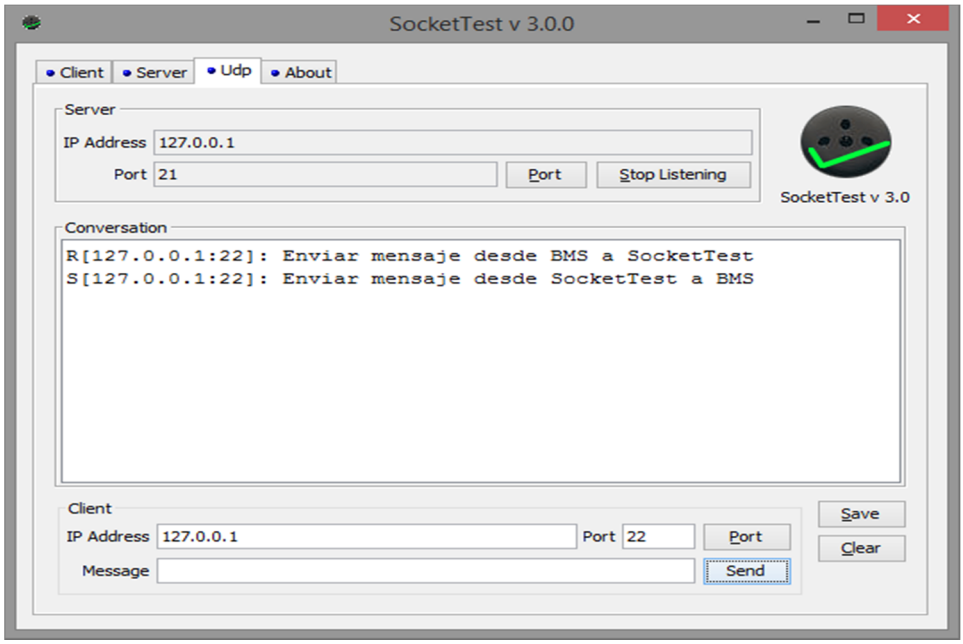
A continuación, para establecer la conexión se debe habilitar, escribiendo un "1" o "true" en la casilla *Enabled*.

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.





11.3 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 Register de Modbus 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
```

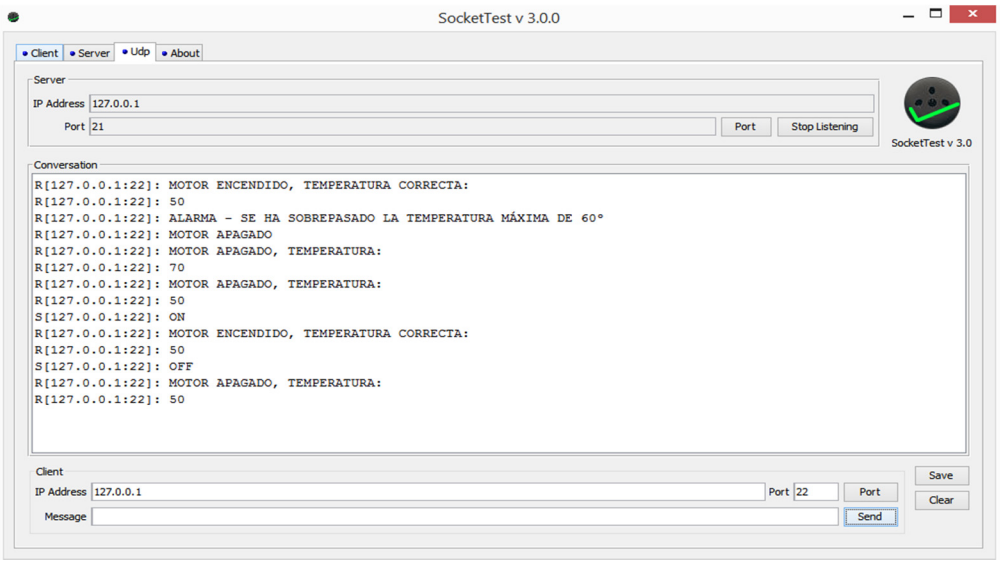
Una vez definido el script se deben introducir las funciones dentro del script nxa.Definitions.lua, en este script se deben definir los siguientes parámetros para que se inicie el script:

```
...

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:

UDP	
OUT	???
IN	???
Enabled	Verdadero
Connected	Verdadero
LastError	
Config	
LocalHost	127.0.0.1
LocalPort	22
RemoteHost	127.0.0.1
RemotePort	21

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 60°

   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

12 XCON: INTERFAZ TCP

El Transmission Control Protocol (TCP) o Protocolo de Control de Transmisión, es uno de los protocolos fundamentales en Internet.

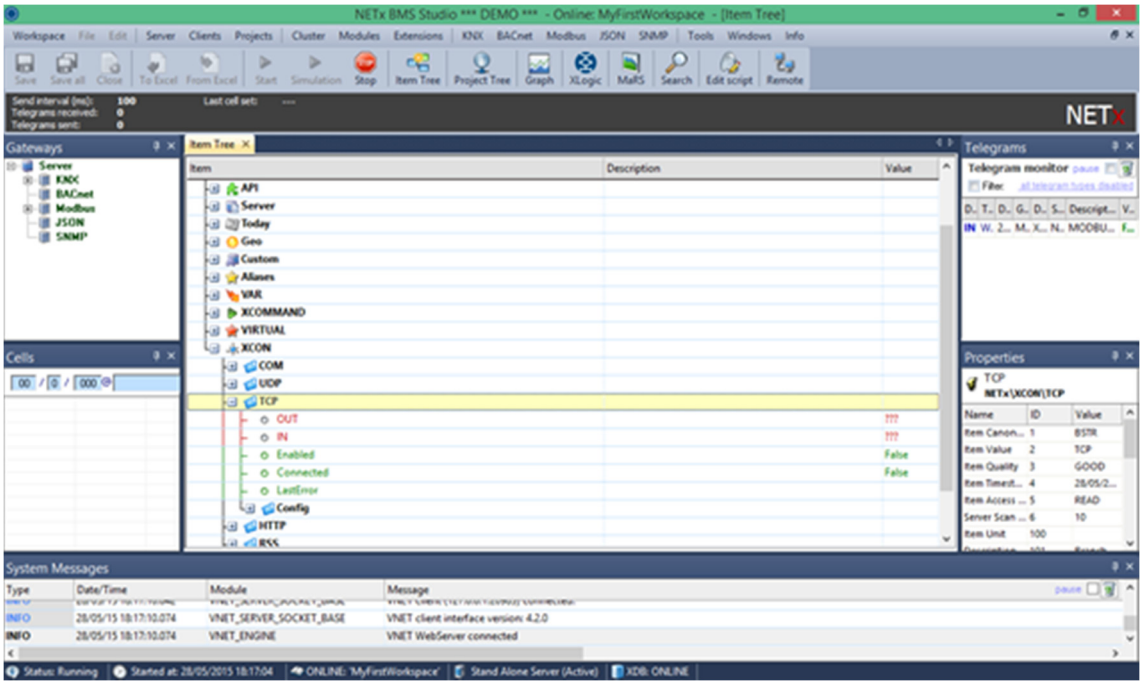
Con el uso de protocolo TCP, las aplicaciones pueden comunicarse en forma segura (gracias al de acuse de recibo -ACK- del protocolo TCP) independientemente de las capas inferiores. Esto significa que los routers (que funcionan en la capa de Internet) sólo tiene que enviar los datos en forma de datagrama, sin preocuparse con el monitoreo de datos porque esta función la cumple la capa de transporte (o más específicamente el protocolo TCP).

Para ver más en detalle sus utilidades, características y aplicaciones recomendamos consultar la siguiente referencia:

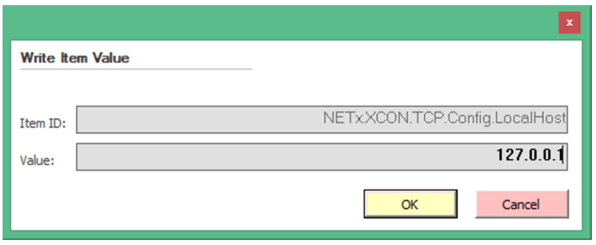
https://es.wikipedia.org/wiki/Transmission_Control_Protocol

12.1 CONFIGURACIÓN DE LA CONEXIÓN TCP

Tras iniciar el BMS Server desplegamos el menú de "TCP" para configurar los parámetros de conexión de forma adecuada.



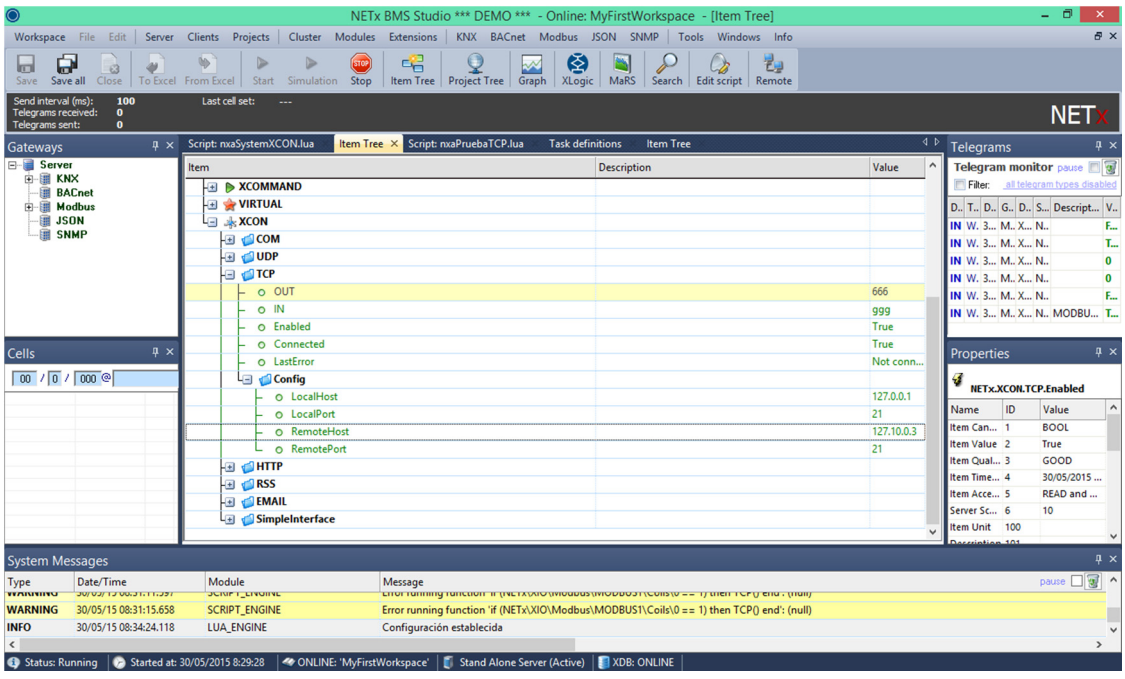
Para configurar nuestra conexión desplegamos el menú "Config" del apartado "TCP" presente en el "Item Tree" e introducimos los parámetros de configuración apropiados clicando con el botón derecho en cada uno de ellos y seleccionando "Write item value…". Se nos abrirá una ventana en la aplicación con una apariencia similar a la siguiente:



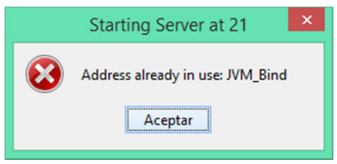
En nuestro caso configuramos la conexión de la siguiente forma:

- Dirección IP del LocalHost: 127.0.0.1
- Puerto de conexión LocalPort: 21
- Dirección IP del RemoteHost: 127.0.0.1
- Puerto de conexión RemotePort: 22

Nota: Las direcciones IP de LocalHost y RemoteHost pueden ser diferentes o iguales, ya que se emplean puertos diferentes. En el caso de que el BMS Server y la interfaz TCP emplearan el mismo puerto, no podrían utilizar la misma dirección IP, tendrían que ser diferentes.



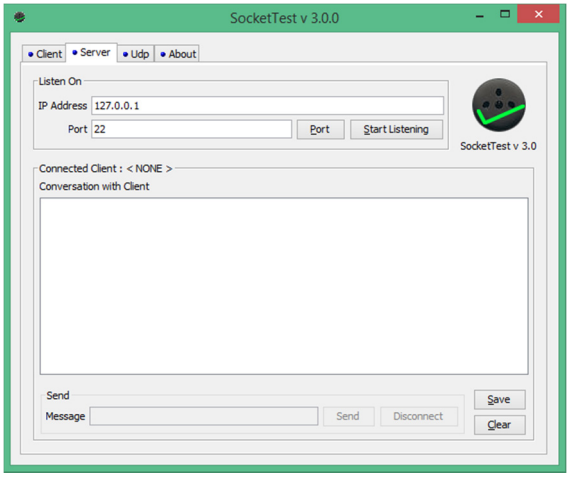
Si se intenta emplear la misma IP, el programa mostrará un mensaje igual al presente en la siguiente imagen indicando que dicha dirección ya está en uso.



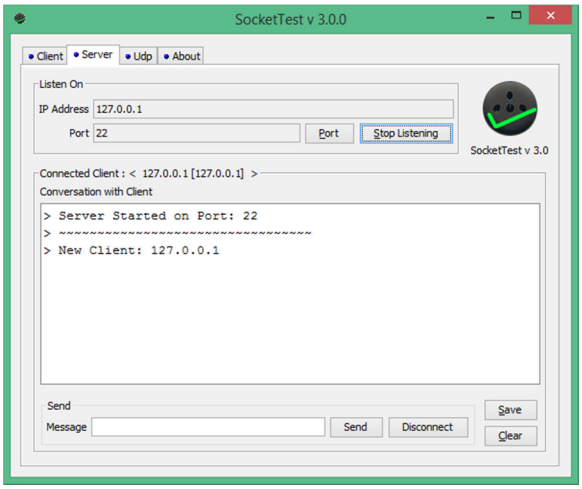
Tras realizar la configuración será necesario modificar el valor del parámetro "Enabled" para permitir la conexión. Esta modificación se realizará de la misma forma que los anteriores e introduciremos "1" o "True" para habilitar la conexión.

COMPROBACIÓN DE LA CONEXIÓN TCP

A fin de comprobar que la conexión se ha configurado y establecido correctamente, emplearemos un programa terminal que simulará la comunicación con el BMS Server, en nuestro caso emplearemos el programa "SocketTest".

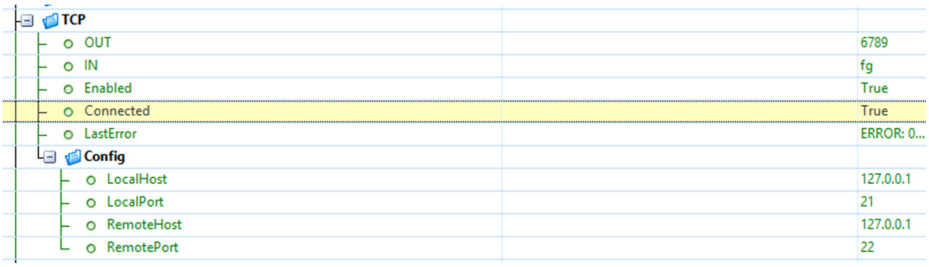


Para que la comunicación se establezca deberemos configurar el programa terminal con la misma IP que introdujimos anteriormente en "RemoteHost" y con el mismo puerto de conexión que introdujimos en "RemotePort". Una vez configurada la conexión tanto en el BMS Server como en el terminal, presionamos el botón "Start Listening" sobre el SocketTest para verificar que la conexión se ha establecido correctamente.



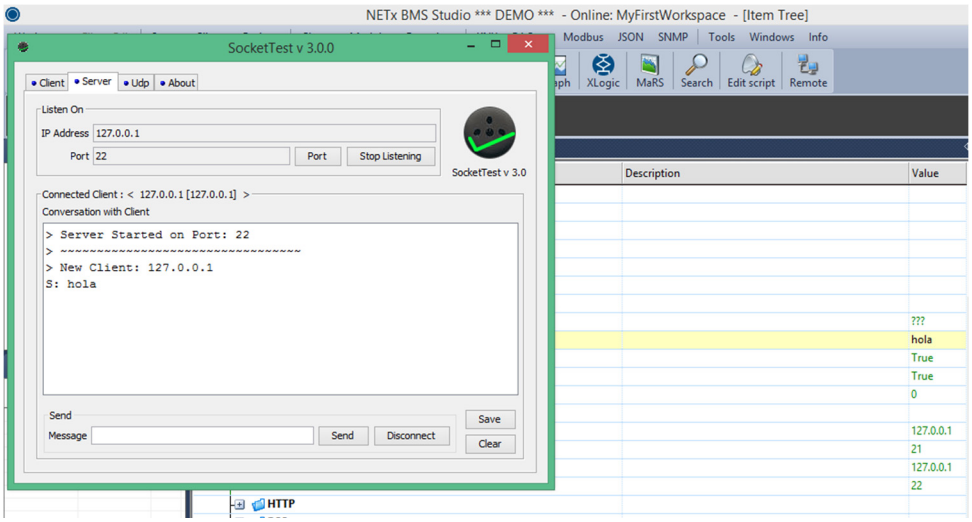
Si la comunicación se ha configurado y establecido de forma adecuada en el programa terminal visualizaremos el mensaje mostrado en la imagen superior, y

en el BMS aparecerá la variable "Connected" con el valor "True" tal y como se aprecia en la siguiente imagen.

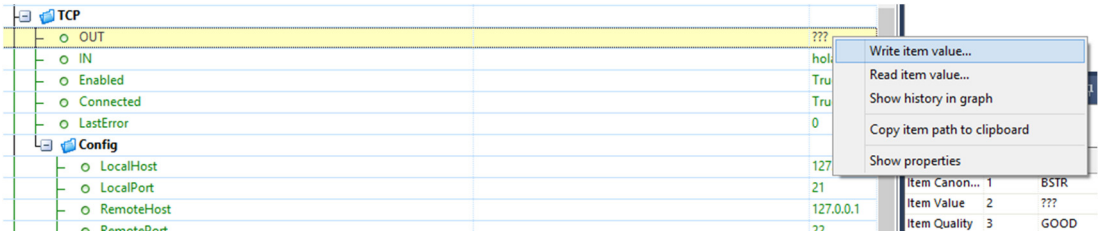


Una vez configurada, establecida y comprobada la conexión ya será posible el envío y recepción a través de ésta. Para enviar un mensaje escribiremos el mensaje deseado en "Send / Message" en el SocketTest y presionamos en el botón "Send". En la siguiente imagen puede apreciarse el envío y recepción de la cadena de caracteres "hola".

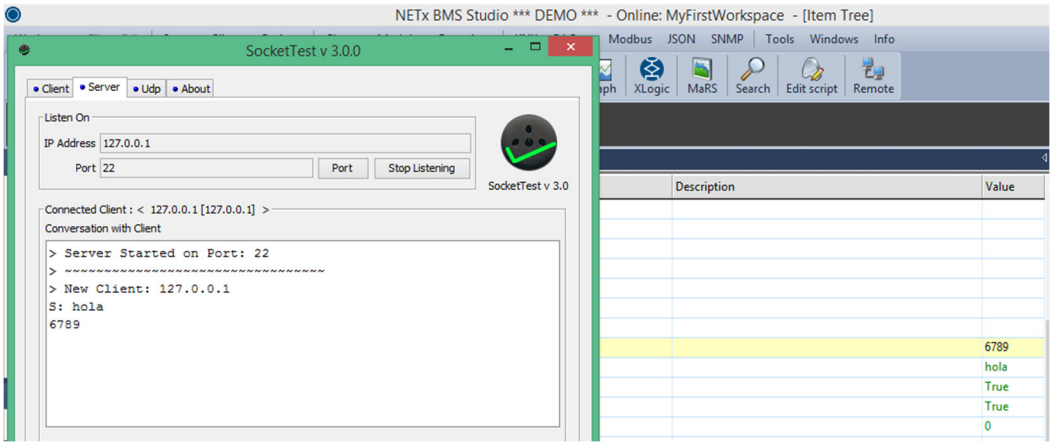
EJEMPLO DE COMUNICACIÓN TCP



La conexión TCP es una conexión bidireccional, por lo que también es posible enviar mensajes desde el BMS al terminal; para ello nos iremos al elemento "OUT" dentro del desplegable "TCP" y escribiremos el mensaje que deseamos enviar de la misma forma que hemos escrito anteriormente los parámetros de configuración.



En nuestro caso hemos enviado el valor numérico 6789 tal y como puede apreciarse en la siguiente captura de pantalla:



12.2 PARÁMETROS DE LA CONFIGURACIÓN TCP

Como se ha reflejado antes en las imágenes, para la comunicación TCP se utilizan los siguientes parámetros:

OUT

Este parámetro es el encargado de enviar los datos introducidos en el BNS Server empleando la comunicación TCP, es decir, es el buffer de bytes empleado para el envío por parte del BMS Server. Sus características son las siguientes:

- Data Type (Tipo de Dato). Es el tipo de datos que soporta, en este caso, una cadena de caracteres.
- Default value (Valor por Defecto). No tiene ningún valor por defecto.
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro OUT. Los modos de este parámetros son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro OUT; para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.OUT.

IN

Este parámetro es el encargado de enviar los datos introducidos mediante la comunicación TCP, es decir, es el buffer de bytes empleado para la recepción de datos por parte del BMS Server. Sus características son las siguientes:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, una cadena de caracteres.
- Default value (Valor por Defecto). No tiene ningún valor por defecto.

- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro IN. El único modo de acceso a este parámetro es de lectura (Read).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro IN, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.IN.

ENABLED

Este parámetro es el encargado de establecer la conexión del BMS Server mediante TCP. Es responsable de la creación de una conexión o desconexión de la comunicación TCP. Sus características son las siguientes:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta. En este caso, un dato Booleano (BOOL), es decir, 0 ó 1.
    - El 0 indica Falso (False). La conexión no se ha realizado.
    - El 1 indica Verdadero (True). La conexión se ha realizado.
- Default value (Valor por Defecto). El valor por defecto que trae este parámetro es 0, Falso (False).
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro Enabled. Los modos de este parámetros son lectura y escritura (Read and write).
- Standard Path (Ruta Estándar). Es la ruta del Item del parámetro Enabled, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Enabled.

CONNECTED

Este parámetro es el encargado de establecer la conexión TCP, es decir, cuando este parámetro vale 1 indica que el BMS Server está conctado. Al igual que el parámetro Enabled, es el responsable de la creación de una conexión o desconexión con la interfaz TCP. Si uno de los dos parámetros se encontrara a 0 (False), la conexión TCP no se realizaría. Las características de este parámetro son las siguientes:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, Booleano (BOOL), es decir, 0 ó 1.
    - El 0 indica Falso (False). La conexión no se ha realizado.
    - El 1 indica Verdadero (True). La conexión se ha realizado.
- Default value (Valor por Defecto). El valor por defecto que trae este parámetro es 0, Falso (False).

- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro Connected. El modo de acceso a este parámetro es solo lectura (Read).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro Connected, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Connected.

LASTERROR

Este parámetro es el encargado de indicar si se ha establecido algún error al establecer la conexión TCP. Las características de este parámetro son:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, una cadena de caracteres.
- Default value (Valor por Defecto). El valor por defecto que trae este parámetro es una cadena vacía (Empty string).
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro LastError. Los modos de acceso a este parámetro son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro LastError, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.LastError.

LOCALHOST

Este parámetro es el que contiene la dirección IP Local, en este caso sería la dirección IP del BMS Server. Las características de este parámetro son:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, una cadena de caracteres.
- Default value (Valor por Defecto). No tiene valor por defecto.
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro LocalHost. Los modos de acceso a este parámetro son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro LocalHost, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Config.LocalHost.

LOCALPORT

Este parámetro es el que contiene el Puerto Local, en este caso sería el Puerto del BMS Server. Las características de este parámetro son:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta. Es un entero de 4 bytes (-128 a +127 ó 0 a 255).
- Default value (Valor por Defecto). El valor por defecto es 0.
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro LocalPort. Los modos de este parámetros son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro LocalPort, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Config.LocalPort.

REMOTEHOST

Este parámetro es el que contiene la dirección IP del dispositivo o simulador remoto para poder establecer la comunicación TCP, en nuestro caso sería la dirección IP del Simulador TCP, SocketTest. Las características de este parámetro son:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, una cadena de caracteres.
- Default value (Valor por Defecto). No tiene valor por defecto.
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro RemoteHost. Los modos de este parámetros son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro RemoteHost, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Config.RemoteHost.

REMOTEPORT

Este parámetro es el que contiene el Puerto del dispositivo o simulador remoto para poder establecer la comunicación TCP, en este caso sería el Puerto del Simulador TCP, SocketTest. Las características de este parámetro son:

- Data Type (Tipo de Dato). Es el tipo de dato que soporta, en este caso, un entero de 4 bytes.
- Default value (Valor por Defecto). El valor por defecto es 0.
- Access Rights (Modos de Accesos). Indica en qué modos podemos acceder al parámetro RemotePort. Los modos de este parámetros son lectura y escritura (Read and Write).
- Standard Path (Ruta Estándar). Es la ruta del Ítem del parámetro RemoteHost, para poderla emplear, por ejemplo, con los comandos de la programación LUA. La ruta es NETx.XCON.<TCP>.Config.RemotePort.

12.3 COMUNICACIÓN TCP EMPLEANDO SCRIPTS LUA

La función principal que utiliza el BMS Server para la ejecución de los Scripts LUA es la función **xcon.CreateTCP**.

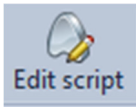
Esta función viene por defecto en el BMS Server, en el archivo nxaSystemXCON.lua, es un Script que inicializa los parámetros antes mencionados y permite utilizarlos.

Los parámetros a tener en cuenta son:

- El identificador de la conexión dentro de la secuencia de comandos LUA.
- La dirección IP de la Interfaz Local, en este caso la dirección IP del BMS Server. Si el campo está vacío la detecta el sistema.
- El Puerto que se utiliza para la conexión, si es 0 el sistema selecciona uno.
- La dirección IP o el nombre Host del servidor remoto, en este caso el del Simulador TCP SocketTest.
- El Puerto del Servidor Remoto que se utiliza para la conexión.

CONFIGURACIÓN DE LA CONEXIÓN A PARTIR DEL SCRIPT POR DEFECTO

Desde el BMS Server podemos acceder a varios Scripts proporcionados por defecto simplemente clicando en "Edit script" (botón que se aprecia en la imagen adjunta). Dentro del Script nxaSystemXCON.lua, se pueden modificar los parámetros OUT, IN, Enabled, Connected, LastError, LocalHost, LocalPort, RemoteHost y RemotePort, para establecer la configuración por defecto de la comunicación TCP y así cuando se inicie el programa BMS Server aparecerá dicha configuración.



La parte que nos interesa del archivo es la comunicación TCP y es la siguiente:

```
--
================================================================================

-- TCP Subsystem

--
================================================================================

function Create__SysTCP()

 local CfgCngFunc = "__SysTCPConfigChanged()"

 local SndFunc    = "__SysTCPSend()"
```

```
local EnbFunc    = "__SysTCPEnabled()"

sysTCP = {}

sysTCP[IsCON] = false

sysTCP[NM]    = "__SysTCP"

sysTCP[DP]    = "TCP"


sysTCP[OUT] = nxa.AddSysCustomItem(OUT, EptDESC, nxa.access.All,      nxa.type.String,  "", 0, 0, IdSep, bPath, sysTCP[DP])

sysTCP[IN]  = nxa.AddSysCustomItem(IN,  EptDESC, nxa.access.Readable, nxa.type.String,  "", 0, 0, IdSep, bPath, sysTCP[DP])

sysTCP[ENB] = nxa.AddSysCustomItem(ENB, EptDESC, nxa.access.All,      nxa.type.Boolean, "", 0, 0, IdSep, bPath, sysTCP[DP])

sysTCP[CON] = nxa.AddSysCustomItem(CON, EptDESC, nxa.access.Readable, nxa.type.Boolean, "", 0, 0, IdSep, bPath, sysTCP[DP])

sysTCP[LE]  = nxa.AddSysCustomItem(LE,  EptDESC, nxa.access.All,      nxa.type.String,  "", 0, 0, IdSep, bPath, sysTCP[DP])

sysTCP[LH]  = nxa.AddExtSysCustomItem(LH, EptDESC, nxa.access.All, nxa.type.String,  true, false, false, "", 0, 0, IdSep, bPath, sysTCP[DP], cfgPath)

sysTCP[LP]  = nxa.AddExtSysCustomItem(LP, EptDESC, nxa.access.All, nxa.type.Integer, true, false, false, "", 0, 0, IdSep, bPath, sysTCP[DP], cfgPath)

sysTCP[RH]  = nxa.AddExtSysCustomItem(RH, EptDESC, nxa.access.All, nxa.type.String,  true, false, false, "", 0, 0, IdSep, bPath, sysTCP[DP], cfgPath)

sysTCP[RP]  = nxa.AddExtSysCustomItem(RP, EptDESC, nxa.access.All, nxa.type.Integer, true, false, false, "", 0, 0, IdSep, bPath, sysTCP[DP], cfgPath)


nxa.SetValue(sysTCP[ENB], false)

nxa.SetValue(sysTCP[CON], false)

nxa.SetValue(sysTCP[LE], "")

nxa.SetValue(sysTCP[LH], "")

nxa.SetValue(sysTCP[LP], 0)

nxa.SetValue(sysTCP[RH], "")

nxa.SetValue(sysTCP[RP], 0)


nxa.AddScriptTask(sysTCP[LH],  "", true, true, true, 100, CfgCngFunc)

nxa.AddScriptTask(sysTCP[LP],  "", true, true, true, 100, CfgCngFunc)
```

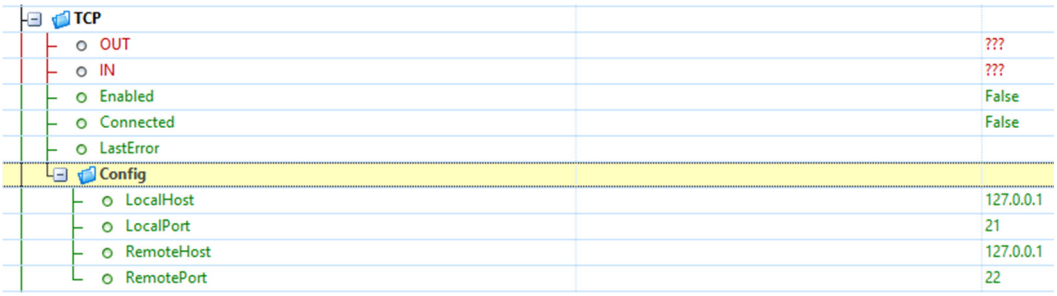
```
 nxa.AddScriptTask(sysTCP[RH],  '', true, true, true, 100, CfgCngFunc)
 nxa.AddScriptTask(sysTCP[RP],  '', true, true, true, 100, CfgCngFunc)
 nxa.AddScriptTask(sysTCP[OUT], '', true, true, true, 100, SndFunc)
 nxa.AddScriptTask(sysTCP[ENB], '', true, true, true,   0, EnbFunc)
end

function __SysTCPEnabled()
 xcon.Close(sysTCP[NM])
 if (nxa.InputValue() == true) then
   -- nxa.LogInfo("Sys TCP: Enabled")
   xcon.CreateTCP(
     sysTCP[NM],
     nxa.GetValue(sysTCP[LH], ''),
     nxa.GetValue(sysTCP[LP], 0),
     nxa.GetValue(sysTCP[RH], ''),
     nxa.GetValue(sysTCP[RP], 0))
 end
end

function __SysTCPConfigChanged()
 -- nxa.LogInfo("Sys UDP: configuration changed")
 nxa.SetValue(sysTCP[ENB], false)
end

function __SysTCPSend()
 if (sysTCP[IsCON] == false) then
   nxa.SetValue(sysTCP[LE], "Not connected")
 else
   xcon.SendText(sysTCP[NM], nxa.InputValue())
 end
end
```

```
function __SysTCP_OnErrorEvent(vData)
 -- nxa.LogWarning("Sys TCP: " .. vData)
 nxa.SetValue(sysTCP[LE], vData)
end

function __SysTCP_OnConnectEvent()
 -- nxa.LogInfo("Sys TCP: connected")
 sysTCP[IsCON] = true
 nxa.SetValue(sysTCP[CON], true)
end

function __SysTCP_OnDisconnectEvent()
 -- nxa.LogInfo("Sys TCP: disconnected")
 sysTCP[IsCON] = false
 nxa.SetValue(sysTCP[CON], false)
end

function __SysTCP_OnReceiveEvent(vData, vSize, vIP, vPort)
 -- nxa.LogInfo("Sys TCP: received " .. vSize .. " Bytes from " .. vIP .. ":" .. vPort .. " = " .. vData)
 nxa.SetValue(sysTCP[IN], vData)
end
```

Dentro de este apartado nos centraremos en las líneas donde modificar la configuración por defecto, estas líneas son las que se muestran a continuación:

```
nxa.SetValue(sysTCP[ENB], false)
nxa.SetValue(sysTCP[CON], false)
nxa.SetValue(sysTCP[LE], "")
nxa.SetValue(sysTCP[LH], "127.0.0.1")
nxa.SetValue(sysTCP[LP], 21)
nxa.SetValue(sysTCP[RH], "127.0.0.1")
nxa.SetValue(sysTCP[RP], 22)
```

Se ha puesto por defecto la configuración que hemos utilizado para los ejemplos anteriores. Al iniciar el BMS Server obtenemos:



Como podemos apreciar, la configuración se ha realizado de forma correcta e idéntica a la configuración establecida manualmente. De esta forma, incorporando los parámetros de configuración en el script de lua nos evitamos realizar la configuración de forma manual cada vez que arrancamos la simulación del BMS Server.

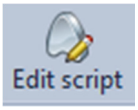
CONFIGURACIÓN A TRAVÉS DE UN SCRIPT PROPIO

Para crear un Script nos dirigimos a la ruta:

C:\Program Files\NETxAutomation\NETx.BMS.Server.2.0\Workspaces\Workspace\ScriptFiles

Creamos un documento de texto con extensión ".lua" donde escribiremos nuestro código.

Una vez creado el archivo que aloja el código lua, podremos modificarlo desde el propio BMS Server o desde el block de notas o programas similiares. En el caso de que deseemos modificar el script desde el propio programa, clicaremos en el botón Edit Script que se aprecia en la siguiente imagen:



El código empleado en nuestro Script es el siguiente:

```
function TCP()

-- Configuración de la comunicación TCP entre NETx BMS Studio y un simulado TCP, en este caso se ha utilizado SocketTest v 3.0.0

nxa.SetValue("NETx.XCON.TCP.Config.LocalHost","127.0.0.1")

nxa.SetValue("NETx.XCON.TCP.Config.LocalPort","21")

nxa.SetValue("NETx.XCON.TCP.Config.RemoteHost","127.0.0.1")

nxa.SetValue("NETx.XCON.TCP.Config.RemotePort","22")

-- Mensaje informativo
```

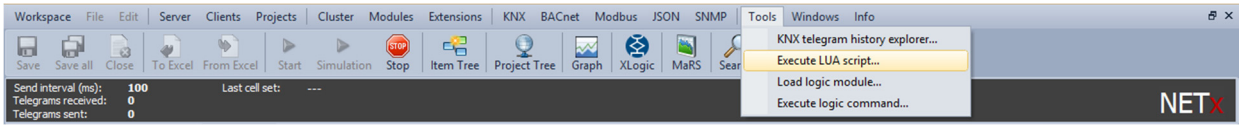
```
nxa.LogInfo("Configuración establecida")

-- Ejecución de la Tarea. Escribe en el parámetro OUT del BMS Server el valor del Holding Registers 1.

-- Escribe en el Holding Register 0 el valor del parámetro IN.

nxa.SetValue("NETx.XCON.TCP.OUT",nxa.GetValue("NETx\\XIO\\Modbus\\MODBUS1\\Holding Registers\\1"))

nxa.WriteValue("NETx\\XIO\\Modbus\\MODBUS1\\Holding Registers\\0",nxa.GetValue("NETx.XCON.TCP.IN"))

end
```

Una vez ya creado el script con nuestro código, será necesario dar un paso más para poder ejecutarlo. Al igual que hemos hecho con nuestro script, abriremos el script "nxaDefinitions.lua" alojado en la ruta especificada anteriormente e incluiremos la siguiente línea:

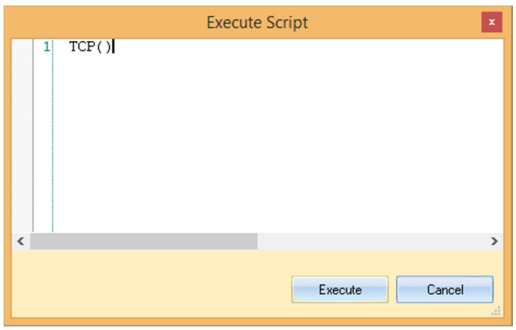
```
require "Nombredelarchivo"
```

Esta línea se incluirá a continuación de las líneas ya presentes en el archivo que presentan un "require" y nos permitirá el empleo de nuestro propio Script.

Una vez creado y modificado el Script hasta tener el código deseado, desde el BMS Server se ejecuta el Script desde la pestaña Execute LUA script, que se encuentra en Tools.



Tras realizar este paso nos aparecerá la siguiente ventana:



Para ejecutar el código deseado escribiremos el nombre de la función que deseamos ejecutar tal y como puede apreciarse en la imagen superior, y presionamos execute. En el caso de que todo se desarrolle de forma correcta se ejecutará nuestro código, en el caso de que se produzca un error en la ejecución o un mensaje, éste aparecerá en la ventana System messages del programa.

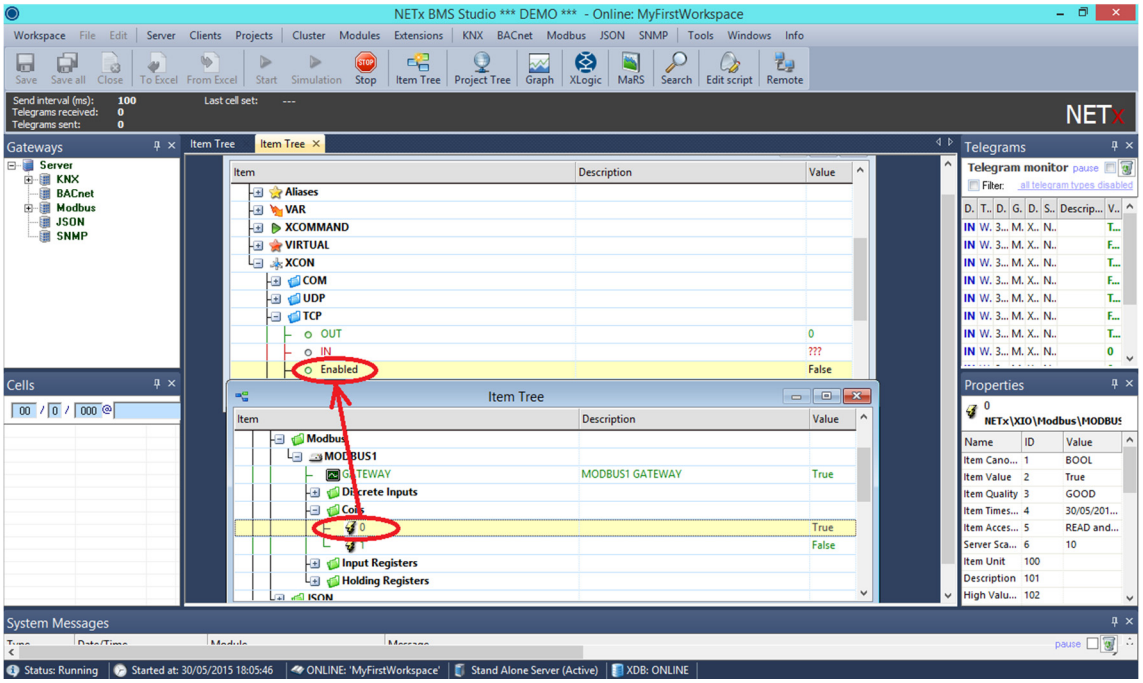
Al ejecutar este script se modifican los parámetros ya mencionados en la configuración manual:

- LocalHost: 127.0.0.1
- LocalPort: 21
- RemoteHost: 127.0.0.1
- RemotePort: 22

Como podemos apreciar el parámetro Enable no se modifica con nuestro script, esto es debido a que dicho parámetro debe ser modificado de forma manual, de manera que será posible realizar y ejecutar una función dentro de un script que establezca la configuración pero para establecer la conexión será preciso que se modifique el parámetro Enable manualmente. Tras esto, ya queda configurada la conexión y será posible emplear la conexión TCP a nuestro antojo.

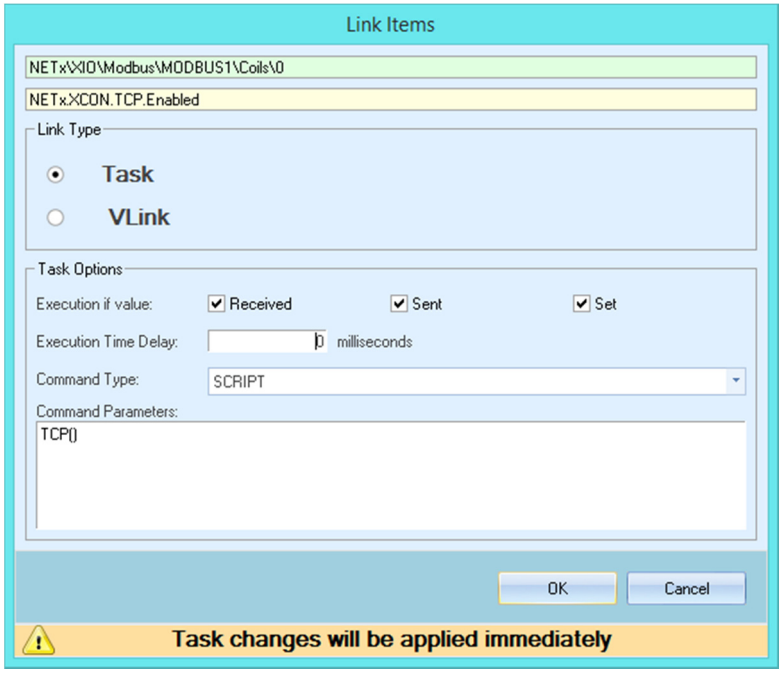
12.4 EJECUCIÓN DE UN SCRIPT ASOCIADO A UN PARÁMETRO O PUNTO VIRTUAL

También es posible configurar el BMS Server vinculando la ejecución de un script de lua a un parámetro o punto virtual, de manera que modificando éste se ejecute el parámetro.

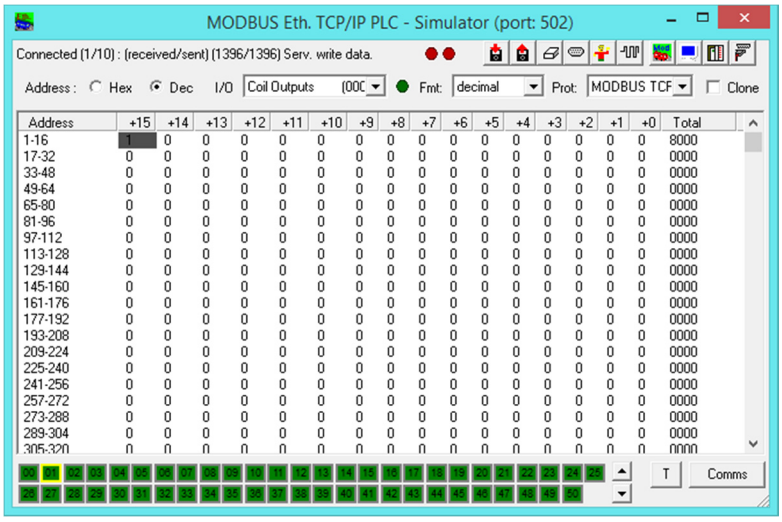


Para configurar la ejecución de ésta forma el procedimiento es muy sencillo, abriremos dos pestañas Item Tree, de forma que podamos visualizar el parámetro al que deseamos asignar la tarea que ejecutará el script y el punto con el que deseamos vincularlo. A continuación procedemos arrastrando uno

sobre otro como se aprecia en la imagen superior, de forma que se nos mostrará la siguiente ventana:

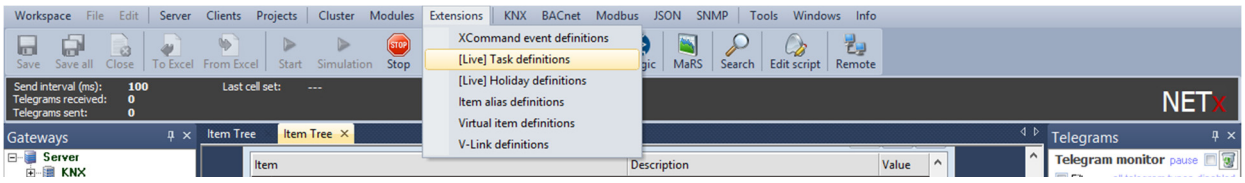


Como podemos apreciar en la parte superior de la ventana se muestran los dos puntos vinculados, en nuestro caso se ha vinculado el "Coil 0" con el parámetro "Enabled" dentro de "XCON/TCP". Para asignar el script a ejecutar seleccionaremos la opción "Task" dentro de "Link Type" y dentro de "Task Options" seleccionaremos "SCRIPT" como "Comand Type" y escribiremos el nombre de la función a ejecutar dentro del cuadro de diálogo "Command Parameters". Finalmente cliclaremos en "Ok" y ambos puntos quedarán vinculados entre sí y a la tarea seleccionada. Una vez realizado todos los pasos descritos, en nuestro ejemplo, cuando se modifique el Coil asignado desde el terminal (tal como se aprecia en la imagen inferior), se ejecutará la tarea,



También se pueden crear tareas desde la pestaña "Extensions/Task definitions". Este procedimiento ya se encuentra explicado en la documentación

proporcionada con el programa, por lo que para evitar la duplicidad de información, consultaremos dicha documentación en el caso de que sea necesario.



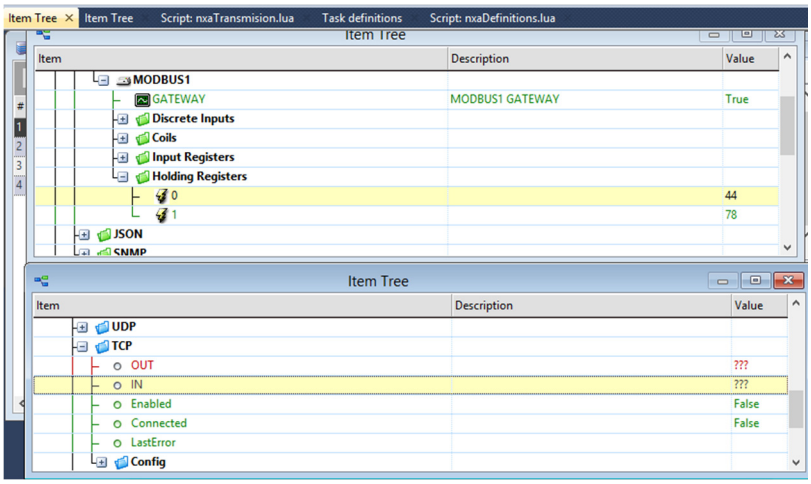
TRANSMISIÓN DE UN MENSAJE MEDIANTE TCP

Para transmitir un mensaje empleando la comunicación TCP se ha generado un Script llamado "nxaTransmision.lua". El contenido de este Script puede apreciarse a continuación:

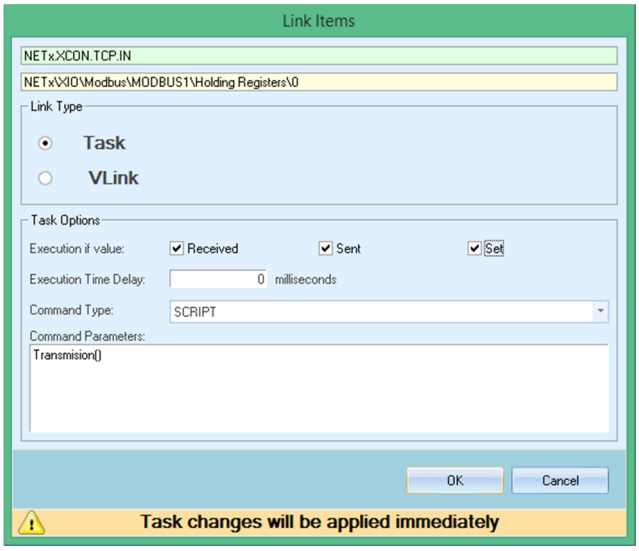
```
function Transmision()

-- Comunicación TCP entre NETx BMS Studio y un simulador TCP, en este caso se ha utilizado SocketTest v 3.0.0

nxa.SetValue("NETx.XCON.TCP.OUT",nxa.GetValue("NETx\\XIO\\Modbus\\MODBUS1\\Holding Registers\\1"))

-- Mensaje informativo

nxa.LogInfo("Comunicacion establecida")

end
```

Este Script transmitirá mediante TCP el valor del "Holding Register 1", y dicho valor aparecerá en el terminal, en nuestro caso, SocketTest. Para ello emplearemos la función "nxa.SetValue()", esta función se encargará de tomar el valor del Holding Register 1 y enviarlo mediante el parámetro "OUT" de TCP, tal y como puede apreciarse en el código. El procedimiento es el siguiente:

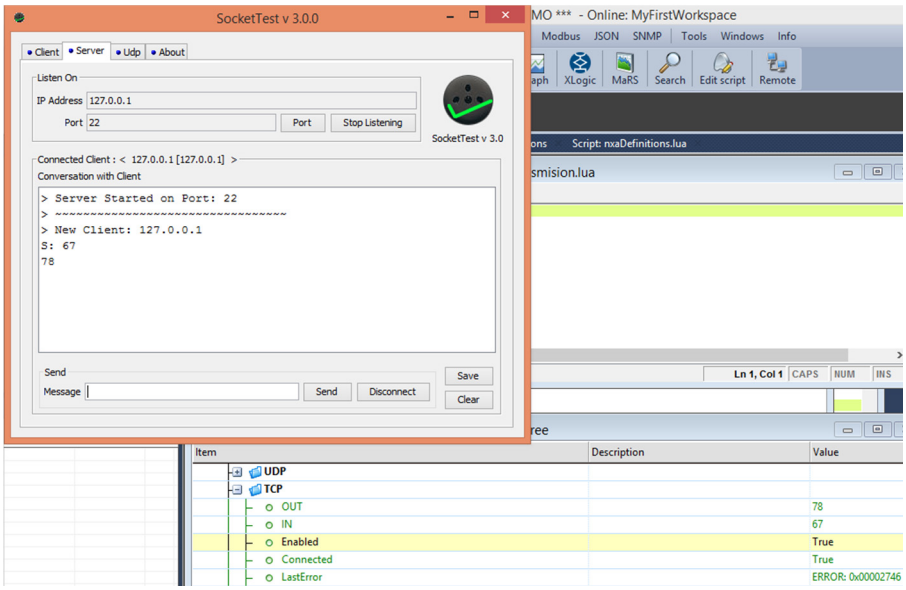
1. Se vincula la entrada de datos con el Holding Register 0



2. Se asigna el script de lua a la tarea en concreto y se configura para que al modificar el Holding Register se realice un envío. De esta forma realizamos un envío al BMS y programamos una recepción de datos.



3. Y por último, modificamos el Holding Register introduciendo un valor y éste aparecerá en el terminal tal y como se aprecia en la captura de pantalla:



13 XCON: INTERFACES HTTP Y RSS

13.1 LA INTERFAZ HTTP

La interfaz HTTP puede ser usada para obtener información de un host remoto desde el servidor de NETx BMS. Una simple petición HTTP es enviada al host remoto y su resultado es almacenado en otro ítem. Éste puede ser leído, analizado, y más información puede ser sacada de él. Necesita ser habilitada antes de usarla.

Parámetros:

GET

- El tipo de datos es String, su valor por defecto es None.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<HTTP>.GET.
- Es necesario que el parámetro Enabled esté activado para que al modificar el parámetro se envié la petición HTTP get de forma inmediata.
- Nota: No es necesario incluir http:// ya que es incluido al llamar al comando. Ejemplo: www.dte.us.es

PUT

- El tipo de datos es String, su valor por defecto es None.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<HTTP>.PUT
- Es necesario que el parámetro Enabled esté activado para que al modificar el parámetro se envié la petición HTTP put de forma inmediata.
- Nota: No es necesario incluir http:// ya que es incluido al llamar al comando.

RESULT

- El tipo de datos es String, su valor por defecto es None.
- Las reglas de acceso establecidas permiten la Lectura.
- Su ruta estándar es NETx.XCON.<HTTP>.RESULT.

- El resultado de la última petición HTTP enviada será almacenada en este ítem. Ejemplo: <HTML><HEAD>My Page</HEAD><BODY>Hello World!</BODY ></HTML>.

ENABLED

- El tipo de datos es Boolean y su valor por defecto es 0 (False).
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<HTTP>.Enabled.
- Según el manual, si este ítem del servidor es puesto a "True" cualquier comando Get/Put será inmediatamente enviado fuera del servidor a su destino determinado. Es necesario poner éste parámetro a True antes de hacer el comando Get o Put.

LASTERROR

- El tipo de datos es String y su valor por defecto es empty string.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<HTTP>.LastError
- Si un error ocurre durante la transmisión, éste será almacenado aquí. El ítem del servidor mantendrá este valor. Este cambiará en el siguiente error, también puede ser sobrescrito mediante un script y de otras maneras.

13.2 LA INTERFAZ RSS

La interfaz RSS permite al usuario obtener información desde un proveedor RSS. Ésta necesita ser habilitada antes para poder usarse. Escribiendo una RSS URL inicializaremos la comunicación.

RSS facilita la gestión y publicación de información y noticia webs. RSS es una forma estandarizada de distribución de la información de las páginas web a los lectores de las páginas. Esta información se distribuye a través de las fuentes RSS o Canales RSS. Gracias a RSS, los lectores pasan a tener una herramienta útil para mantenerse informado sobre las noticias y webs que le resultan de interés, conservando y almacenando toda la información en un solo lugar que se actualiza de manera automática.

En el archivo RSS simplemente están los datos de las novedades del sitio, como el título, fecha de publicación o la descripción. El programa que lea el RSS será

encargado de darle estilo o apariencia a los datos que se incluyan en el archivo y presentarlos de una manera atractiva al usuario y de fácil lectura.

Que RSS sea un formato basado en XML significa que el archivo RSS se compone por una serie de etiquetas definidas que tendrán un formato dado, que respetará las reglas generales de XML.

Parámetros:

GET

- El tipo de datos es String, su valor por defecto es None.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<RSS>.GET.
- Es necesario que el parámetro Enabled esté activado para que al modificar el parámetro se envié la petición RSS get de forma inmediata.
- La URL de la fuente RSS se colocará aquí.
- Ejemplo: http://www.dte.us.es (dado que funciona igual que http para url normales)
- En el caso de usar una URL RSS sería necesario poner "http:// " dirección ".xml". Ejemplo: http://estaticos.elmundo.es/elmundo/rss/portada.xml

CHANNEL

- El tipo de datos es INT4, su valor por defecto es -1.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<RSS>.CHANNEL
- El canal es índice basado en cero del canal que se quiera obtener del documento RSS. Con "-1" se devuelven todos los canales.

RESULT

- El tipo de datos es String, su valor por defecto es 0 (False).
- Las reglas de acceso establecidas permiten la Lectura.
- Su ruta estándar es NETx.XCON.<RSS>.RESULT.
- El resultado de la última petición RSS enviada será almacenada en este ítem. Ejemplo: Código del .xml

ENABLED

- El tipo de datos es Boolean y su valor por defecto es 0 (False).

- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<RSS>.Enabled.
- Si este ítem del servidor es puesto a "True" cualquier comando Get será inmediatamente enviado fuera del servidor a su destino determinado.

LASTERROR

- El tipo de datos es String y su valor por defecto es empty string.
- Las reglas de acceso establecidas permiten la Lectura y la Escritura.
- Su ruta estándar es NETx.XCON.<RSS>.LastError
- Si ocurre un error durante la transmisión, éste será almacenado aquí.

NOTA: Es necesario también comentar las dos funciones NXA para enviar emails existentes para HTTP y RSS:

13.3 FUNCIONES LUA PARA HTTP Y RSS

XCON.CREATEHTTP

Esta función puede ser usada para crear una conexión HTTP en LUA.

Parámetro: string – Handle que identifica la conexión dentro del script LUA.

XCON.CREATERSS

Esta función puede ser usada para crear una conexión RSS en LUA.

Parámetro: string – Handle que identifica la conexión dentro del script LUA.

Estas serán utilizadas en los siguientes ejemplos.

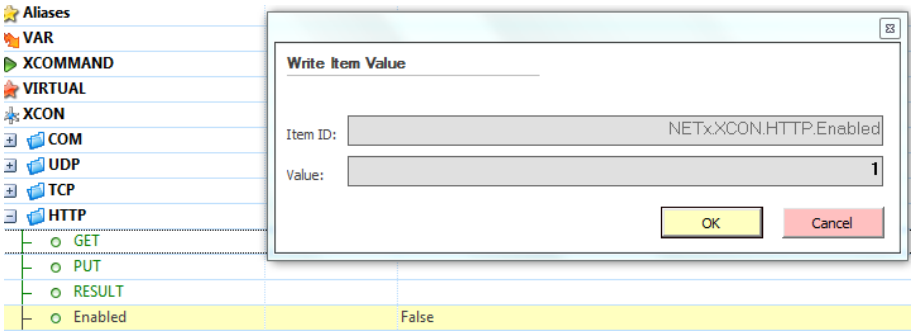
13.4 EJECUCIÓN Y EJEMPLO DE USO DE LA INTERFAZ HTTP

Primero vamos a ver cómo funciona el comando Get. Podemos hacerlo de dos formas:

DE FORMA MANUAL USANDO LA INTERFAZ GRÁFICA:

De esta manera es muy sencillo obtener el resultado y visualizar el resultado. Primero, debemos desglosar la subpestaña NETx del Item tree, después desglosar la subpestaña XCON y finalmente desglosar también el apartado de la interfaz HTTP.

Una vez hecho esto, tan solo tenemos que clicar con el botón derecho del ratón en el apartado Enabled -> Write ítem value y ponemos 1. Así pondremos enabled a True y habilitaremos la conexión HTTP.



Ahora, procederemos a realizar el envío del comando Get, para ello haremos click derecho en GET -> Write ítem value y ponemos la URL deseada (recordar que no es necesario incluir http:// ya que se incluirá automáticamente en el envío). Por ejemplo: Si escribimos "www.google.es" obtendremos en RESULT el código HTML de la página web.

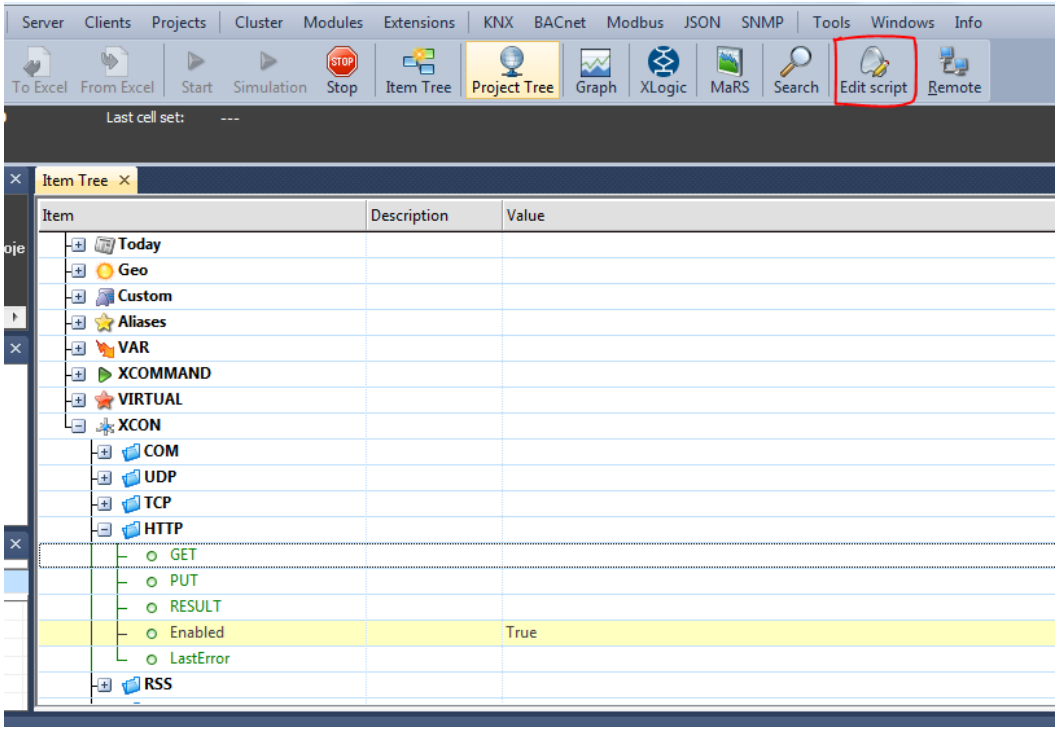
MEDIANTE SCRIPTS

Esta forma es un poco más compleja, pero conociendo los comandos adecuados veremos que es muy útil, ya que podríamos establecer que cada vez que se iniciase el servidor NETx BMS se ejecutase dicho fragmento de código entre otras utilidades.

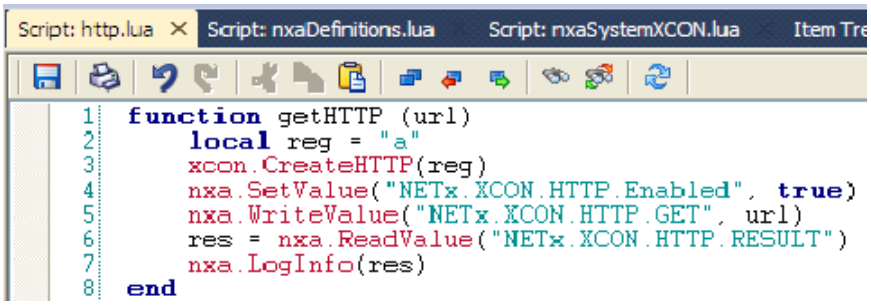
Primero crearemos el script, para ello nos crearemos un archivo http.lua en la carpeta donde se almacenan los script cuya ruta es

C:\Program Files (x86)\NETxAutomation\NETx.BMS.Server.2.0\Workspaces\Workspace\ScriptFiles.

Una vez hecho esto clicaremos en la opción Edit script, y elegiremos el archivo .lua que acabamos de crear.



A continuación se mostrará el contenido de http.lua y se explicarán algunas de las funciones utilizadas.



xcon.CreateHTTP(Handle): Crea una conexión HTTP en LUA.

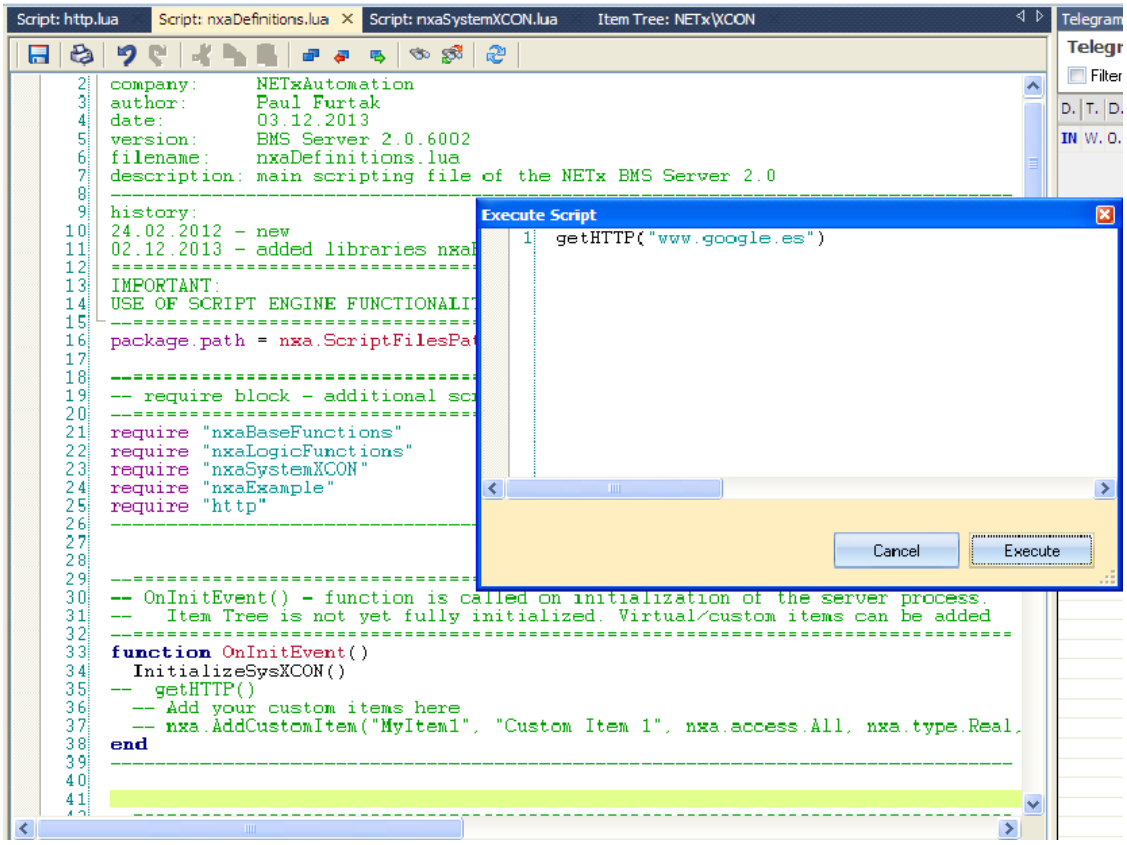
nxa.SetValue ("rutaDelItem", valor) -- rutaDelItem obtenida haciendo click derecho en Enabled-> copy ítem path to clipboard

nxa.WriteValue ("rutaDelItem", valor) -- Esta función escribe activamente el valor dado del item especificado. En comparación con SetValue, una escritura de valor también propagará el cambio de valor al dispositivo de campo. Un retraso opcional también podría especificarse.

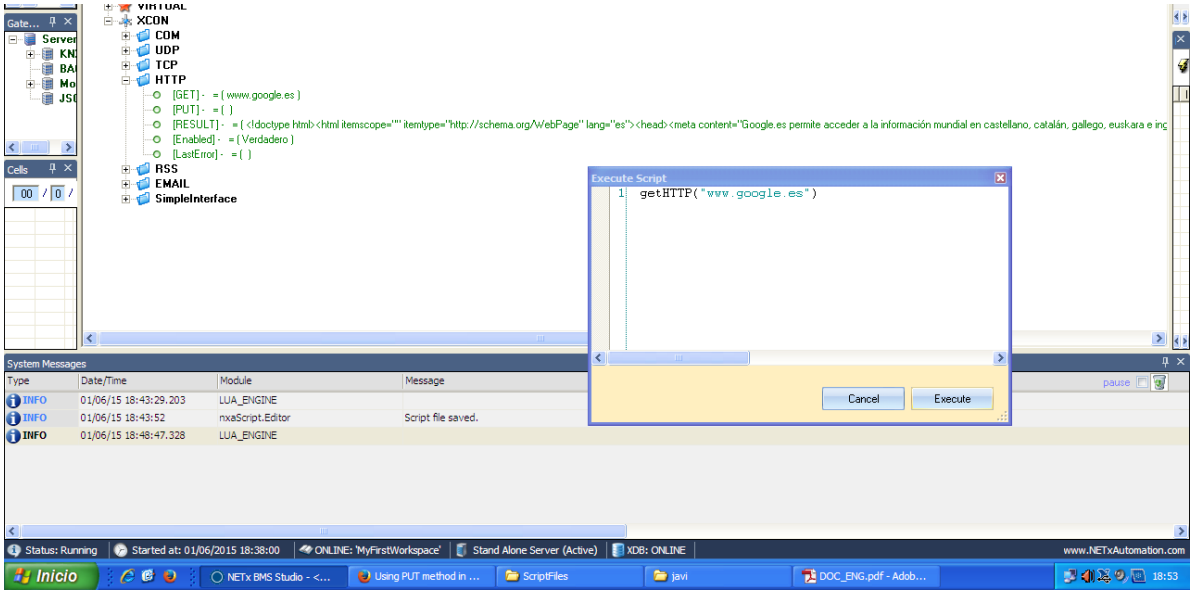
nxa.ReadValue ("ruta") --Lee el valor del campo RESULT

nxa.LogInfo --lo que se desea mostrar por la consola System Messages

También es necesario modificar el script nxaDefinitions.lua, si queremos que nuestro script pueda ser ejecutado mediante la herramienta Execute LUA script deberemos incluir el script creado en este mediante el comando require tal y como podemos observar en la siguiente captura:



Por último, en la siguiente página mostraremos el resultado de ejecutar dicho script viendo cómo se rellenan los campos de la interfaz gráfica por sí mismos.



A la hora de probar el comando Put el procedimiento sería el mismo, tan solo sería necesario cambiar la ruta y la url en el comando nxa.WriteValue, y poner en su lugar la sintaxsis adecuada para una HTTP Put request, sería algo similar a esto:

```
*** REQUEST
GET /users/123 HTTP/1.1
Host: www.example.org
...

*** RESPONSE
200 OK HTTP/1.1
...

<html>
...
<form action="http://www.example.org/users/123" method="put" if-match="q1w2e3r4t5">
  <input name="user-name" value="" />
  <input name="hat-size" value="" />
  <input type="submit" />
</form>
...
</html>

*** REQUEST
PUT /users/123 HTTP/1.1
Host: www.example.org
If-Match: "q1w2e3r4t5"
Content-Type: application/x-www-form-urlencoded
Content-Length: nnn

user-name=mike&hat-size=medium

*** RESPONSE
200 OK HTTP/1.1
...
<html>
...
<ul>
  <li>user-name: mike</li>
  <li>hat-size: medium</li>
</ul>
...
<html>
```

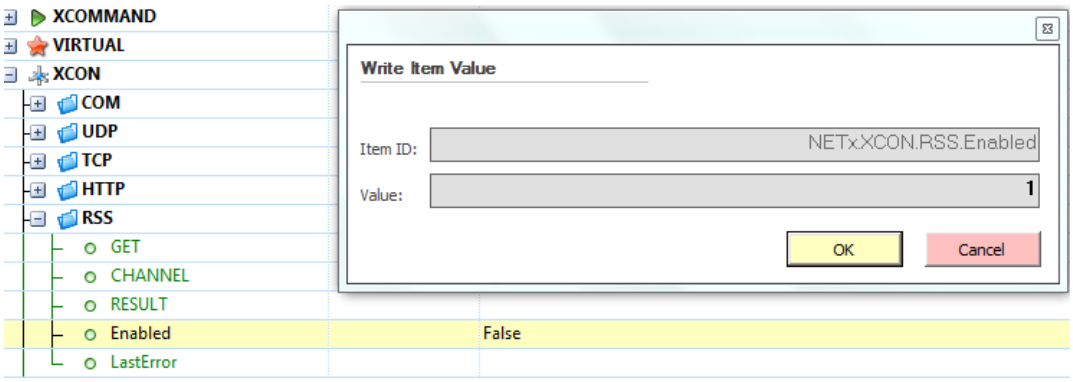
13.5 EJECUCIÓN Y EJEMPLO DE USO DE LA INTERFAZ RSS

Ahora, vamos a comprobar el funcionamiento de del comando Get de la interfaz RSS. También podemos hacerlo de dos formas:

DE FORMA MANUAL USANDO LA INTERFAZ GRÁFICA

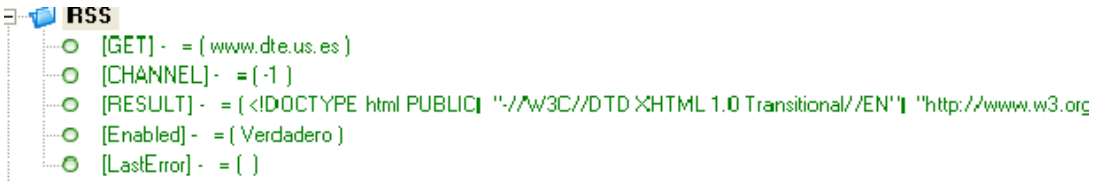
De esta manera es muy sencillo obtener el resultado y visualizar el resultado. Primero, debemos desglosar la subpestaña NETx del Item tree, después desglosar la subpestaña XCON y finalmente desglosar también el apartado de la interfaz RSS.

Una vez hecho esto, tan solo tenemos que clicar con el botón derecho del ratón en el apartado Enabled -> Write ítem value y ponemos 1. Así pondremos enabled a True y habilitaremos la conexión RSS.



Ahora, procederemos a realizar el envío del comando Get, para ello haremos click derecho en GET -> Write ítem value y ponemos la URL deseada (recordar que no es necesario incluir http:// ya que se incluirá automáticamente en el envío en caso de ser una url normal).

Por ejemplo: si escribimos "www.dte.us.es" obtendremos en RESULT el código HTML de la página web.

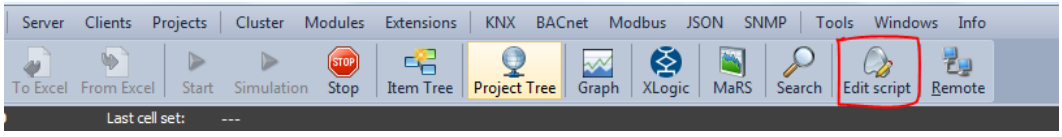


MEDIANTE SCRIPTS

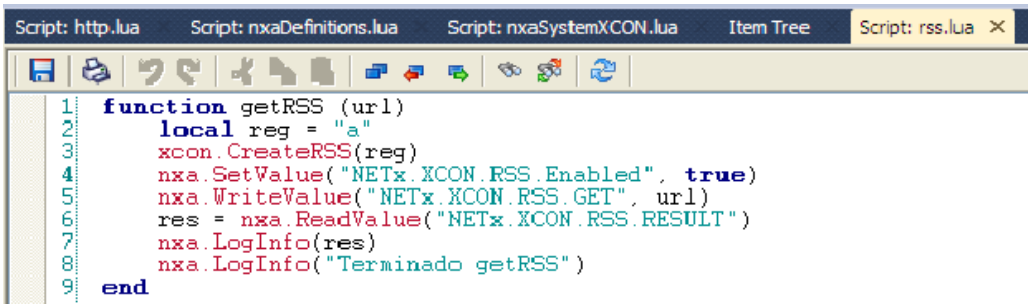
Primero crearemos el script, para ello nos crearemos un archivo rss.lua en la carpeta donde se almacenan los script

C:\Program Files (x86)\NETxAutomation\NETx.BMS.Server.2.0\Workspaces\Workspace\ScriptFiles.

Una vez hecho esto clicaremos en la opción Edit script, y elegiremos el archivo .lua que acabamos de crear.



A continuación se mostrará el contenido de http.lua y se explicarán algunas de las funciones utilizadas.



xcon.CreateRSS(Handle) -- Crea una conexión RSSen LUA.

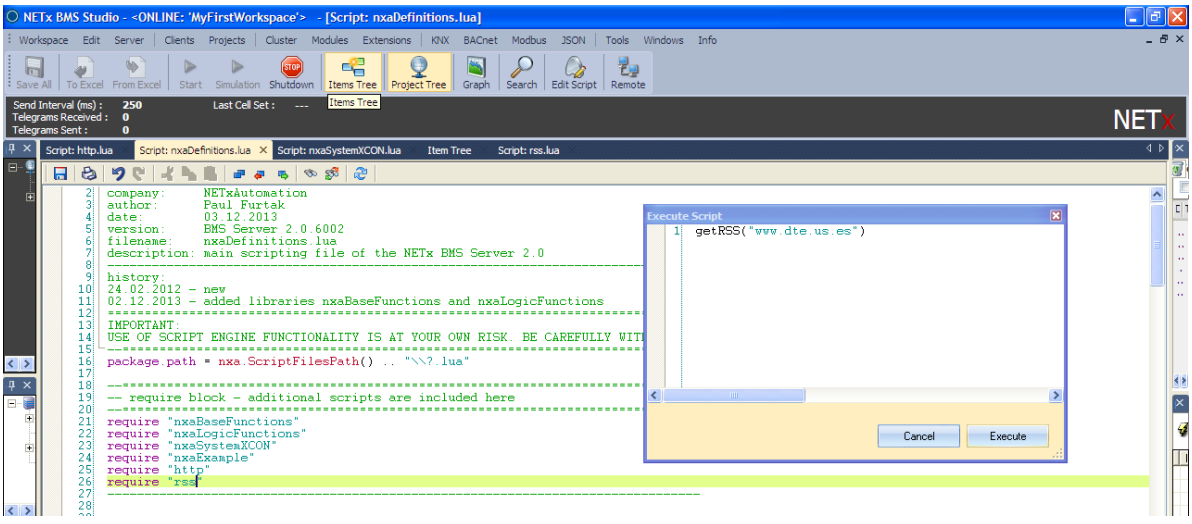
nxa.SetValue ("rutaDelItem", valor) -- rutaDelItem obtenida haciendo click derecho en Enabled-> copy ítem path to clipboard

nxa.WriteValue ("rutaDelItem", valor) -- Esta función escribe activamente el valor dado del item especificado. En comparación con SetValue, una escritura de valor también propagará el cambio de valor al dispositivo de campo. Un retraso opcional también podría especificarse.

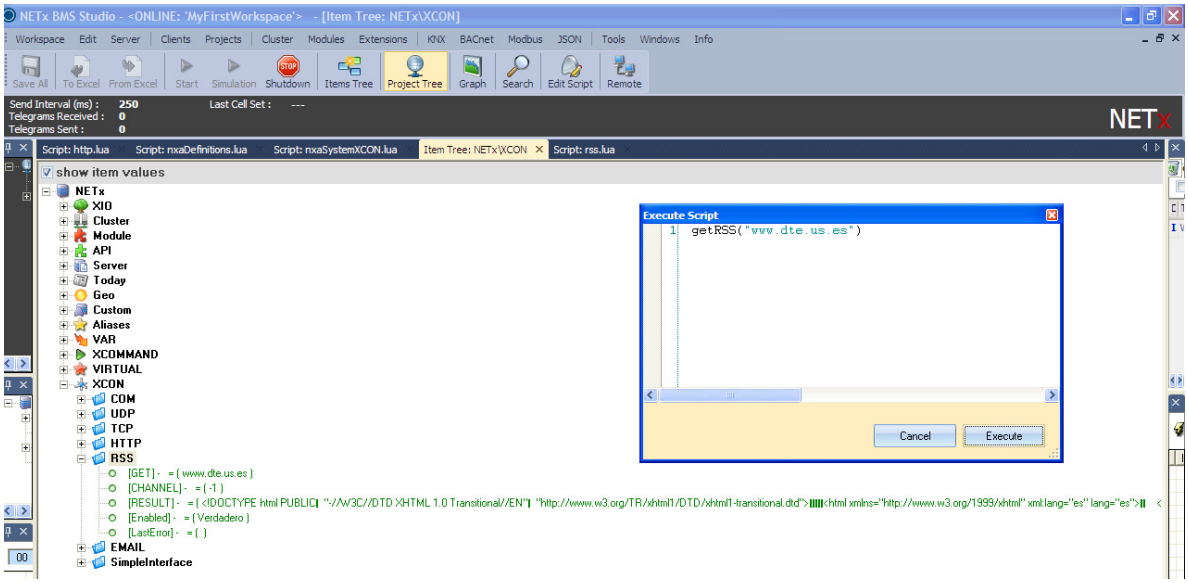
nxa.ReadValue ("ruta" --: Lee el valor del campo RESULT

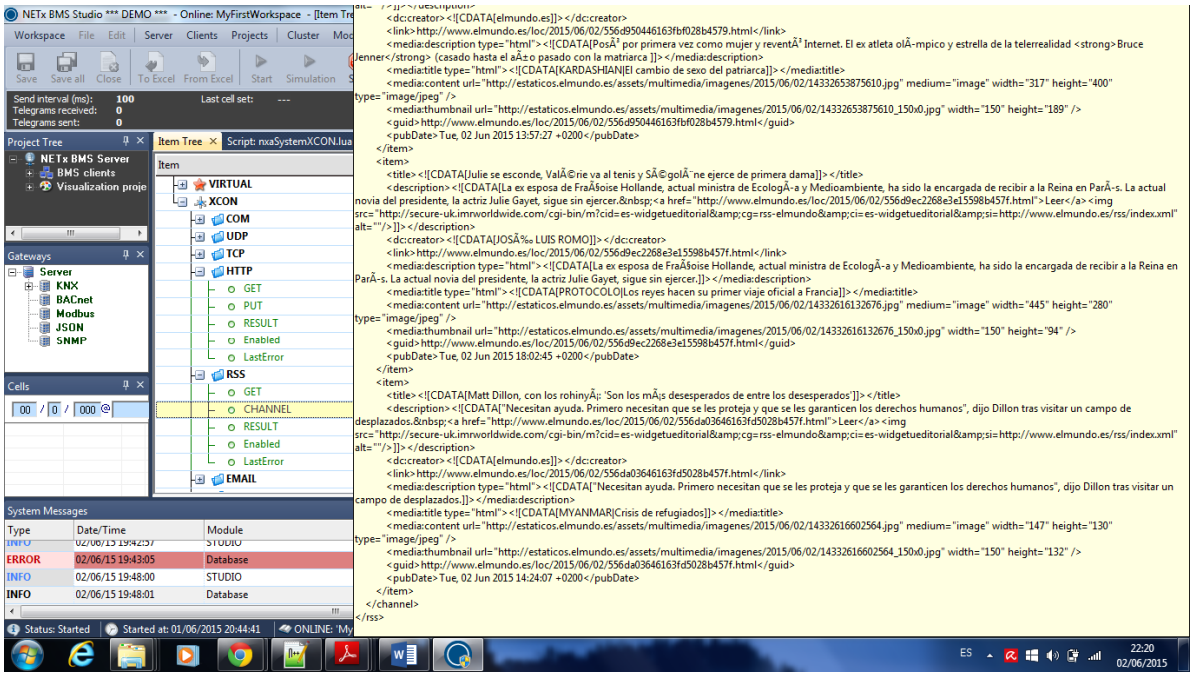
nxa.LogInfo -- lo que se desea mostrar por la consola System Messages

También es necesario modificar el script nxaDefinitions.lua, si queremos que nuestro script pueda ser ejecutado mediante la herramienta Execute LUA script deberemos incluir el script creado en este mediante el comando require tal y como podemos observar en la siguiente captura.



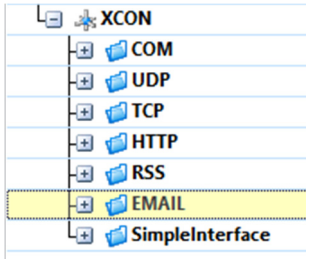
Por último, mostraremos el resultado de ejecutar dicho script viendo cómo se rellenan los campos de la interfaz gráfica por sí mismos.


NETx BMS Studio - <ONLINE: 'MyFirstWorkspace'> - [Item Tree: NETx\XCON]
Workspace Edit Server Clients Projects Cluster Modules Extensions KNX BACnet Modbus JSON Tools Windows Info
Save All To Excel From Excel Start Simulation Shutdown Items Tree Project Tree Graph Search Edit Script Remote
Send Interval (ms) : 250
Telegrams Received : 0
Telegrams Sent : 0
Last Cell Set : ---
NETx
Script: http.lua Script: nxaDefinitions.lua Script: nxaSystemXCON.lua Item Tree: NETx\XCON Script: rss.lua
show item values
NETx
XIO
Cluster
Module
API
Server
Today
Geo
Custom
Aliases
VAR
XCOMMAND
VIRTUAL
XCON
COM
UDP
TCP
HTTP
RSS
[GET] - = (www.dte.us.es)
[CHANNEL] - = (-1)
[RESULT] - = (<!DOCTYPE html PUBLIC| "-//W3C//DTD XHTML 1.0 Transitional//EN"| "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">|||<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">|| <
[Enabled] - = (Verdadero)
[LastError] - = ( )
EMAIL
SimpleInterface
Execute Script
1 getRSS("www.dte.us.es")
Cancel
Execute


NETx BMS Studio *** DEMO *** - Online: MyFirstWorkspace
Workspace File Edit Server Clients Projects Cluster
Save Save all Close To Excel From Excel Start Simulation
Send interval (ms): 100
Telegrams received: 0
Telegrams sent: 0
Last cell set: ---
Project Tree
NETx BMS Server
BMS clients
Visualization proje
Item Tree
Script: nxaSystemXCON.lua
Item
VIRTUAL
XCON
COM
UDP
TCP
HTTP
GET
PUT
RESULT
Enabled
LastError
RSS
GET
CHANNEL
RESULT
Enabled
LastError
EMAIL
Gateways
Server
KNX
BACnet
Modbus
JSON
SNMP
Cells
System Messages
Type Date/Time Module
ERROR 02/06/15 19:43:05 Database
INFO 02/06/15 19:48:00 STUDIO
INFO 02/06/15 19:48:01 Database
Status: Started
Started at: 01/06/2015 20:44:41
ES
22:20
02/06/2015

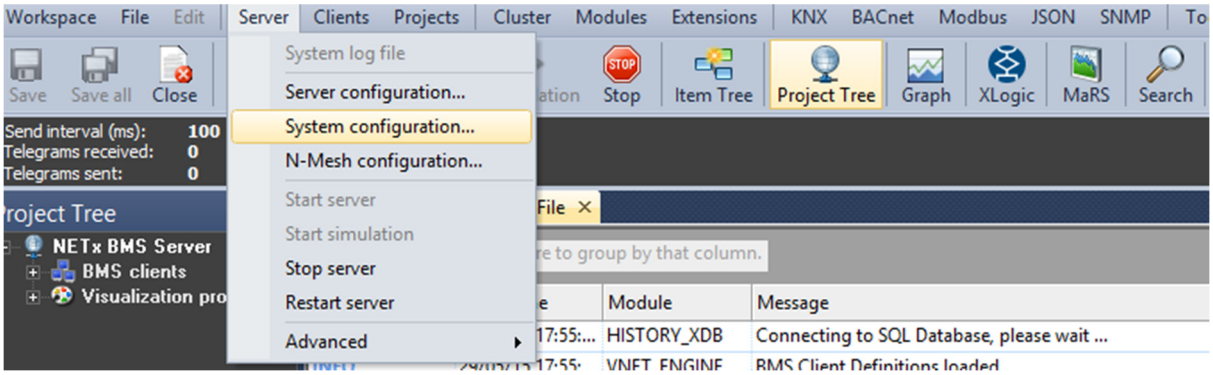
14 XCON: INTERFAZ EMAIL

El objetivo de este manual es describir el funcionamiento y configuración de la opción EMAIL dentro del interfaz XCON.

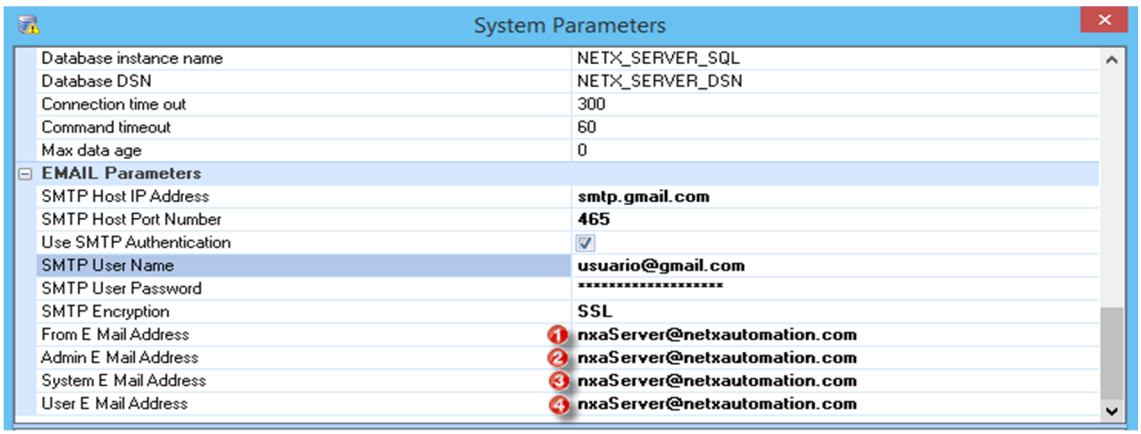


14.1 CONFIGURACIÓN DE LA INTERFAZ EMAIL

Para poder utilizar la función primero deberemos configurar el servidor de correo SMTP en la pestaña de configuración del BMS Server.



En este caso, vamos a modificar los parámetros para un servidor de correo Gmail. Hay que tener en cuenta que el puerto 465 corresponde con el sistema de encriptación SSL y que debemos sustituir el usuario por el email desde el que queremos enviar los emails.

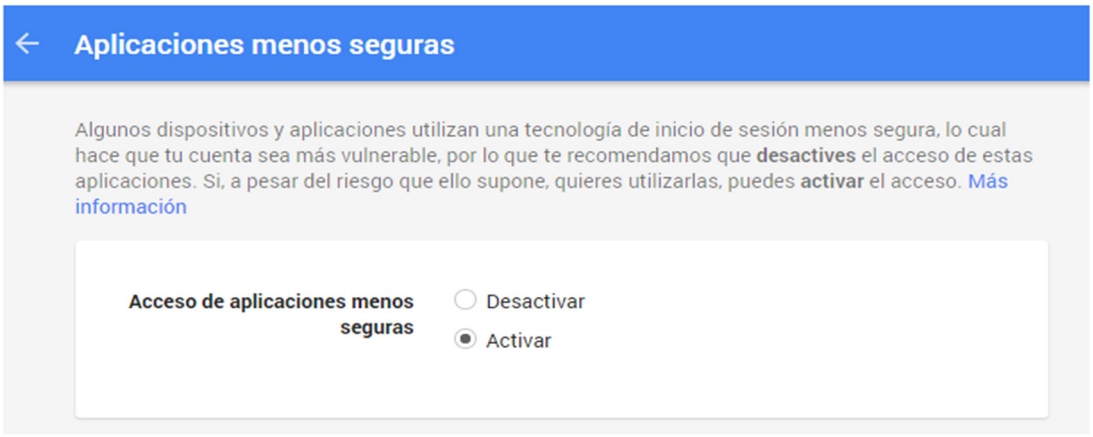


1. Email desde el que queremos enviar la notificación.
2. Email de destino al administrador del programa.
3. Email de destino al instalador del sistema.
4. Email de destino al usuario del sistema.

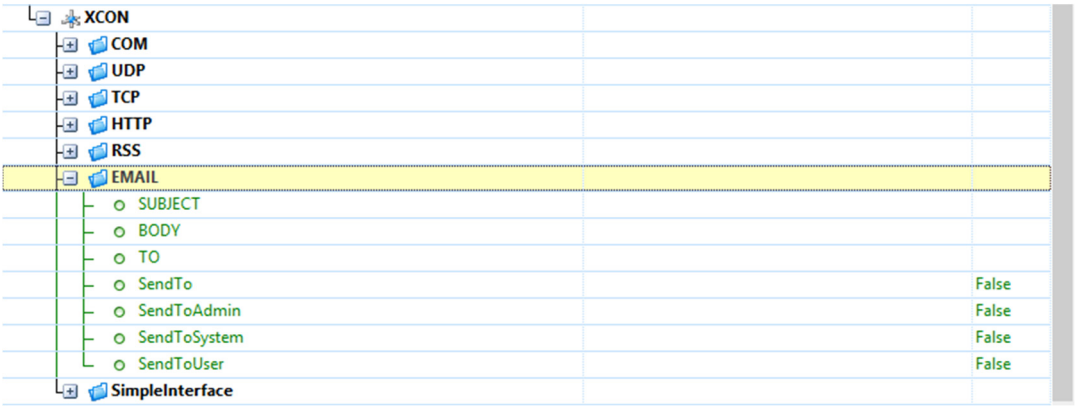
Una vez que hemos configurado el servidor de correo con el que queremos trabajar, ya estamos preparados para enviar notificaciones por email a los distintos interesados de la instalación.

Antes de poder enviar notificaciones, el SMTP de Gmail nos puede pedir que habilitemos el acceso a aplicaciones menos seguras. Al intentar enviar un correo desde el BMS Server, recibiremos un email, en la dirección que hemos configurado, alertándonos de que una aplicación "no segura" intenta conectarse. Si no es así, encontraremos la información en el siguiente enlace:

https://www.google.com/settings/security/lesssecureapps



Cambiando la configuración para permitir el acceso del programa al servidor, tendríamos todo listo para empezar a enviar las notificaciones. El siguiente paso es escribir en los campos de EMAIL el mensaje que queremos enviar y quién es su destinatario. Por defecto nos encontramos la configuración que aparece en la siguiente imagen:



- El campo SUBJECT es una cadena de caracteres donde se escribe el asunto del mensaje.
- El campo BODY es una cadena de caracteres donde se escribe el contenido del mensaje.
- El campo TO es una cadena de caracteres donde podemos escribir distintos emails para enviar el mensaje a varios destinatarios.
- El campo SendTo es un dato de tipo booleano, un 0 indica que no se hace nada mientras que un 1 manda la orden de enviar un email a los usuario definidos en TO.
- El campo SendToAdmin es un dato de tipo booleano, un 0 indica que no se hace nada mientras que un 1 manda la orden de enviar un email al usuario definido en la configuración como Admin (Imagen 3).
- El campo SendToAdmin es un dato de tipo booleano, un 0 indica que no se hace nada mientras que un 1 manda la orden de enviar un email al usuario definido en la configuración como System (Imagen 3).
- El campo SendToAdmin es un dato de tipo booleano, un 0 indica que no se hace nada mientras que un 1 manda la orden de enviar un email al usuario definido en la configuración como User (Imagen 3).

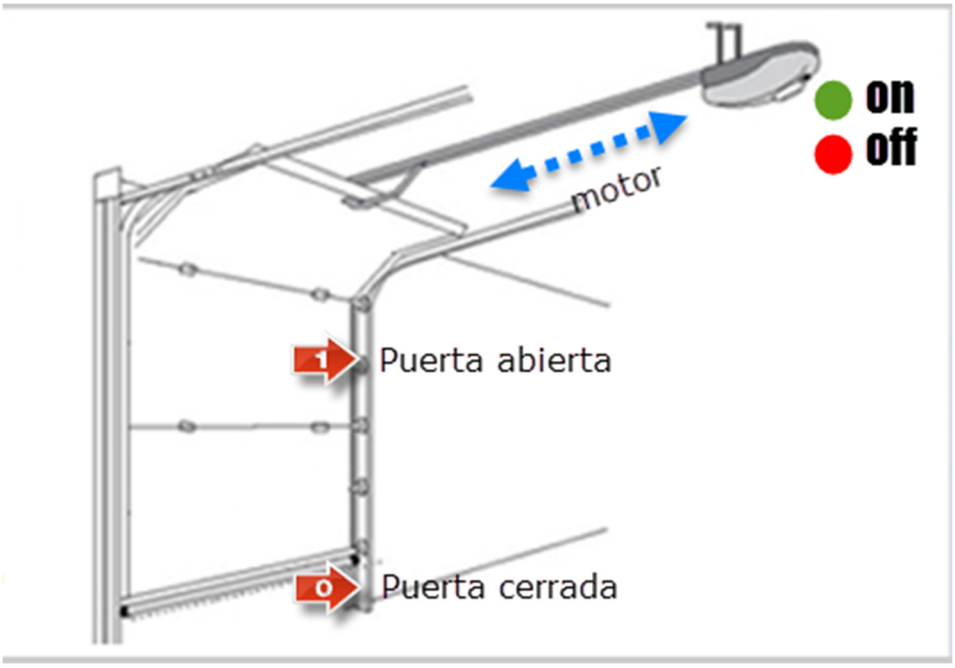
Para editar manualmente cada uno de los campos, bastará con hacer doble click con el ratón sobre el campo que queramos modificar. Finalmente para enviar el email debemos rellenar los campos SUBJECT y BODY. Después escribiremos un 1 en el campo correspondiente a la dirección de correo que queremos enviárselo. Si queremos enviarlo a varios destinatarios también es necesario rellenar el campo TO.

14.2 EJEMPLO DE ENVÍO DE EMAILS A TRAVÉS DE UN SCRIPT

Para entender el funcionamiento real del envío de mensajes vía email se ha escrito un script, en lenguaje de programación LUA, con un ejemplo de aplicación real. El programa describe de manera sencilla el funcionamiento, en caso de fallo, de un sistema con puerta de garaje automática.

DESCRIPCIÓN DEL PROGRAMA

El sistema de funcionamiento de una puerta de garaje automática se simula de manera sencilla con dos variables: una para el sensor de apertura-cierre y otra para el motor de funcionamiento de la puerta. Para poder utilizar el máximo de funciones del interfaz EMAIL se ha añadido un indicador que muestra al administrador si el sistema está funcionando correctamente o no (apertura indebida de caja del motor).



Como se ve en la imagen anterior, cuando la puerta está cerrada, el sensor de la puerta vale 0. Para abrir la puerta se acciona el motor con sentido hacia la derecha, recogiendo la puerta. Así, una vez que el sensor de posición infrarrojo deja de estar activo, se convierte en un 1 lo que indica que la puerta está abierta. Al llegar al final del recorrido del motor, la puerta permanecerá abierta hasta que el vehículo termine de pasar o pasen 15 segundos. Entonces comenzará a cerrarse, sentido hacia la izquierda, hasta llegar a la posición final de puerta cerrada (sensor infrarrojo de nuevo en 0).

Si el sistema funciona bien no es necesario el envío de ningún email. Sin embargo, se han planteado tres tipos de problemas distintos por los que podría estar fallando:

1. Indicador de funcionamiento on-off, comunica al administrador del sistema si la caja del motor ha sido manipulada sin consentimiento.
2. Funcionamiento incorrecto del motor, el sentido de giro y el estado del sensor no son coherentes.
3. La puerta del garaje permanece abierta durante un largo periodo de tiempo.

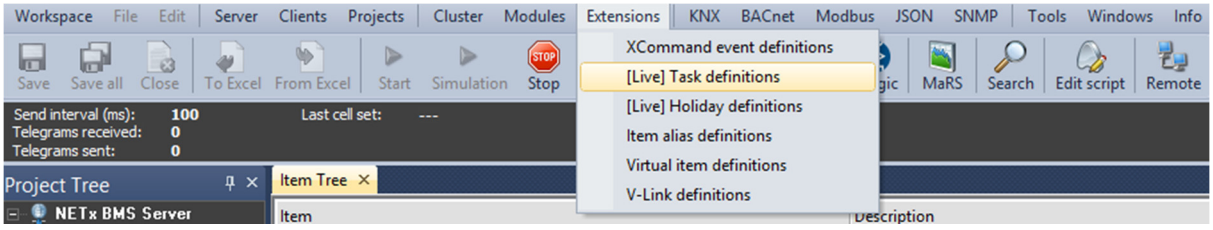
En función del tipo de error, el sistema envía un email de notificación a un usuario distinto. 1-administrador, 2-técnico del sistema y 3-usuario o cliente.

CONFIGURACIÓN DE LAS VARIABLES EN EL BMS SERVER STUDIO

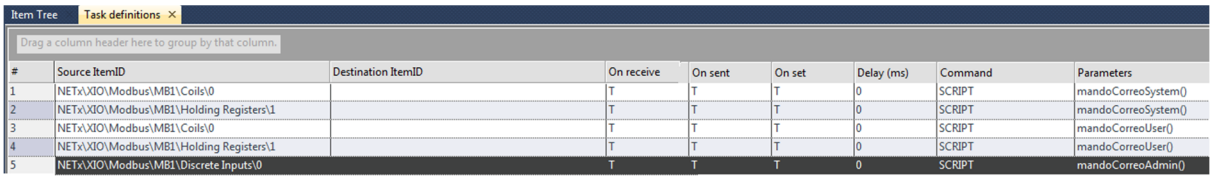
Para simular el funcionamiento del sistema hay que configurar algunos registros del NETx BMS Studio:

- Un Holding Register para simular el funcionamiento de un motor bidireccional (-500, 0, 500).
- Un Coil para simular el funcionamiento del sensor de puerta abierta-cerrada.
- Un Discrete Input para simular el funcionamiento del sensor de caja de motor abierta sin autorización.

Las tareas se definen en la pestaña que aparece en la siguiente imagen.

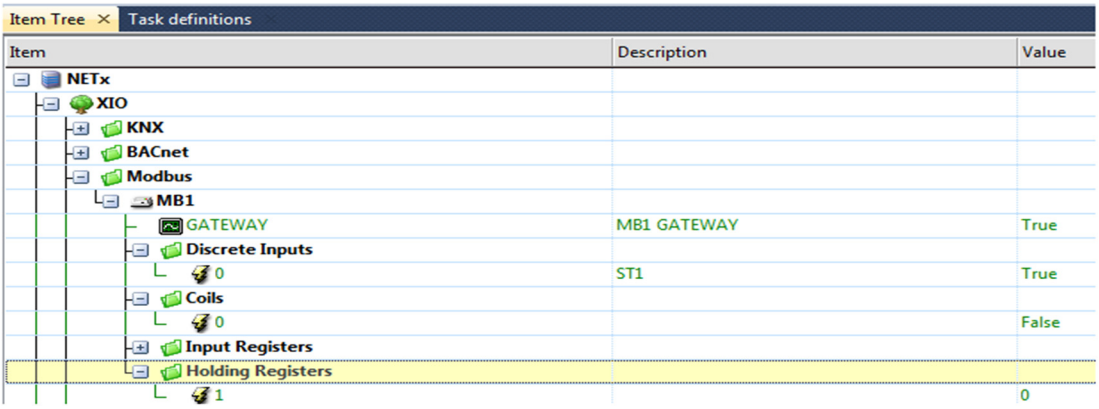


El cambio de valor en las variables ModBus *NETx\XIO\Modbus\MB1\Coils\0 ó en NETx\XIO\Modbus\MB1\Holding Registers\1* puede afectar en dos funciones distintas (*mandoCorreoSystem()* y *mandoCorreoUser()*), mientras que la variable *NETx\XIO\Modbus\MB1\Discrete Inputs\0* solo afecta a la función *mandoCorreoAdmin()*. Por claridad, se repiten las dos primeras variables en la definición de tareas, aunque también se podía haber incluido la llamada a ambas funciones en una misma fila.



#	Source ItemID	Destination ItemID	On receive	On sent	On set	Delay (ms)	Command	Parameters
1	NETx\XIO\Modbus\MB1\Coils\0		T	T	T	0	SCRIPT	mandoCorreoSystem()
2	NETx\XIO\Modbus\MB1\Holding Registers\1		T	T	T	0	SCRIPT	mandoCorreoSystem()
3	NETx\XIO\Modbus\MB1\Coils\0		T	T	T	0	SCRIPT	mandoCorreoUser()
4	NETx\XIO\Modbus\MB1\Holding Registers\1		T	T	T	0	SCRIPT	mandoCorreoUser()
5	NETx\XIO\Modbus\MB1\Discrete Inputs\0		T	T	T	0	SCRIPT	mandoCorreoAdmin()

Los valores de las variables que indican un funcionamiento correcto del sistema se muestran en la siguiente imagen.



CÓDIGO EJEMPLO SCRIPT LUA PARA ENVÍO DE EMAILS

A continuación se puede observar el código realizado en un fichero .lua que es leído y utilizado por el proyecto NETx BMS realizado:

```
--[[============================================================================

NETxAutomation Software GmbH - all rights reserved
nxaScriptEngine is using LUA $ 1 scripting language (http://www.lua.org/)

--! @nxaEmailConf.lua
--! @brief Este script contiene funciones para el envío de correos a los diversos tipos de usuarios del sistema NETx. Este sistema controla el uso de una puerta de garaje configurado con tres puntos de control.

1) La seguridad de la caja sellada del motor se realiza mediante el punto NETx\XIO\Modbus\MB1\Discrete Inputs\0 que comunicará al Administrador que ha sido manipulada indebidamente.

2) Cuando el motor de la puerta de garaje funcione incorrectamente, esto es cuando:
- El motor siga intentando cerrar la puerta del garaje y está ya esté completamente cerrada.
- El motor siga intentando abrir la puerta del garaje y está esté completamente abierta.
Esto se llevará a cabo con el uso de dos puntos:
NETx\XIO\Modbus\MB1\Coils\0 controla que puerta esté cerrada (false) ó abierta (true).
NETx\XIO\Modbus\MB1\Holding Registers\1 controla la velocidad del motor.
> 0 está abriendo la puerta.
< 0 está cerrando la puerta.
Al suceder alguno de estos casos, se le comunicará al instalador del sistema para que pueda subsanar el fallo.

3) En este caso, se le indica al usuario del sistema que la puerta del garaje se ha quedado abierta. Esto se controla con los dos puntos indicados en el caso anterior para cuando esté abierta y el motor no se esté moviendo (velocidad == 0).
--============================================================================]]
--
[[============================================================================
--! @brief Funcion que manda un correo al Administrador cuando se dan las condiciones del caso 1
--============================================================================]]
function mandoCorreoAdmin()
        nxa.SetValue("NETx.XCON.EMAIL.SUBJECT", "Notificación Centro de Control NETx")
        if(nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Discrete Inputs\\0") == false) then
                nxa.LogInfo('Mando correo Admin');
                nxa.SetValue("NETx.XCON.EMAIL.BODY", "Error en el direccionamiento del Motor
```

```
de puerta garaje.."..."Apertura de caja de control de motor no autorizada")

xcon.SendAdminEmail(nxa.GetValue("NETx.XCON.EMAIL.SUBJECT"),nxa.GetValue("NETx.XCON.EM
AIL.BODY"));
        end

end

--[[=========================================================================
--! @brief Funcion que manda un correo al Instalador cuando se dan las condiciones del caso 2
--=========================================================================]]
function mandoCorreoSystem()
        nxa.SetValue("NETx.XCON.EMAIL.SUBJECT", "Notificación Centro de Control NETx")

        if(nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Coils\\0") == false
                and
                nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Holding Registers\\1") < 0) then
                        nxa.LogInfo('Mando correo System1');
                        nxa.SetValue("NETx.XCON.EMAIL.BODY", "Fallo de funcionamiento en
puerta de garaje. Motor activado con puerta cerrada")

        xcon.SendSystemEmail(nxa.GetValue("NETx.XCON.EMAIL.SUBJECT"),nxa.GetValue("NETx.X
CON.EMAIL.BODY"));
        end
        if(nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Coils\\0") == true
                and
                nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Holding Registers\\1") > 0) then
                        nxa.LogInfo('Mando correo System2');
                        nxa.SetValue("NETx.XCON.EMAIL.BODY", "Fallo de funcionamiento en
puerta de garaje. Motor activado con puerta abierta")

        xcon.SendSystemEmail(nxa.GetValue("NETx.XCON.EMAIL.SUBJECT"),nxa.GetValue("NETx.X
CON.EMAIL.BODY"));
        end

end

--[[=========================================================================
--! @brief Funcion que manda un correo al Usuario cuando se dan las condiciones del caso 3
--=========================================================================]]
function mandoCorreoUser()
        nxa.SetValue("NETx.XCON.EMAIL.SUBJECT", "Notificación Centro de Control NETx")

        if(nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Coils\\0") == true
                and
                nxa.GetValue("NETx\\XIO\\Modbus\\MB1\\Holding Registers\\1") == 0) then
                        nxa.LogInfo('Mando correo User');
                        nxa.SetValue("NETx.XCON.EMAIL.BODY", "Puerta del garaje ABIERTA")

        xcon.SendUserEmail(nxa.GetValue("NETx.XCON.EMAIL.SUBJECT"),nxa.GetValue("NETx.XC
ON.EMAIL.BODY"));
        end

end
```