NXA
FUNCIONES PARA EXTRACCIÓN DE DATOS

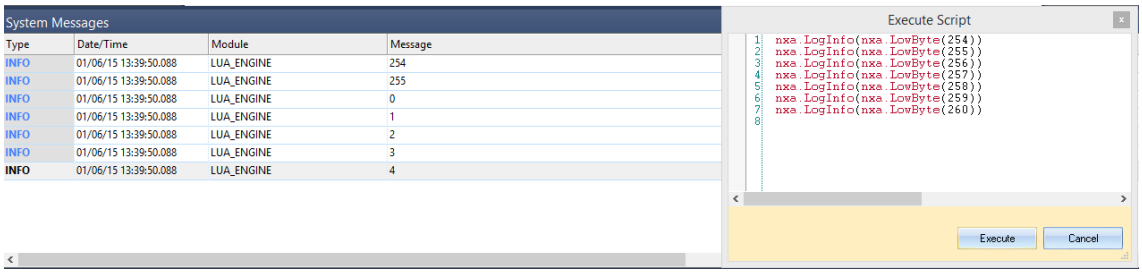
4.6.5. Información general acerca de las funciones nxa de LUA para extraer datos:

nxa.LowByte

Obtener el byte menos representativo del valor de entrada.

Parámetros:
Número: valor de entrada

Returns:
Número: byte menos representativo del valor de entrada



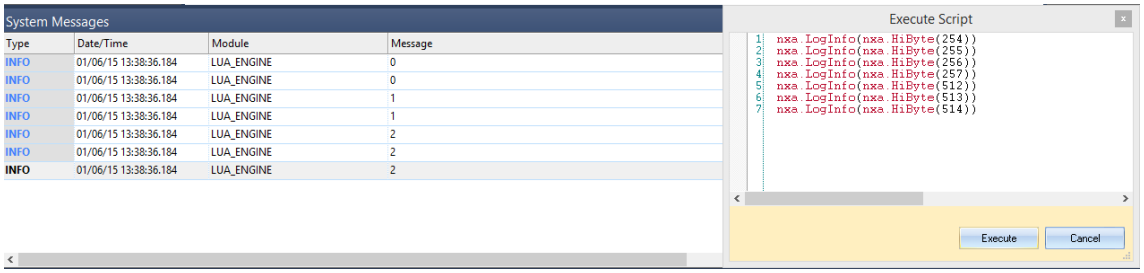
Como podemos ver en el ejemplo, en los 2 primeros casos son valores de menos de 256 (valores que se pueden representar con 1 byte), y en el siguiente te devuelve 0, ya que necesita otro bit para representarlo y los primeros 8 bits (el primer byte) son cero. A partir de este vemos que devuelve el byte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

Returns:
Número: byte más representativo del valor de entrada.



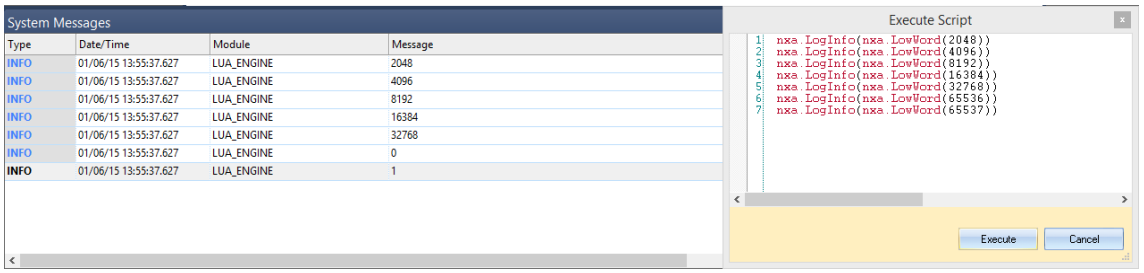
En el ejemplo podemos ver como hasta 256 el valor se representa con 1 solo byte, pero cuando pasa de 256 devuelve 1, ya que el bit que necesita para representarlo ya pertenece al byte más representativo, el cual queremos mostrar. Vemos que ocurre lo mismo al pasar de 512, múltiplo de 256, y se pone a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

nxa.LowWord

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

Parámetros:
Número: valor de entrada

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



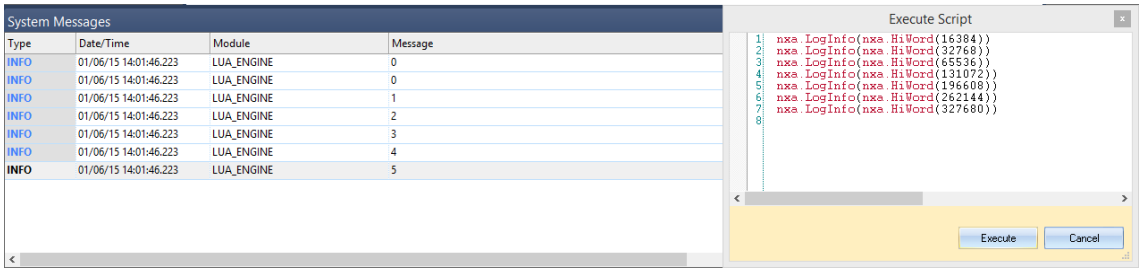
El ejemplo es similar al método **nxa.LowByte**, pero en este caso en vez de un byte es una palabra de 16 bits, por lo que ocurre lo mismo pero a partir del valor 65536. Vemos que con este valor se pone a 0 y a partir de este devuelve el valor de entrada menos 65536 o múltiplos del mismo.

nxa.HiWord

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

Parámetros:
Número: valor de entrada

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



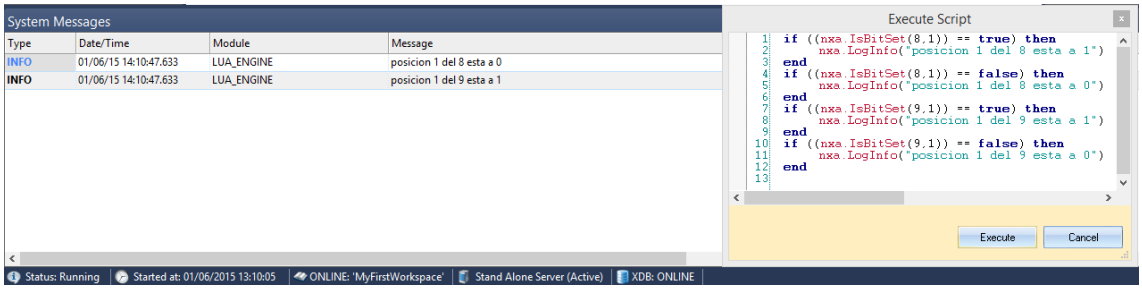
El ejemplo es similar a **nxa.HiByte**, donde vemos que cuando pasa de 65536 devuelve 1, ya que el bit que necesita para representarlo ya pertenece al byte más representativo, el cual queremos mostrar. Vemos que, como se explica el ejemplo de **nxa.HiByte**, va cambiando con los múltiplos de 65536.

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

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



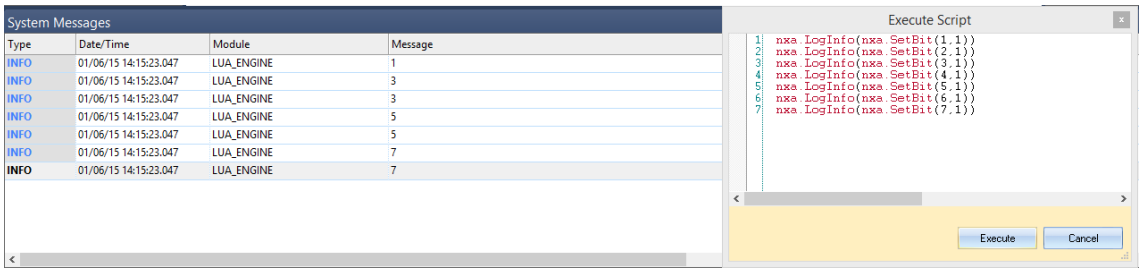
Podemos ver en este sencillo ejemplo donde comprobamos de los valores 8 (1000) y 9 (1001) la posición 1. Como vemos, comprobamos si con el método nos devuelve verdadero o falso, si nos devuelve verdadero nos devuelve “posición 1 del X está a 1”, y si nos devuelve falso “posición 1 del X está a 0”. Vemos 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

Returns:
Número: resultado al cambiar el bit



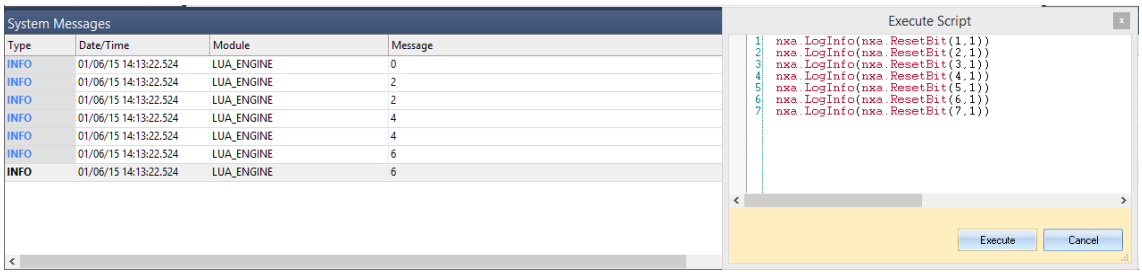
Como podemos ver en este ejemplo, hemos cambiado el bit 1 de los valores del 1 al 7. Como se puede comprobar, solo cambia los valores 2, 4 y 6, y esto sucede porque los valores 1, 3, 5 y 7 ya tienen el bit 1 a 1, y no tienen ningún cambio.

nxa.ResetBit

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

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

Returns:
Número: resultado al cambiar el bit



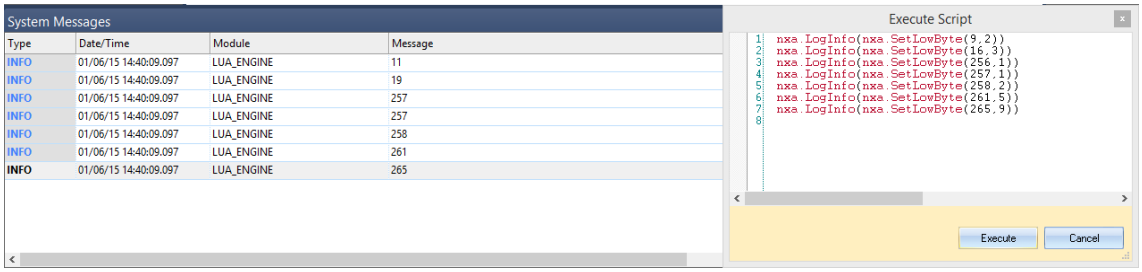
El ejemplo es similar el del método **nxa.SetBit**, pero en este caso en vez de poner el bit a 1, lo ponemos a 0. Vemos que en este caso los que cambian son los valores 1, 3, 5, y 7 ya que son los que tienen el bit 1 a 1 y los pone a 0. Los valores 2,4 y 6 se quedan igual ya que el bit 1 ya está a 0.

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.

Returns:
Número: resultado del método OR de los dos valores.



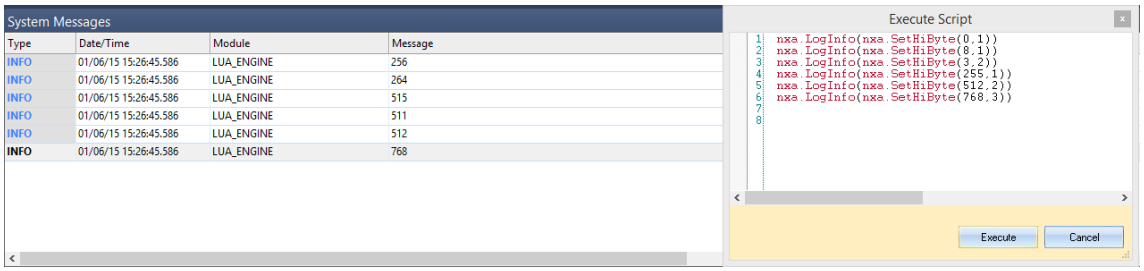
Como podemos ver en el ejemplo, al principio puede parecer una suma de valores ya que 9 (1000) al aplicarle este método con 2(10) te sale la suma de ambos, 11(1010). Pero si vemos en los siguientes ejemplos podemos comprobar que si le aplicamos un byte X, a un número cuyo byte más bajo es este mismo 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.

Returns:
Número: resultado del método OR de los dos valores.



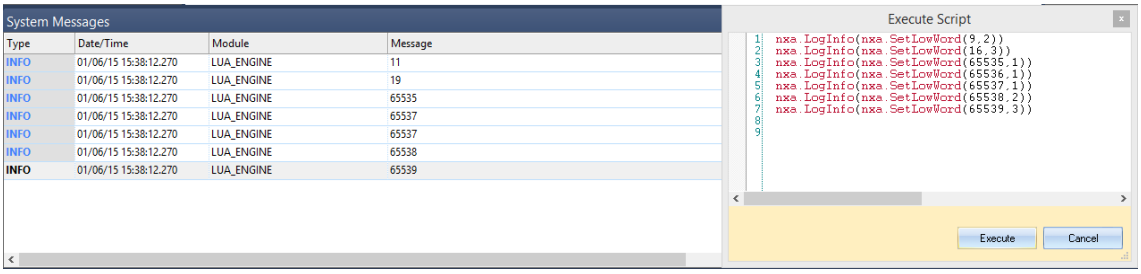
Como vemos en el ejemplo, el método es igual que **nxa.SetLowByte**, pero en este caso se hace con el byte más representativo del valor. Como podemos ver en el ejemplo, en el caso de 512 (1000000000), al aplicarle el método con el 2 (10), el byte más significativo es 10, por lo que el valor no cambia.

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.

Returns:
Número: resultado del método OR de los dos valores.



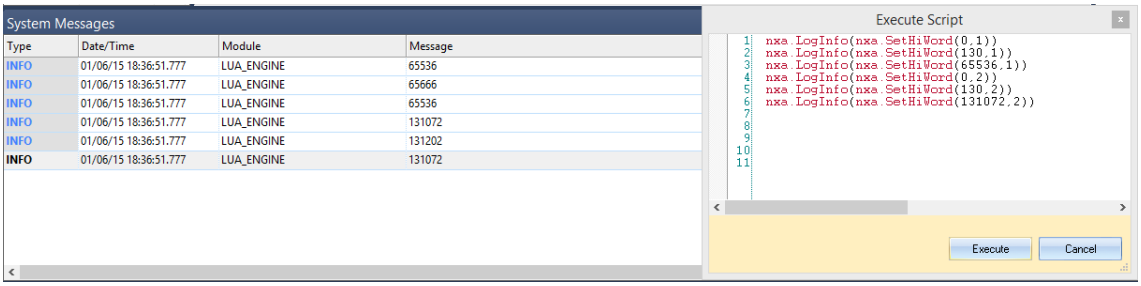
Como podemos ver, el ejemplo es similar a **nxa.SetLowByte**, solo que en vez de 1 byte (8 bits) es una palabra (16 bits). De hecho podemos ver que hemos usado los mismos ejemplos al principio, y puede parecer una suma de valores ya que 9 (1000) al aplicarle este método con 2(10) te sale la suma de ambos, 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.

Returns:
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.