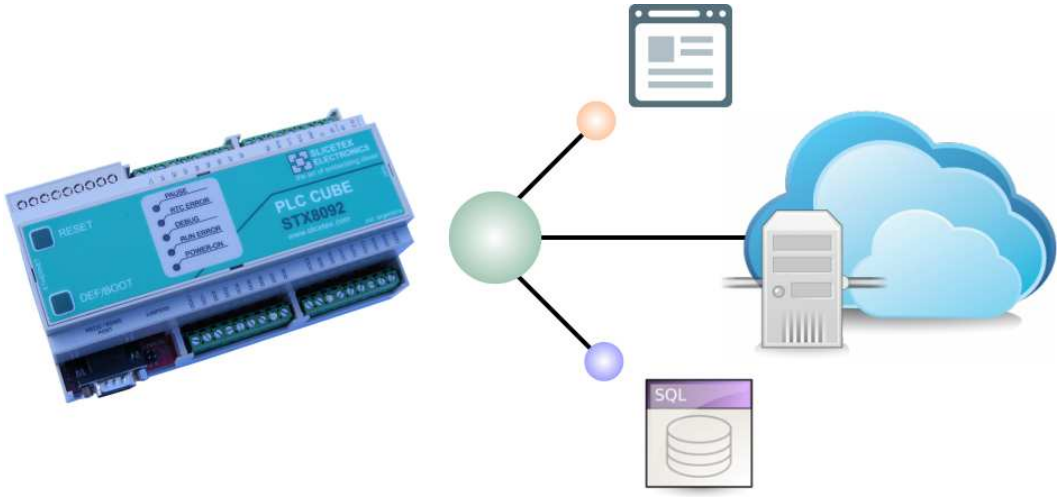
Slicetex Ladder Designer Studio

NOTA DE APLICACIÓN

AN032
Cliente Web (HTTP) para PLC

Autor: Ing. Boris Estudiez



Modelos Aplicables	AX, CX y DX

1 Descripción General

La presente nota de aplicación explica cómo utilizar el Cliente Web disponible en nuestros PLC.

El Cliente Web (HTTP) mediante peticiones GET y POST puede transferir información a un servidor web en internet, esto le permitirá de forma simple almacenar, registrar y mostrar datos en una página web. Al mismo tiempo podrá leer la respuesta del servidor web para realizar una comunicación bidireccional entre el PLC y el sitio web.

Mediante la transferencia de información a una página web, es posible trasladar los datos del PLC (mediciones analógicas, eventos, etc) a la nube on-line. Utilizando herramientas web como PHP, ASP, Perl o Java desde el lado servidor, podrá manejar los datos enviados desde el PLC y realizar acciones complejas como escribir una base datos SQL o almacenar información en archivos de registros.

2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el software StxLadder y el PLC adquirido. Sugerimos leer los siguientes documentos:

1. Manual de Usuario del software StxLadder.
2. Manual de Programación Pawn del PLC (si utiliza lenguaje Pawn)
3. Hoja de datos técnicos del PLC.

Mas documentación puede encontrar en la página del producto: www.slicetex.com.

Para consultas y soporte, ponemos a disposición un foro de discusión en: www.slicetex.com/foro donde puede leer preguntas de otros usuarios y realizar también sus propias preguntas.

2.1 ¿Cómo Leer esta Nota de Aplicación?

¡No se deje abrumar por la cantidad de hojas!, es muy fácil leer y comenzar a usar el **Cliente Web** si lo hace de la siguiente manera:

Recomendamos leer la sección 4 . **Teoría de Funcionamiento** en pag. 3 por completo.

- Si usa Lenguaje Ladder, continúe leyendo la sección 5 . **Ejemplo de Uso Inicial en Lenguaje Ladder** en la página 7. Al finalizar el ejemplo, se dan recomendaciones para continuar lectura.

- Si usa Lenguaje Pawn, continúe leyendo la sección 6 . **Ejemplo de Uso Inicial en Lenguaje Pawn** en la página 27. Al finalizar el ejemplo, se dan recomendaciones para continuar lectura.

3 Requerimientos

Para esta nota de aplicación, debe tener instalado en su computadora el entorno de Programación **StxLadder** (Slicetex Ladder) y utilizar un firmware actualizado con soporte **Cliente Web** en el PLC.

Se recomienda estar familiarizado con los conceptos básicos de los servidores web.

4 Teoría de Funcionamiento

Los PLC de Slicetex Electronics permiten configurarse como **Cliente Web** para realizar una conexión al servidor. La comunicación se realiza en general por el puerto Ethernet disponible en el dispositivo, o alguna otra interfaz compatible con Internet.

El PLC se comunica con el servidor Web utilizando el protocolo HTTP (Hypertext Transfer Protocol)

Una transacción típica **HTTP** (simplificada) se muestra a continuación en la Figura 1:

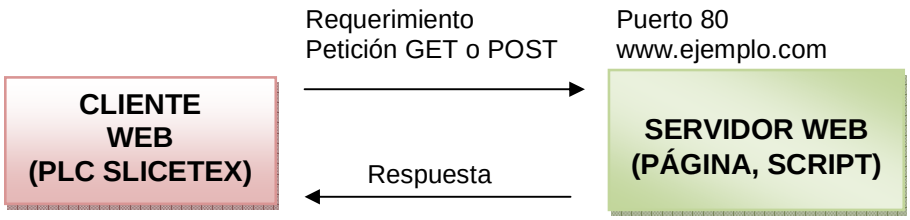


Fig. 1: Transacción HTTP.

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción HTTP para conectarse a un servidor web. Los pasos son los siguientes:

1. **Configuración**: El PLC debe configurar los parámetros para conectarse el servidor web, por ejemplo dirección, numero de puerto, timeout de inactividad, etc.
2. **Requerimiento**: El cliente envía un requerimiento para obtener una página web (recurso) o enviar datos con una petición tipo GET o POST.
3. **Respuesta**: El servidor procesa el requerimiento del cliente, obtiene los dados enviados por el PLC y devuelve una respuesta que depende de la pagina web, ya sea si esta es estática o dinámica (con información variable).
4. **Recepción:** El cliente recibe la respuesta y la procesa si es necesario.
5. **Fin:** El cliente puede realizar otro requerimiento al servidor si es necesario, volviendo al punto 2.

4.1 Peticiones GET y POST al Navegar

Un Cliente Web comúnmente puede enviar dos tipos peticiones para solicitar un recurso o página web:

1. Petición **GET**: La más utilizadas por los navegadores para cargar una página.
2. Petición **POST**: La más utilizada para enviar datos desde un formulario al servidor.

Estas peticiones están definidas profundamente por el protocolo HTTP (RFC 2616), pero solo es necesario entender el concepto general para nuestros fines de programación con el PLC.

Supongamos una página web dentro de un servidor con la siguiente dirección:

ejemplo.com/personas/contacto.html

Un Cliente Web (por ejemplo un navegador) para solicitar la página **contacto.html** al servidor **ejemplo.com**, normalmente realizaría una petición GET de la siguiente forma:

```
GET /personas/contacto.html
```

Notar como el recurso **contacto.html** se encuentra adentro de la carpeta **/personas**, por ello el cliente debe especificar la dirección completa del recurso dentro del servidor **ejemplo.com** (sin incluirlo, ya que está conectado a el servidor) pero incluyendo la barra invertida "**/**" inicial del directorio raíz.

Ahora, si un navegador quiere enviar información con datos de una persona (nombre y teléfono) a una página web tipo PHP mediante una petición GET, veríamos en el URL o barra de direcciones de nuestro navegador la siguiente dirección:

```
ejemplo.com/personas/datos.php?nombre=Rodya&telefono=0351153423793
```

Esto se traduce en una petición GET con la siguiente sintaxis:

```
GET /personas/datos.php?nombre=Rodya&telefono=0351153423793
```

Notar cómo se añaden dentro de la URL los datos de la persona con la cadena **nombre=Rodya&telefono=0351153423793** separada por un carácter "?" seguido al nombre del recurso **datos.php**.

El recurso **datos.php** es un archivo con lenguaje PHP (puede ser cualquier otro lenguaje, ej: ASP, Java, etc) que mediante algunas funciones especiales, podrá recuperar y procesar la cadena de datos:

nombre=Rodya&telefono=0351153423793

Separando los campos en:

nombre=Rodya
telefono=0351153423793

De esta manera, el servidor obtiene el nombre de la persona "**Rodya**" y su teléfono "**0351153423793**".

La cadena **nombre=Rodya&telefono=0351153423793** se llama "**Query String**" (más información en sección 4.2) y tiene un formato especial para codificar datos en la URL.

Alternativamente, si el Cliente Web no quiere que los datos a enviar aparezcan en la URL o barra de direcciones del navegador, utilizaría una petición del tipo **POST**.

En este caso el "**Query String**" se envía por separado al servidor y no en la ruta del recurso, volviendo al ejemplo anterior, pero realizando una petición POST, utilizaríamos la siguiente sintaxis:

```
POST /personas/datos.php
```

Y por separado (de forma no visible al navegador) enviaría el **Query String** con los datos:

```
nombre=Rodya&telefono=0351153423793
```

Como vemos, la petición POST esta optimizada para el envió de datos y además ofrece mayor seguridad porque no queda en el historial del navegador. Pero su uso, depende de que acepte el servidor web.

Tanto para una petición GET o POST, el servidor responderá con un código de respuesta, las más usuales son:

- 200 : Respuesta estándar para peticiones correctas.
- 301 : Movido permanentemente.
- 400 : Error en sintaxis.
- 404 : Recurso no encontrado.

Seguramente las haya visto en alguna oportunidad al navegar por internet.

Finalmente es posible que el servidor responda con un mensaje de respuesta, es decir con los datos del recurso solicitado (página web, archivo, texto, etc) o simplemente no responda con información alguna, cerrando la conexión.

Por ejemplo, luego de un GET o POST, puede responder:

```
Datos aceptados. Muchas gracias
```

4.2 Query String

Un "**Query String**" es la cadena de consulta o cadena con datos para enviar al servidor web mediante las peticiones **GET** y **POST** vistas con anterioridad.

La misma consiste en una cadena de la forma: **campo1=valor1&campo2=valor2&campoX=valorX**

En el **Query String** cada **valor** se especifica con un **campo**, separado por un "**=**". Si hay varios campos, los mismos se separan mediante el símbolo "**&**".

```
campo=valor
```

El **campo** en general es una palabra que identifica la naturaleza del **valor** a transmitir, y el **valor** es una cadena ya sea con datos numéricos o texto.

Ejemplo, enviar valor de RPM=598 y Temperatura=25.4:

```
RPM=598&Temperatura=25.4
```

Ejemplo, enviar Nombre “Ernesto Sabato” y Mail “sabato@gmail.com”:

```
Nombre=Ernesto+Sabato&Mail=sabato%40gmail.com
```

Algunos caracteres (ya sea en el campo o valor) deben codificarse en “porcentaje”, también llamado “**URL Encode**”, estos caracteres son el espacio (por +), la arroba (por %40), como en el ejemplo anterior.

Las letras (A-Z y a-z), números (0-9), y caracteres “*”, “-“, “_”, “.”, “~” no necesitan ser codificados ya que son caracteres **no reservados**. Para el espacio puede usarse (+) o (%20), pero para el resto se usa el signo "%" seguido del número hexadecimal ASCII correspondiente al carácter codificado.

En **StxLadder**, en el menú “**Herramientas > Conversión > URL Encode**” hay un conversor para convertir cadenas a formato URL encoded, de tal forma que le sea fácil enviar un **Query String** complejo.

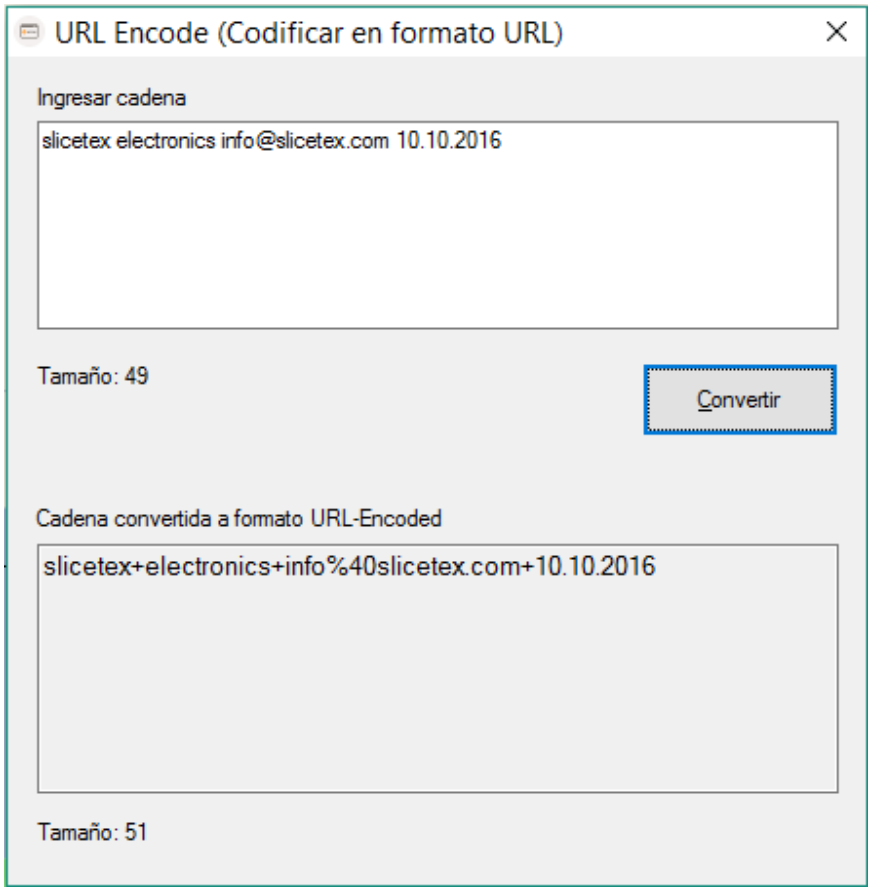


Fig. 2: Aplicación URL Encode de StxLadder

Recuerde **solo** convertir a formato URL encoded el **campo** o el **valor** a transmitir, porque los símbolos “&” y “=” usados para separar los valores deben preservarse.

5 Ejemplo de Uso Inicial en Lenguaje Ladder

Antes de comenzar con las definiciones de los componentes Ladder disponibles, empezaremos con un ejemplo básico para utilizar el Cliente Web rápidamente.

Nos detendremos en los puntos importantes para entender los conceptos claves a medida que avance el ejemplo.

En esta sección expondremos un ejemplo básico en lenguaje Ladder, si está usando lenguaje Pawn, puede ir directamente al ejemplo Pawn en la página 27.

Es altamente recomendado que practique este ejemplo, así podrá comprender las nociones fundamentales de uso del Cliente Web del PLC.

5.1 Crear el Proyecto

Comience un nuevo proyecto en **StxLadder** y seleccione el **modelo de PLC** que tenga adquirido.

Nota: El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendGet1.zip**

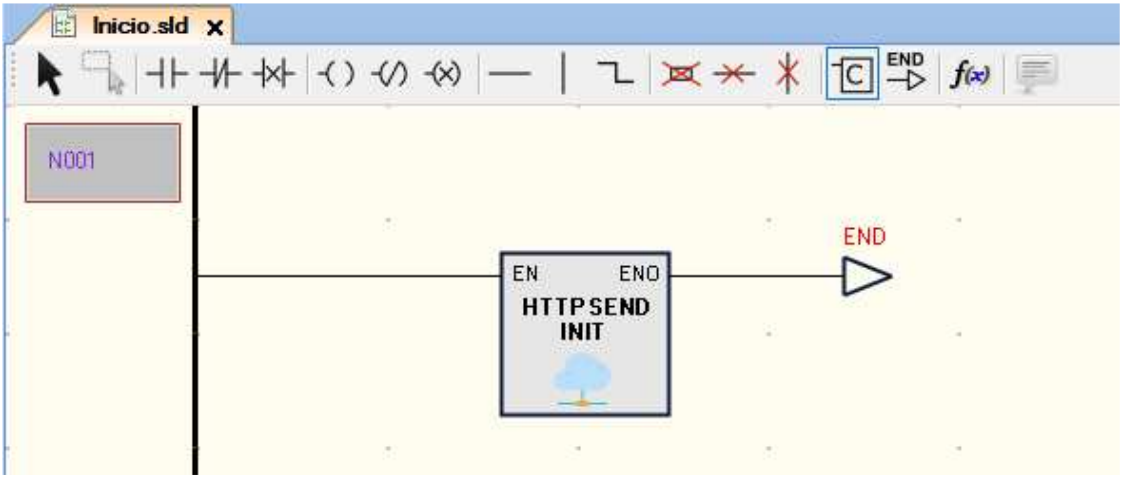
5.2 Objetivo del Proyecto

Este ejemplo tendrá como objetivo transmitir cada 30 segundos 4 números enteros mediante una petición GET a un servidor web de ejemplo, llamado **ejemplo.com**. Luego de una transmisión, se incrementan los valores de los 4 números enteros.

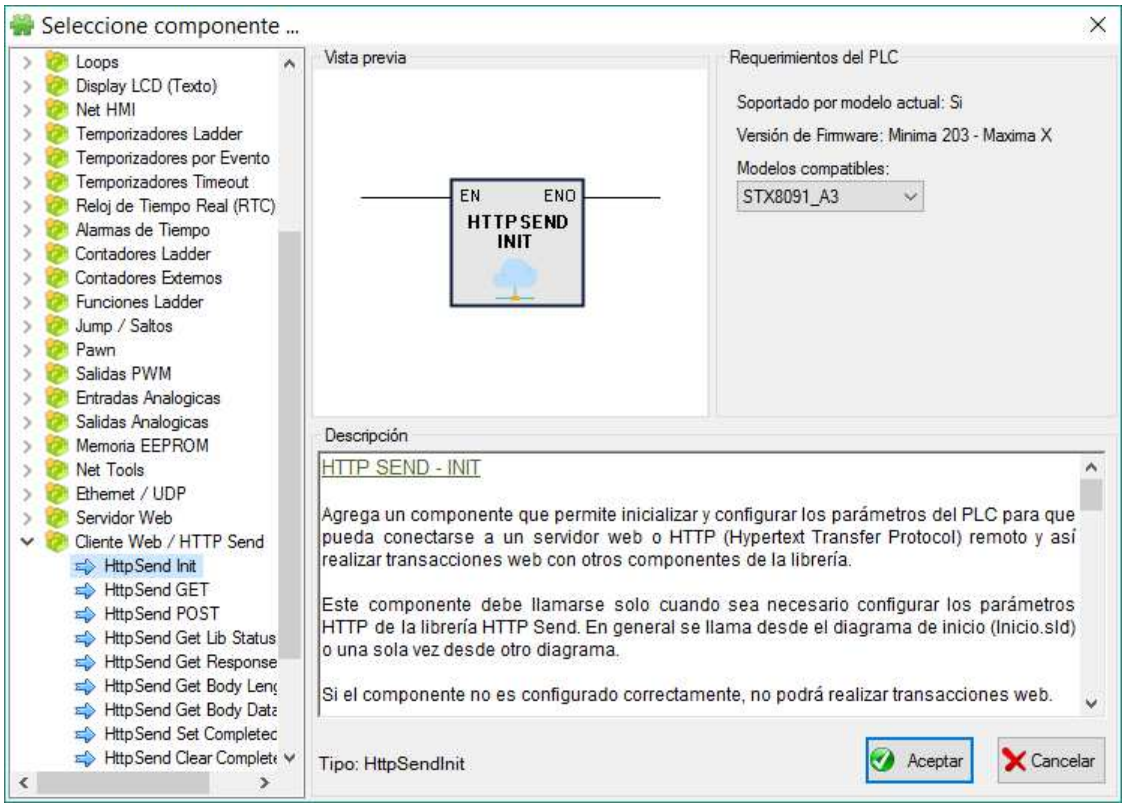
La dirección web completa del servidor de ejemplo será: **ejemplo.com/data/log.php**

5.3 Configurar Cliente Web

En el diagrama **Inicio.sld** del proyecto insertaremos el componente **HttpSendInit** en la network **N001** como se muestra a continuación:



El componente **HttpSendInit** se encuentra en el grupo "**Cliente Web / HTTP Send**" accesible mediante la herramienta para agregar de componentes Ladder, mostrado a continuación:



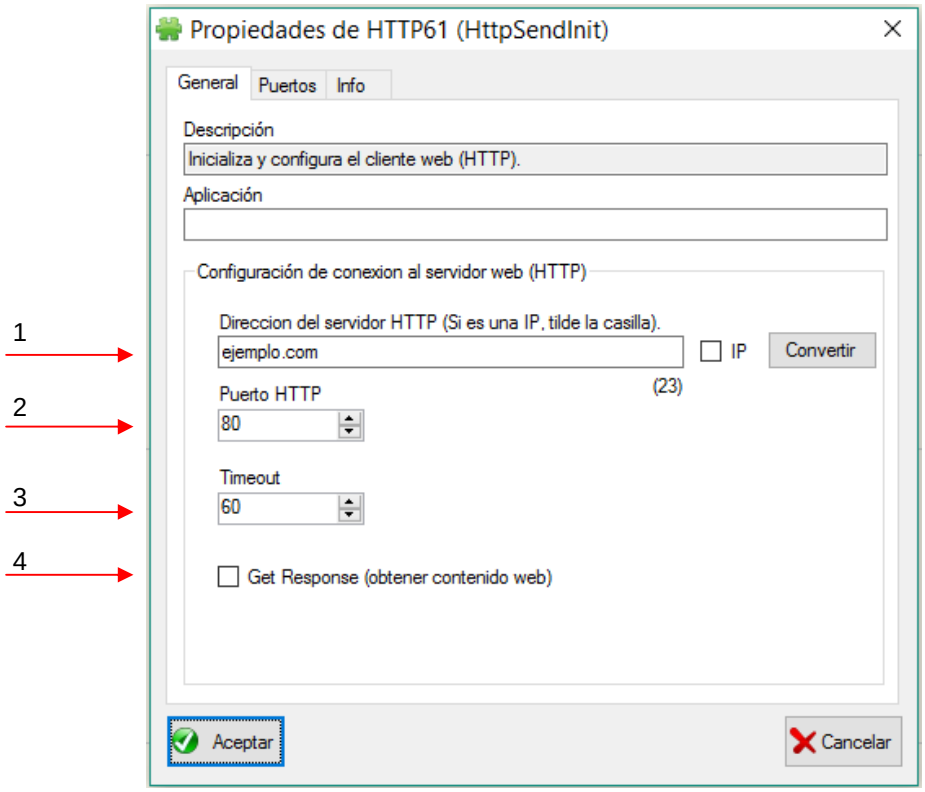
Notar como en el área de "**Descripción**" de la selección de componente se puede ver información de uso del componente, muy útil para tenerlo como referencia en sus proyectos.

El componente **HttpSendInit** lo colocamos en el diagrama **Incio.sld** porque solo configuraremos una sola vez el Cliente Web al inicio de nuestra aplicación, pero si deseamos cambiar la dirección u otros parámetros, podemos llamarlo nuevamente desde otro lugar del programa.

El próximo paso consiste en configurar las propiedades del componente con los siguientes parámetros del servidor web:

- **Dirección HTTP**: ejemplo.com
- **IP**: No seleccionar.
- **Puerto HTTP**: 80
- **Timeout**: 60
- **Get Response**: No seleccionar.

La ventana de propiedades del componente queda como se muestra en la siguiente figura:



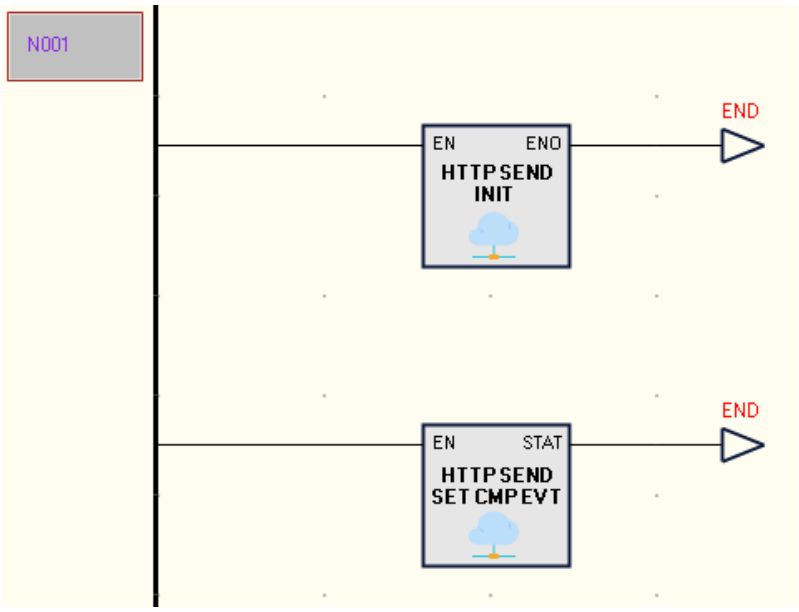
Notar los siguientes puntos señalados en la figura superior:

1. Dirección o nombre del servidor web (HTTP). En este caso utilizamos "**ejemplo.com**". Si es una dirección IP (por ejemplo 192.168.1.15) seleccionar la casilla **IP**.
2. Puerto TCP donde el servidor escucha peticiones de los clientes, por defecto 80.
3. Timeout o tiempo en segundos que el cliente espera una respuesta del servidor luego de la conexión al mismo.
4. Casilla "**Get Response**", seleccionar si estamos interesado en obtener el contenido del cuerpo del mensaje o recurso de respuesta del servidor web.

5.4 Crear Evento de Fin de Transacción HTTP

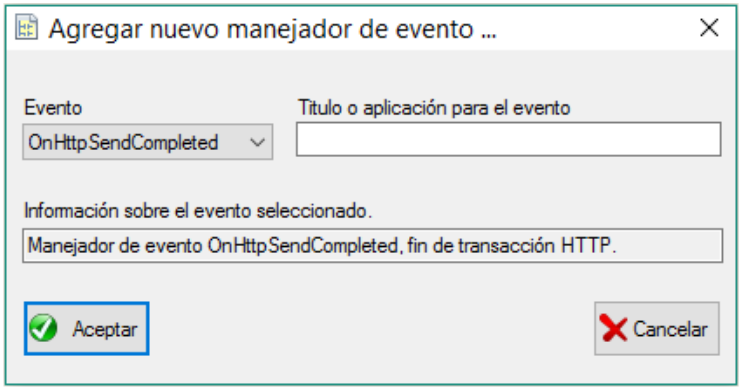
En el siguiente paso vamos a agregar el componente **HttpSendSetCompletedEvent** que permite activar el evento "**OnHttpSendCompleted**", el cual es un evento que llama al diagrama "**OnHttpSendCompleted.sld**" cuando la transacción HTTP fue completada (con o sin errores).

El componente lo insertamos en la network **N001** a continuación de **HttpSendInit** también:



Finalmente debemos agregar el diagrama **OnHttpSendCompleted.sld** desde el menú "**Proyecto > Agregar > Evento Ladder**".

En el siguiente dialogo, elija el evento **OnHttpSendCompleted** y haga click en el botón "**Aceptar**":



Ahora podrá ver en el explorador de proyecto en nuevo diagrama de evento creado. El mismo será llamado cuando la transacción HTTP finalice.

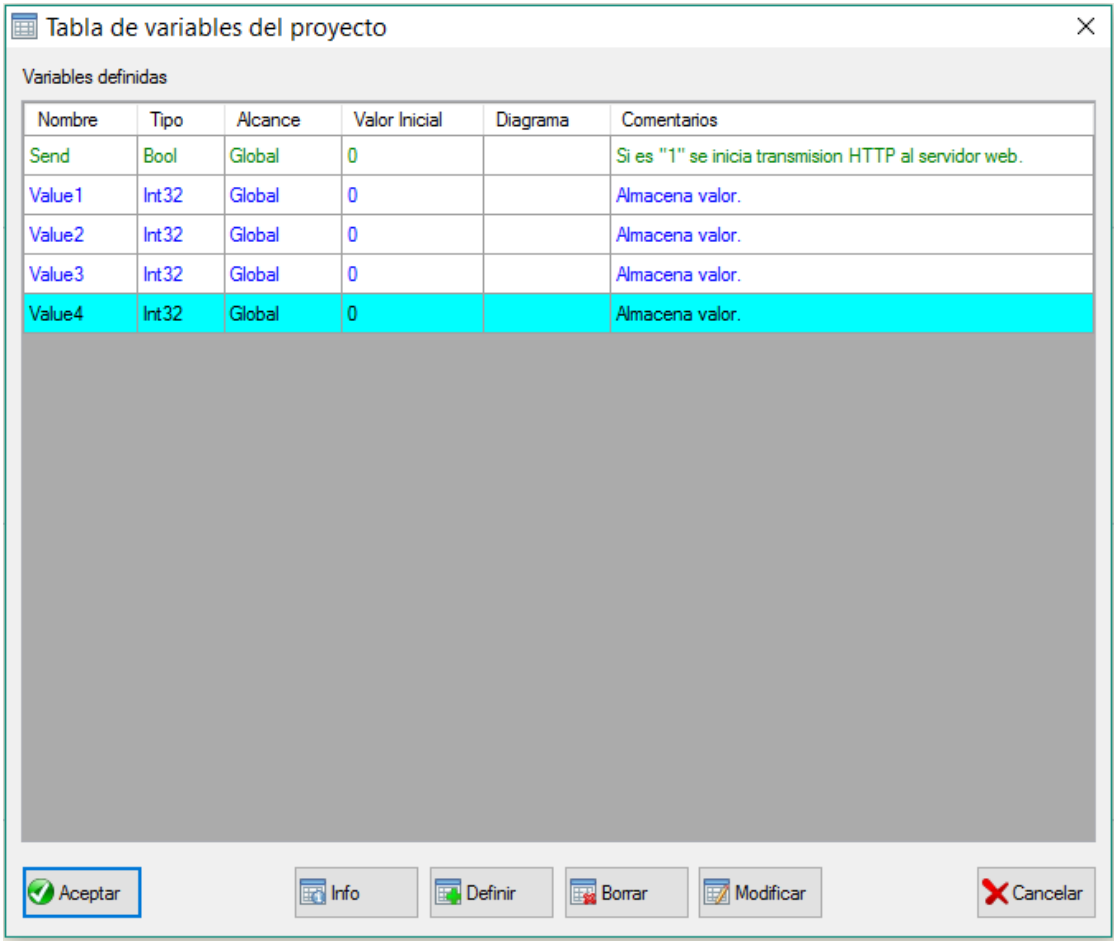
5.5 Crear Variables

En el siguiente paso vamos a crear variables para que el programa funcione de acuerdo a nuestro objetivo inicial (ver pág. 7).

Para ello desde menú "**Proyecto > Tabla de variables**" seleccionamos botón "**Definir**" y creamos las siguientes variables:

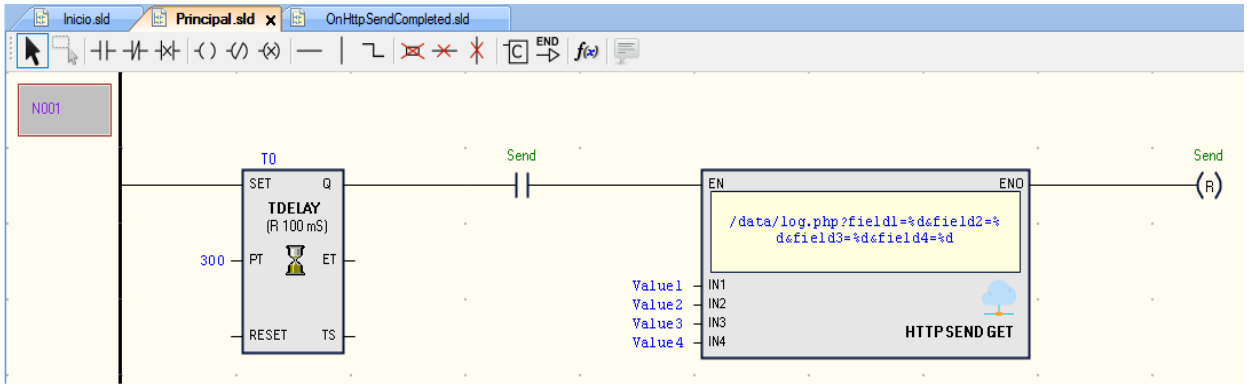
- Definir 4 variables globales tipo **Int32**, llamadas **Value1**, **Value2**, **Value3** y **Value4**.
- Definir variable global tipo **Bool** llamada **Send**, colocar **Valor Inicial** igual a 1. Se usará como indicador para enviar petición.

La siguiente ventana muestra la tabla de variables del proyecto:



5.6 Enviar Petición GET

Desde el diagrama **Principal** enviaremos el valor de las 4 variables enteras tipo **Int32** definidas previamente mediante el componente **HttpSendGet**, como se muestra a continuación:



Como podemos observar en la figura superior, utilizamos un timer tipo **TP** con la entrada configurada para que cada 300 * 0.1 = 30 segundos habilite su salida **Q** con el valor "1" y se ejecute el resto de los componentes.

Si hacemos zoom al resto de los componentes luego del timer, podremos observar mejor el componente **HttpSendGet**:

El componente **HttpSendGet** se ejecuta si la variable "**Send**" es "1" (recordar que su valor inicial es 1, ver pág. 11), si la transacción HTTP fue aceptada para ser enviada por el PLC (lo cual no quiere decir que se haya enviado con éxito), la salida **ENO** es "1" y la variable "**Send**" se pone a "0".

Al finalizar la transacción HTTP (con o sin errores), se genera el evento **OnHttpSendCompleted**, el cual ejecuta el diagrama del mismo nombre, incrementando las variables **Value** y estableciendo **Send** a 1 para el próximo envío. Pero esto lo veremos en la próxima sección.

El componente **HttpSendGet** envía el valor de las 4 variables conectadas a sus entradas (**IN1**, **IN2**, **IN3** e **IN4**), es decir **Value1**, **Value2**, **Value3** y **Value4**.

Para enviar los valores de las variables a la dirección: **ejemplo.com/data/log.php**

Debemos formar el siguiente **Query String** (ver sección 4.2 pág. 5 para información sobre el Query String):

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Donde los símbolos %d son códigos de formato que serán reemplazados por el valor de las variables apuntadas por las entradas **IN1**, **IN2**, **IN3** e **IN4** del componente (ver documentación del componente **HttpSendGet**).

Finalmente, como es una petición GET (ver sección 4.1, pág. 4), debemos colocar el Query String al PATH (ruta del recurso) separado por el símbolo "?", resultando:

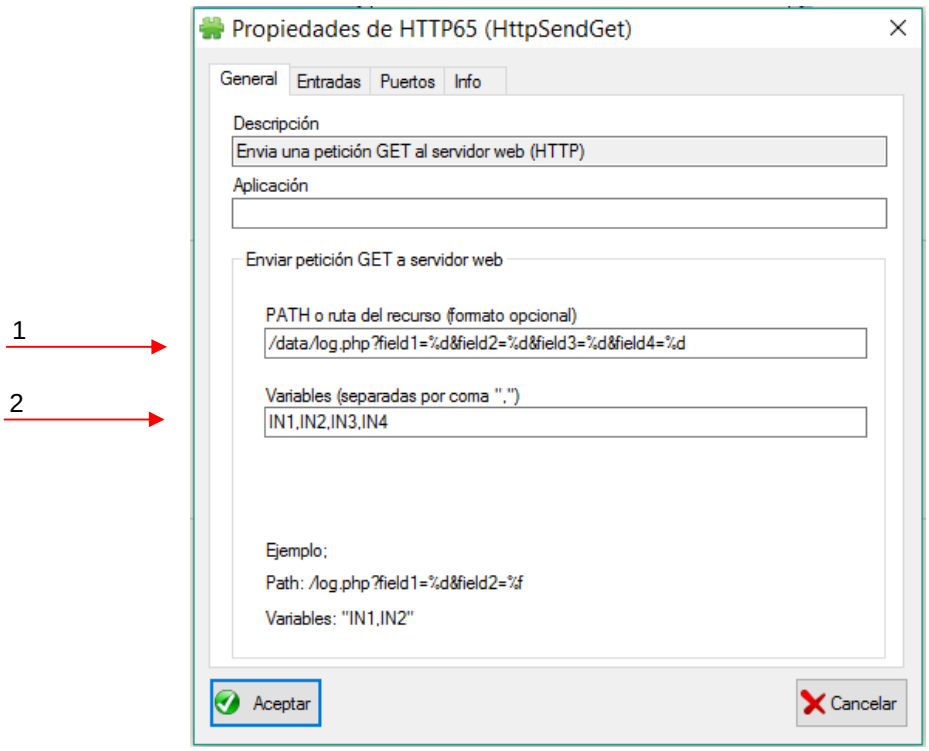
```
/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d
```

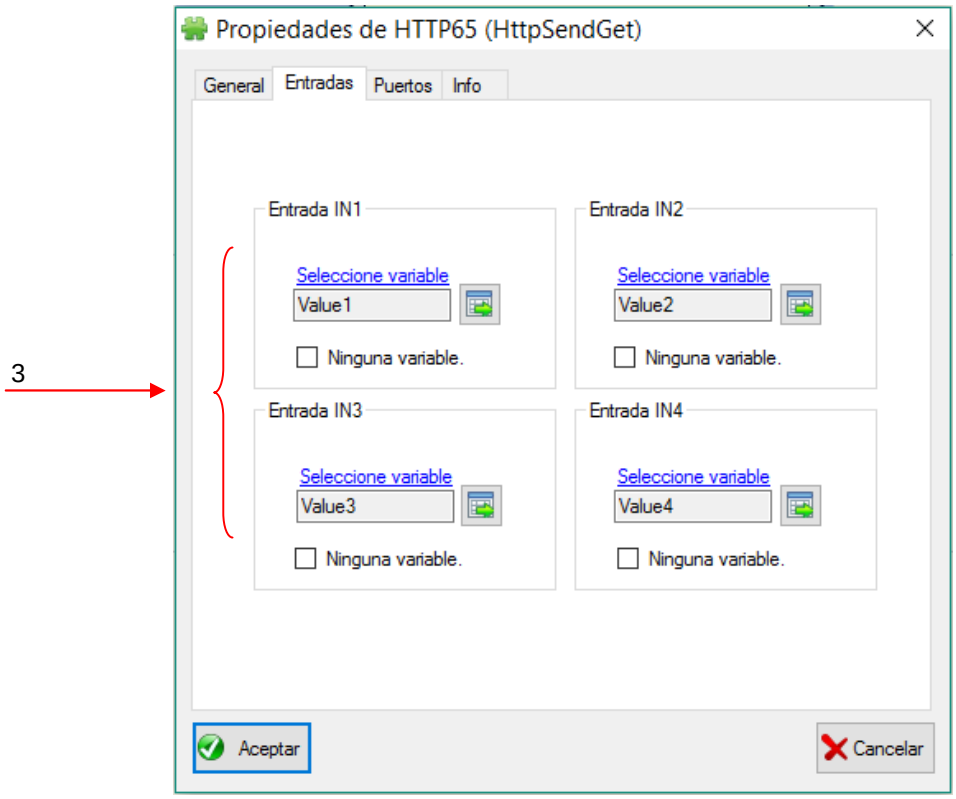
Entonces, supongamos que Value1=1, Value2=2, Value3=3 y Value4=4, el componente **HttpSendGet** enviara la siguiente cadena en la petición GET al servidor web:

```
/data/log.php?field1=1&field2=2&field3=3&field4=4
```

Luego, el servidor desde el archivo **log.php** obtendrá los campos **field1**, **field2**, **field3** y **field4**, y sus respectivos valores mediante lenguaje PHP u otro. Esta es la forma que se envían datos a un servidor web mediante GET.

El componente tiene las siguientes propiedades:





Donde de acuerdo a los números señalados en las dos últimas figuras:

1. **PATH**: Ruta del recurso a solicitar y el **Query String** (opcional, pero usado en este caso). Esta cadena puede tener formato, por lo tanto utilizar códigos de formato (%d, %f, %s, etc) que serán reemplazados por el valor de las variables listadas en el campo “Variables”.

2. **Variables**: Variables cuyos valores se utilizaran para reemplazar en los códigos de formato de la cadena PATH. Dichas variables corresponden a los puertos de entrada del componente.

3. **Entradas**: Nombre de variables apuntadas y usadas por los puertos IN1, IN2, IN3 e IN4.

5.7 Manejo del Evento OnHttpSendCompleted

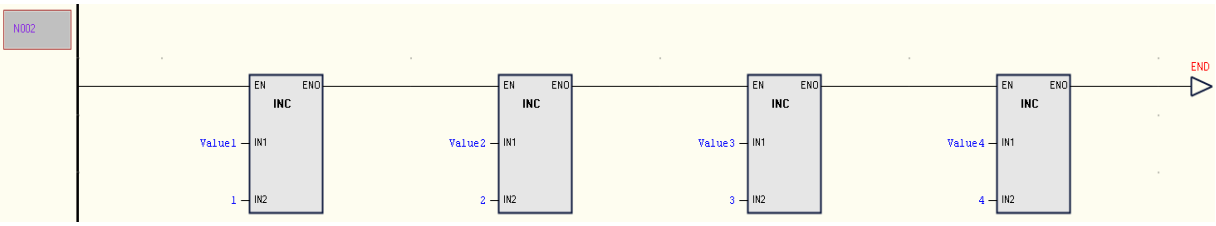
El evento **OnHttpSendCompleted** se genera cuando la transacción HTTP finaliza, ya sea con error o sin errores. Al finalizar, el diagrama **OnHttpSendCompleted.sld** es ejecutado y por lo tanto podemos colocar la lógica para procesar el evento.

En este ejemplo no comprobaremos errores (por ejemplo si no se pudo conectar al servidor web, existe algún error en la librería o el servidor web respondió con algún código de respuesta con error).

Simplemente pondremos la variable **Send** a "1" para que desde el diagrama **Principal** se llame nuevamente al componente **HttpSendGet** para otro envío y además incrementaremos los valores de las variables **Value1**, **Value2**, **Value3** y **Value4**.

Establecer **Send**=1:

Incrementar valores de variables **Value1**, **Value2**, **Value3** y **Value4**.



Al terminar de ejecutar los componentes colocados, el diagrama **OnHttpSendCompleted.sld** finaliza y el PLC retorna al control al diagrama **Principal.sld** en el punto donde fue interrumpido.

5.8 Prueba del Ejemplo GET con Servidor Web y PHP

Para probar el ejemplo necesita un servidor web funcionando y un script PHP que acepte los valores enviados por el PLC y mostrarlos de alguna forma (ya sea en una página web o almacenarlos en un archivo de texto).

Como la configuración de un servidor web y PHP excede esta nota de aplicación, solo expondremos un pequeño script PHP llamado **log.php** que podrá subir a su página web para hacer la prueba. Dicho script almacena los valores recibidos del PLC en un archivo de texto **log.txt**, de tal forma que allí podrá ver los datos recibidos cada vez que el PLC se conecte al servidor.

Recuerde en el proyecto Ladder especificar la dirección y puerto de su servidor web en el componente **HttpSendInit** y el PATH correcto al script en su servidor en el componente **HttpSendGet**, ya que el ejemplo basa la conexión en la dirección **ejemplo.com/data/log.php**, por ende la misma debe cambiarse por otra válida.

Script PHP **log.php**:

```
<?php

      // Obtener valores recibidos via HTTP GET.
      $field1 = $_GET["field1"];
      $field2 = $_GET["field2"];
      $field3 = $_GET["field3"];
      $field4 = $_GET["field4"];

      // Obtener cadena de fecha/hora actual.
      $datetime = date("Y/m/d H:i:s");

      // Armar cadena con datos recibidos.
      $logString = sprintf("Time: %s Field1: %s Field2: %s Field3: %s
                           Field4: %s\r\n", $datetime, $field1,
                           $field2, $field3, $field4);

      // Agregar cadena con datos recibidos al archivo log.txt.
      file_put_contents("log.txt", $logString, FILE_APPEND);

      // Responder con mensaje.
      echo "Log OK! Hello from PHP!\r\n"
?>
```

El script PHP anterior como podemos ver toma los valores de los campos **field** enviados por el PLC usando la instrucción **$_GET[“nombre_de_campo”]**.

Luego armamos una cadena en la variable **$logString** con la hora/fecha actual y los campos recibidos, ver función **sprintf()**.

Posteriormente con la función **file_put_contents()** guardamos el string **$logString** en el archivo “**log.txt**”, el cual cada vez que lo escribimos, agrega una línea de registro.

Finalmente le decimos al servidor que responda “**Log OK! Hello from PHP!**”. Si bien no leemos la respuesta en el PLC, podemos usarlo en el futuro para transmitir un código o mensaje.

Cuando el PLC haya enviado algunas conexiones al servidor, podremos leer el archivo **log.txt** y su contenido será similar al siguiente:

```
Time: 2016/10/11 13:39:44 Field1: 0 Field2: 0 Field3: 0 Field4: 0
Time: 2016/10/11 13:40:14 Field1: 1 Field2: 2 Field3: 3 Field4: 4
Time: 2016/10/11 13:40:44 Field1: 2 Field2: 4 Field3: 6 Field4: 8
Time: 2016/10/11 13:41:14 Field1: 3 Field2: 6 Field3: 9 Field4: 12
Time: 2016/10/11 13:41:44 Field1: 4 Field2: 8 Field3: 12 Field4: 16
Time: 2016/10/11 13:42:14 Field1: 5 Field2: 10 Field3: 15 Field4: 20
Time: 2016/10/11 13:42:45 Field1: 6 Field2: 12 Field3: 18 Field4: 24
Time: 2016/10/11 13:43:15 Field1: 7 Field2: 14 Field3: 21 Field4: 28
Time: 2016/10/11 13:43:45 Field1: 8 Field2: 16 Field3: 24 Field4: 32
Time: 2016/10/11 13:44:15 Field1: 9 Field2: 18 Field3: 27 Field4: 36
```

Notar que cada línea de **log.txt** es una conexión del PLC, a la cual se le agregó la fecha/hora de la misma y el valor de los campos Field transmitidos, que representan las variables **Value1**, **Value2**, **Value3** y **Value4**.

Observar como cada línea está separada por 30 segundos en el tiempo.

Si todo funciona bien, el archivo **log.txt** puede ser accedido con el navegador en la dirección:

http://ejemplo.com/data/log.txt

El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendGet1.zip**, dentro de la carpeta "**Documentos**" del proyecto puede encontrar el archivo **log.php** y una copia de **log.txt** como referencia.

5.9 Enviar Petición POST

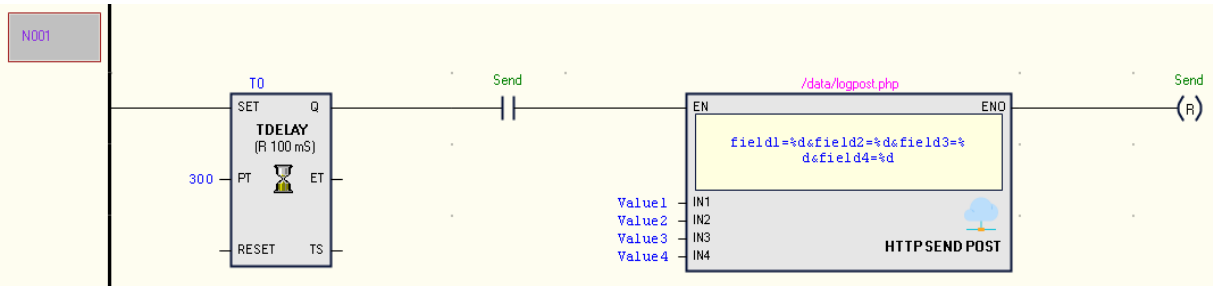
Enviar una petición POST no difiere demasiado de enviar una petición GET desde el punto de vista de programación.

Para ello nos basaremos en el ejemplo anterior **HttpSendGet1.zip**, el cual puede descargar de la página web con el nombre: **HttpSendPost1.zip**

El objetivo del ejemplo POST será el mismo que para el ejemplo GET (ver pág. 7) y será transmitir cada 30 segundos 4 números enteros mediante una petición POST a un servidor web de ejemplo, llamado **ejemplo.com**. Luego de una transmisión, se incrementan los valores de los 4 números enteros.

La dirección web completa del servidor de ejemplo será: **ejemplo.com/data/logpost.php**

Entonces para lograr dicho objetivo, en el ejemplo anterior, cambiamos el componente **HttpSendGet** por **HttpSendPost** en el diagrama principal:



Haciendo zoom al componente **HttpSendPost**:

El componente **HttpSendPost** se ejecuta si la variable "**Send**" es "1" (recordar que su valor inicial es 1, ver pág. 11), si la transacción HTTP fue aceptada para ser enviada por el PLC (lo cual no quiere decir que se haya enviado con éxito), la salida **ENO** es "1" y la variable "**Send**" se pone a "0".

Al finalizar la transacción HTTP (con o sin errores), se genera el evento **OnHttpSendCompleted**, el cual ejecuta el diagrama del mismo nombre, incrementando las variables **Value** y estableciendo **Send** a 1 para el próximo envío. Pero esto lo veremos en la próxima sección.

El componente **HttpSendPost** envía el valor de las 4 variables conectadas a sus entradas (**IN1**, **IN2**, **IN3** e **IN4**), es decir **Value1**, **Value2**, **Value3** y **Value4**.

Para enviar los valores de las variables a la dirección: **ejemplo.com/data/logpost.php**

Debemos formar el siguiente **Query String** (ver sección 4.2 pág. 5 para información sobre el Query):

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Donde los símbolos %d son códigos de formato que serán reemplazados por el valor de las variables apuntadas por las entradas **IN1**, **IN2**, **IN3** e **IN4** del componente (ver documentación del componente **HttpSendPost**).

Finalmente, como es una petición POST (ver sección 4.1, pág. 4), debemos colocar el Query String por separado del PATH (ruta del recurso), resultando:

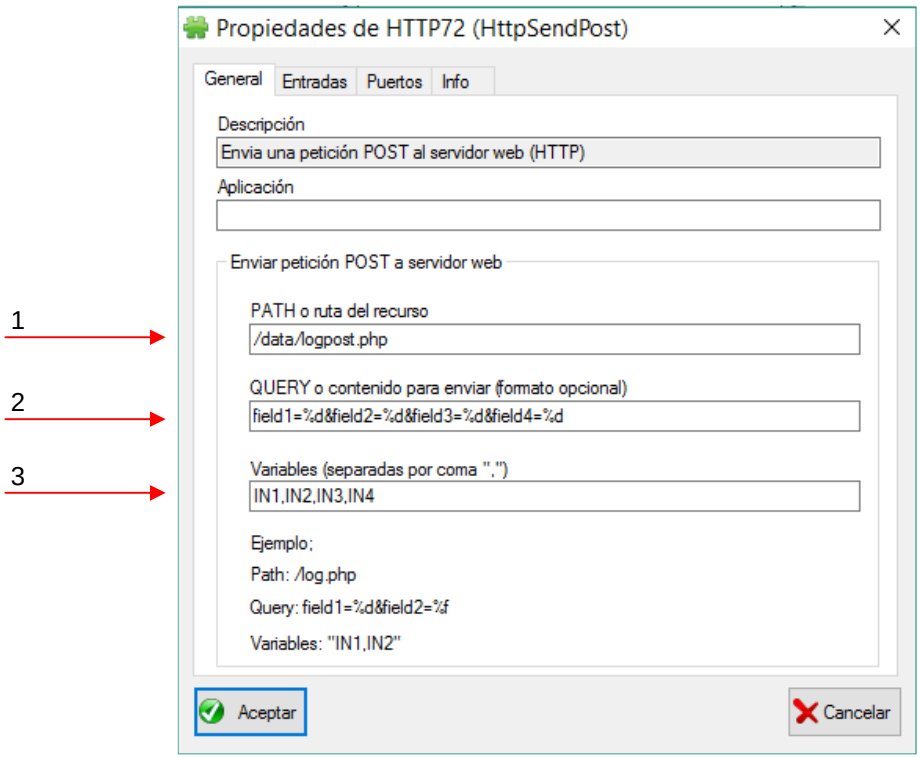
```
PATH  = /data/logpost.php
QUERY = field1=%d&field2=%d&field3=%d&field4=%d
```

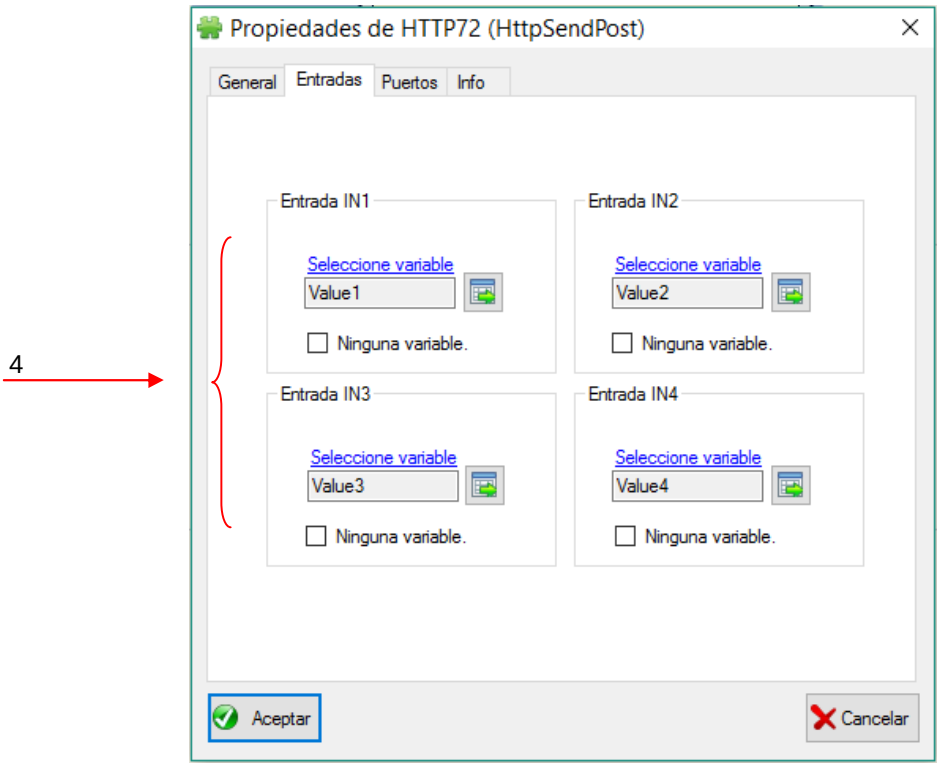
Entonces, supongamos que Value1=1, Value2=2, Value3=3 y Value4=4, el componente **HttpSendPost** enviará la siguiente cadena QUERY en la petición POST al servidor web:

```
field1=1&field2=2&field3=3&field4=4
```

Luego, el servidor desde el archivo **logpost.php** obtendrá los campos **field1**, **field2**, **field3** y **field4**, y sus respectivos valores mediante lenguaje PHP u otro. Esta es la forma que se envían datos a un servidor web mediante POST.

El componente tiene las siguientes propiedades:





Donde de acuerdo a los números señalados en las dos últimas figuras:

1. **PATH**: Ruta del recurso a solicitar

2. **QUERY**: Cadena con datos a enviar. Esta cadena puede tener formato y por lo tanto utilizar códigos de formato (%d, %f, %s, etc) que serán reemplazados por el valor de las variables listadas en el campo “Variables”.

3. **Variables**: Variables cuyos valores se utilizaran para reemplazar en los códigos de formato de la cadena PATH. Dichas variables corresponden a los puertos de entrada del componente.

4. **Entradas**: Nombre de variables apuntadas y usadas por los puertos IN1, IN2, IN3 e IN4.

El resto del código del proyecto fue explicado en el ejemplo GET, por lo tanto el comportamiento es el mismo.

5.9.1 Prueba del Ejemplo POST con Servidor Web y PHP

Para probar el ejemplo necesita un servidor web funcionando y un script PHP que acepte los valores enviados por el PLC y mostrarlos de alguna forma (ya sea en una página web o almacenarlos en un archivo de texto).

Como la configuración de un servidor web y PHP excede esta nota de aplicación, solo expondremos un pequeño script PHP llamado **logpost.php** que podrá subir a su página web para hacer la prueba. Dicho script almacena los valores recibidos del PLC en un archivo de texto **log.txt**, de tal forma que allí podrá ver los datos recibidos cada vez que el PLC se conecte al servidor.

Recuerde en el proyecto Ladder especificar la dirección y puerto de su servidor web en el componente **HttpSendInit** y el **PATH / QUERY** correcto al script en su servidor en el componente **HttpSendPost**, ya que el ejemplo basa la conexión en la dirección **ejemplo.com/data/logpost.php**, por ende la misma debe cambiarse por otra válida.

Script PHP **logpost.php**:

```
<?php

      // Obtener valores recibidos via HTTP POST.
      $field1 = $_POST["field1"];
      $field2 = $_POST["field2"];
      $field3 = $_POST["field3"];
      $field4 = $_POST["field4"];

      // Obtener cadena de fecha/hora actual.
      $datetime = date("Y/m/d H:i:s");

      // Armar cadena con datos recibidos.
      $logString = sprintf("Time: %s Field1: %s Field2: %s Field3: %s
                           Field4: %s\r\n", $datetime, $field1,
                           $field2, $field3, $field4);

      // Agregar cadena con datos recibidos al archivo log.txt.
      file_put_contents("log.txt", $logString, FILE_APPEND);

      // Responder con mensaje.
      echo "Log OK! Hello from PHP!\r\n"
?>
```

El script PHP anterior como podemos ver toma los valores de los campos **field** enviados por el PLC usando la instrucción **$_POST[“nombre_de_campo”]**.

Luego armamos una cadena en la variable **$logString** con la hora/fecha actual y los campos recibidos, ver función **sprintf()**.

Posteriormente con la función **file_put_contents()** guardamos el string **$logString** en el archivo “**log.txt**”, el cual cada vez que lo escribimos, agrega una línea de registro.

Finalmente le decimos al servidor que responda “**Log OK! Hello from PHP!**”. Si bien no leemos la respuesta en el PLC, podemos usarlo en el futuro para transmitir un código o mensaje.

Cuando el PLC haya enviado algunas conexiones al servidor, podremos leer el archivo **log.txt** y su contenido será similar al siguiente:

```
Time: 2016/10/11 20:28:25 Field1: 0 Field2: 0 Field3: 0 Field4: 0
Time: 2016/10/11 20:28:53 Field1: 1 Field2: 2 Field3: 3 Field4: 4
Time: 2016/10/11 20:29:22 Field1: 2 Field2: 4 Field3: 6 Field4: 8
Time: 2016/10/11 20:29:53 Field1: 3 Field2: 6 Field3: 9 Field4: 12
Time: 2016/10/11 20:30:23 Field1: 4 Field2: 8 Field3: 12 Field4: 16
Time: 2016/10/11 20:30:53 Field1: 5 Field2: 10 Field3: 15 Field4: 20
Time: 2016/10/11 20:31:23 Field1: 6 Field2: 12 Field3: 18 Field4: 24
Time: 2016/10/11 20:31:53 Field1: 7 Field2: 14 Field3: 21 Field4: 28
Time: 2016/10/11 20:32:23 Field1: 8 Field2: 16 Field3: 24 Field4: 32
Time: 2016/10/11 20:32:53 Field1: 9 Field2: 18 Field3: 27 Field4: 36
```

Notar que cada línea de **log.txt** es una conexión del PLC, a la cual se le agregó la fecha/hora de la misma y el valor de los campos Field transmitidos, que representan las variables **Value1**, **Value2**, **Value3** y **Value4**.

Observar como cada línea está separada por 30 segundos (aproximadamente por retardos de red) en el tiempo.

Si todo funciona bien, el archivo **log.txt** puede ser accedido con el navegador en la dirección:

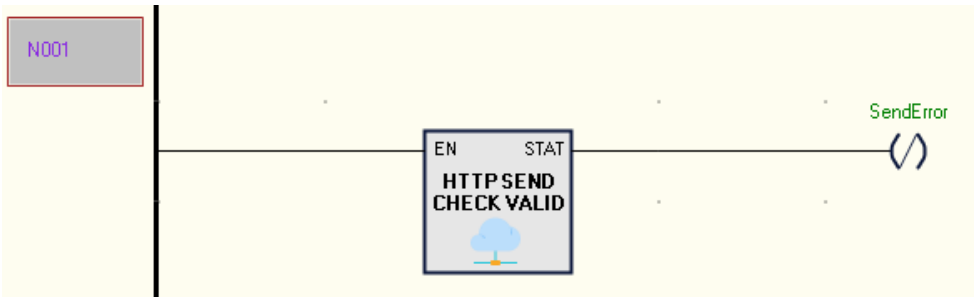
http://ejemplo.com/data/log.txt

El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendPost1.zip**, dentro de la carpeta "**Documentos**" del proyecto puede encontrar el archivo **logpost.php** y una copia de **log.txt** como referencia.

5.10 Comprobar Errores

Para comprobar errores podemos utilizar el componente **HttpSendCheckValidTransaction** el cual retorna en su salida "**STAT**" el valor "1" si la transacción fue exitosa y "0" si contiene algún error.

Partiendo de los ejemplos anteriores (**HttpSendGet1.zip** o **HttpSendPost1.zip**) agregamos una nueva variable global tipo bool llamada **SendError** la cual será "1" si la transacción contiene algún error y "0" si fue exitosa. Luego insertamos una nueva network al comienzo del diagrama **OnHttpSendCompleted** y agregamos el componente **HttpSendCheckValidTransaction** en la network N001 como se muestra a continuación:



Como se puede ver, si la salida STAT=1, la variable **SendError** se pone en "0" porque tenemos una bobina inversora. Si la salida STAT=0, la variable **SendError** se hace "1" indicando error en la transmisión.

El componente **HttpSendCheckValidTransaction** comprueba dos condiciones que deben cumplirse al mismo tiempo en una transacción HTTP normal para que sea totalmente exitosa:

1. El código de estado de la librería HTTP SEND es "0"
2. El código de respuesta del servidor HTTP es "200".

En caso de error, si desea analizar en profundidad, puede obtener los códigos completos retornados por los componentes **HttpSendGetLibStatus** y **HttpSendGetResponseCode**, para así determinar la causa del error.

La variable **SendError** también puede utilizarse en otras partes del proyecto ya que su alcance es global.

Los ejemplos **HttpSendGet2.zip** y **HttpSendPost2.zip** disponibles en nuestro sitio web agregan comprobación de errores.

5.11 Obtener Mensaje Respuesta del Servidor

En algunas aplicaciones puede ser útil obtener el cuerpo del mensaje de respuesta del servidor web.

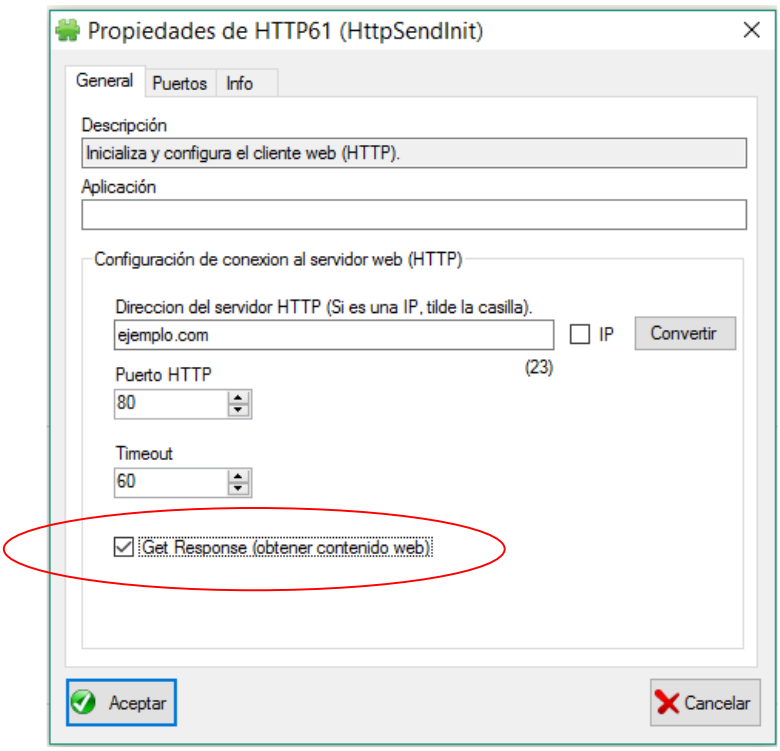
El cuerpo del mensaje de respuesta es el contenido del recurso solicitado, que puede ser una página web, una cadena de texto de un archivo TXT o una imagen.

Puede utilizar esta respuesta para procesar un mensaje o código del servidor web (por ejemplo si consultó una base de datos, la respuesta puede contener un valor que le sirva para validar acceso a la misma).

Note que una página web dinámica (por ejemplo alguna con contenido PHP) puede retornar diferentes valores en cada acceso, que dependen del servidor web y esto es muy útil para comunicarse desde la web con el PLC.

El PLC puede almacenar en un buffer interno cada respuesta del servidor web. Este buffer es limitado y solo almacena los primeros 308 bytes de contenido (valor que puede variar en algunos modelos).

Para que el PLC almacene el contenido en la página web, el componente **HttpSendInit** debe configurarse con la casilla "**Get Response**" seleccionada como muestra la siguiente figura:



Hay dos componentes que están asociados al buffer que contiene los datos recibidos, el componente **HttpSendGetBodyData** (que permite obtener en un array los datos almacenados en el buffer) y **HttpSendGetBodyLength** (que permite obtener la cantidad de bytes almacenados en el buffer correspondiente a la ultima respuesta).

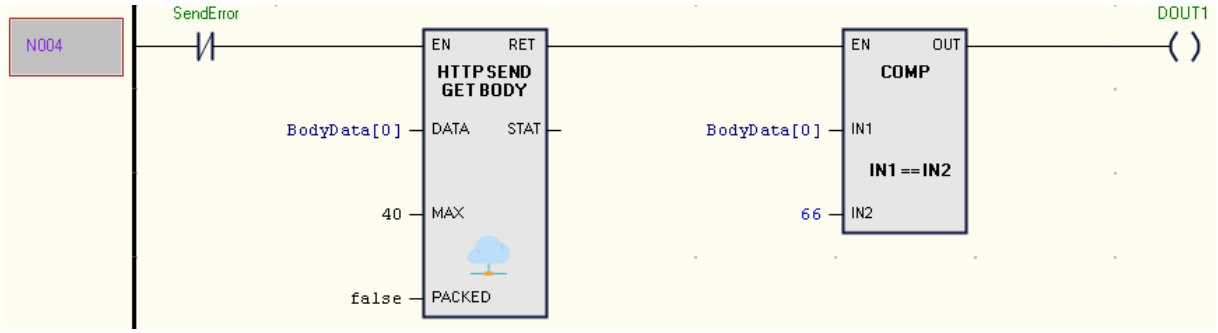
Partiendo de los ejemplos anteriores (**HttpSendGet2.zip** o **HttpSendPost2.zip**) agregamos un array global del tipo **Int32_Array** con 40 elementos llamado **BodyData**. Esto nos va a permitir recuperar 40 bytes del buffer interno del PLC cuando reciba respuesta del servidor web.

Supongamos que el servidor responda ante cada petición GET o POST el siguiente cuerpo de mensaje:

```
Bienvenidos al servidor web!
```

Nuestra misión va ser leer el primer carácter de la respuesta del cuerpo del mensaje y si corresponde a la letra “B” activaremos la salida **DOUT1** del PLC.

Entonces, en el diagrama **OnHttpSendCompleted** agregamos los componentes como muestra la próxima figura:



El componente **HttpSendGetBodyData** (ver **HTTP SEND GET BODY** en figura de arriba) copiará como máximo 40 bytes del buffer interno luego de una transacción y lo guardará en el array **BodyData[]**.
Esta copia se realizará si la transacción fue exitosa, porque comprobamos la variable **SendError** que tiene el valor “1” si hay error (ver sección 5.10), pero al usar un contacto inversor, cuando sea “0” (no error) se ejecutarán el resto de los componentes.

Luego si el primer elemento de **BodyData**, es decir **BodyData[0]**, contiene el valor “66” activará la salida **DOUT1**. Note que el valor “66” representa el carácter “B” en código ASCII, por lo tanto si el servidor retorna un cuerpo de mensaje con la primera letra igual a “B”, la salida **DOUT1** será activada.

Este es un ejemplo trivial, pero puede realizar mayores comprobaciones para operaciones complejas.

Puede utilizar componentes con lenguaje Pawn también, ya que suelen manejar más fácilmente cadenas de texto con información.

Los ejemplos **HttpSendGet3.zip** y **HttpSendPost3.zip** disponibles en nuestro sitio web procesamiento de cuerpo de mensaje de respuesta visto en esta sección.

Tip: En **StxLadder**, en el menú “**Herramientas > Conversión > ASCII**” tiene un conversor ASCII que puede serle de utilidad en sus proyectos.

5.12 Fin de Ejemplos

Estas son las nociones básicas para usar el Cliente Web. Puede usar estos ejemplos y otros disponibles en nuestro sitio de Internet como base para sus proyectos.

Recomendamos altamente:

- Leer las descripciones de cada componente desde el entorno **StxLadder** antes de usarlos para mejorar la comprensión de los mismos.
- Dar un vistazo a Referencia de Componentes en Lenguaje Ladder, página 46.
- Ir a Slicetex.com y descargar programas de ejemplos desde página web.

6 Ejemplo de Uso Inicial en Lenguaje Pawn

Antes de comenzar con las definiciones las funciones Pawn disponibles, empezaremos con un ejemplo básico para utilizar el Cliente Web rápidamente.

Nos detendremos en los puntos importantes para entender los conceptos claves a medida que avance el ejemplo.

En esta sección expondremos un ejemplo básico en lenguaje Pawn, si está usando lenguaje Ladder, puede ir directamente al ejemplo Ladder en la página 7.

Es altamente recomendado que practique este ejemplo, así podrá comprender las nociones fundamentales de uso del Cliente Web del PLC.

6.1 Crear el Proyecto

Comience un nuevo proyecto en **StxLadder** y seleccione el **modelo de PLC** que tenga adquirido.

Nota: El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendGetPawn1.zip**

6.2 Objetivo del Proyecto

Este ejemplo tendrá como objetivo transmitir cada 30 segundos 4 números enteros mediante una petición GET a un servidor web de ejemplo, llamado **ejemplo.com**. Luego de una transmisión, se incrementan los valores de los 4 números enteros.

La dirección web completa del servidor de ejemplo será: **ejemplo.com/data/log.php**

6.3 Configurar Cliente Web

En la función **PlcMain()** del archivo **PlcMain.p** del proyecto vamos a utilizar la función **HttpSendInit()** para inicializar el Cliente Web. La misma debe configurarse una vez antes de enviar una petición al servidor web o utilizar la librería HTTP SEND.

El siguiente código Pawn muestra como inicializar el Cliente Web:

```
HttpSendInit("ejemplo.com", 80, HTTP_SEND_OPT_DEFAULT, 60)
```

Los parámetros utilizados para conectarse al servidor web son:

- **Dirección**: ejemplo.com
- **Puerto HTTP**: 80
- **Opciones**: Utilizar valores por defecto de libraría (**HTTP_SEND_OPT_DEFAULT**)
- **Timeout**: 60

Si deseamos conectarnos a un servidor web con la dirección IP 192.168.1.15 podemos hacer:

```
HttpSendInit("192.168.1.15", 80, HTTP_SEND_OPT_USE_IP, 60)
```

Notar en la línea anterior, que se utiliza la opción de configuración **HTTP_SEND_OPT_USE_IP**, la cual indica que la dirección del servidor esta en formato de números de dirección IP.

En la sección 8.1 de la página 67 puede encontrar una referencia completa del uso de la función **HttpSendInit()**.

6.4 Código de Función PlcMain() Completo

Se utiliza el siguiente código para inicializar el PLC con el Cliente Web:

```
PlcMain()
{
   //
   // Inicializar cliente HTTP.
   //

   HttpSendInit("ejemplo.com", 80, HTTP_SEND_OPT_DEFAULT, 60)

   // Activar evento OnHttpSendCompleted()
   HttpSendSetCompletedEvent()

   // Inicializar eventos Timeout, ver OnTimeout().
   TimeoutInitEvent()

   // Configurar temporizador Timeout1 para generarse en próximos 30 segundos.
   Timeout1SetEvent(30)

   //
   // LOOP PRINCIPAL DE PROGRAMA
   //

   for(;;)
   {
          // Delay 1 segundo.
          DelayS(1)

          // Conmutar Led Debug.
          LedToggle()
   }

   // Retorno.
   return 0
}
```

Observar como en el código anterior se inicializa el Cliente Web con la función **HttpSendInit()** como ya hemos explicado en la sección anterior.

Se habilitarán dos eventos:

- **OnHttpSendCompleted():** Con la función HttpSendSetCompletedEvent().
- **OnTimeout():** Con las funciones TimeoutInitEvent() y Timeout1SetEvent(30).

Luego en el **LOOP** principal del programa simplemente conmuta el led Debug cada 1 segundo, y no hace nada más. Es decir esta libre para cualquier otra operación.

Esto significa que la transmisión HTTP con el servidor la realizaremos desde los eventos habilitados, que se explican en las próximas secciones.

Tip: Recordar que un "evento" interrumpe la ejecución del programa en el PLC de forma asíncrona (en cualquier momento) y se pasa el control de ejecución al código del evento en sí. Luego cuando se termina de ejecutar el evento, el PLC continúa ejecutando el programa desde el punto en que fue interrumpido. Es análogo a las interrupciones en computación.

6.5 Crear Variables

En el siguiente paso vamos a crear variables para que el programa funcione de acuerdo a nuestro objetivo inicial (ver pág. 27).

Para ello definiremos las siguientes variables globales en Pawn al comienzo del archivo **PlcMain.p**:

```
//
// Variable de estado:
// Si su valor es "1" se inicia una transmisión HTTP al servidor web.
//

new Send = 1

//
// Valores a transmitir por HTTP al servidor web.
//

new Value1, Value2, Value3, Value4
```

Donde **Send** es utilizada como flag, si su valor es "1" se inicia la transmisión y si es "0" es porque hay una transacción en curso o no se puede iniciar transmisión. Notar que el valor inicial es "1".

Las variables **Value1**, **Value2**, **Value3** y **Value4** almacenan valores para transmitir y se incrementan luego de cada transmisión.

6.6 Crear Evento Timeout – OnTimeout() - Petición GET

Desde la función **PlcMain()** hemos configurado para que se produzca un evento del tipo **OnTimeout()** cada 30 segundos con las instrucciones:

```
   // Inicializar eventos Timeout, ver OnTimeout().
   TimeoutInitEvent()

   // Configurar temporizador Timeout1 para generarse en próximos 30 segundos.
   Timeout1SetEvent(30)
```

Por lo tanto ahora nos queda crear el código para manejar el evento. La tarea que le asignaremos será la de iniciar la transmisión HTTP con el servidor y enviar mediante una petición GET el valor de las variables **Value1**, **Value2**, **Value3** y **Value4.**

El código del evento se crea a continuación:

```
@OnTimeout()
{
   //
   // Comprobar si Timeout1 ha expirado.
   //

   if(Timeout1Check() == 1)
   {
      //
      // Comprobar si debemos iniciar una transmisión HTTP.
      //

      if(Send == 1)
      {
         // Enviar 4 campos con valores al servidor web.

         if(HttpSendGet("/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d",
            Value1, Value2, Value3, Value4) == 0)
         {
            // Transmisión en curso, borrar flag.
            Send = 0
         }

      // Recargar Timeout1 para generarse en próximos 30 segundos.
      Timeout1Reload(30)
   }
}
```

Cuando el evento **OnTimeout** se produzca, se comprueba si se produjo por expiración de **Timeout1**:

```
if(Timeout1Check() == 1)
```

Si fue así, se comprueba si podemos iniciar una transmisión leyendo el valor de la variable **Send**, que debe ser “1” (recordar que su valor inicial es 1, ver pág. 29):

```
if(Send == 1)
```

Acto seguido enviamos la petición HTTP GET al servidor web con la función **HttpSendGet():**

```
HttpSendGet("/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d", Value1,
            Value2, Value3, Value4)
```

Si la transacción HTTP fue aceptada para ser enviada por el PLC (lo cual no quiere decir que se haya enviado con éxito), el retorno de la función es “0” y la variable “**Send**” se pone a “0” dentro del “**if**”.

Al finalizar la transacción HTTP (con o sin errores), se genera el evento **OnHttpSendCompleted**, el cual ejecuta la función del mismo nombre incrementando las variables **Value** y estableciendo **Send** a 1 para el próximo envío. Pero esto lo veremos en la próxima sección 6.7.

La función **HttpSendGet()** acepta como parámetros la cadena “**PATH**”, que contiene el nombre del recurso a acceder y un **QUERY STRING** (opcional). Además envía el valor de las 4 variables **Value1**, **Value2**, **Value3** y **Value4**, las cuales se especifican al final de la lista de argumentos de la función.

Para enviar los valores de las variables a la dirección: **ejemplo.com/data/log.php**

Debemos formar el siguiente **Query String** (ver sección 4.2 pág. 5 para información sobre el Query String) dentro del PATH:

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Donde los símbolos %d son códigos de formato que serán reemplazados por el valor de las variables **Value1**, **Value2**, **Value3** y **Value4**. Ver documentación de función **HttpSendGet()** en sección 8.

Finalmente, como es una petición GET (ver sección 4.1, pág. 4), debemos colocar el Query String al PATH (ruta del recurso) separado por el símbolo "?", resultando:

```
/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d
```

Entonces, supongamos que Value1=1, Value2=2, Value3=3 y Value4=4, la función **HttpSendGet** enviará la siguiente cadena en la petición GET al servidor web:

```
/data/log.php?field1=1&field2=2&field3=3&field4=4
```

Luego, el servidor desde el archivo **log.php** obtendrá los campos **field1**, **field2**, **field3** y **field4**, y sus respectivos valores mediante lenguaje PHP u otro. Esta es la forma que se envían datos a un servidor web mediante GET.

6.7 Crear Evento de Fin de Transacción HTTP – OnHttpSendCompleted()

En el siguiente paso vamos a definir el código para el evento "**OnHttpSendCompleted**", el cual es un evento que se llama cuando la transacción HTTP fue completada (con o sin errores).

En este ejemplo no comprobaremos errores (por ejemplo si no se pudo conectar al servidor web, existe algún error en la librería o el servidor web respondió con algún código de respuesta con error).

Simplemente pondremos la variable **Send** a "1" para que desde el evento **OnTimeout** se llame nuevamente a la función **HttpSendGet** para otro envío y además incrementaremos los valores de las variables **Value1**, **Value2**, **Value3** y **Value4**.

```
@OnHttpSendCompleted()
{
   // Habilitar flag para inciar próxima transmisión.
   Send = 1

   // Incrementar variables.
   Value1 += 1
   Value2 += 2
   Value3 += 3
   Value4 += 4
}
```

Recordar que el evento **OnHttpSendCompleted** fue habilitado desde **PlcMain()** con la instrucción:

```
   HttpSendSetCompletedEvent()
```

6.8 Prueba del Ejemplo GET con Servidor Web y PHP

Para probar el ejemplo necesita un servidor web funcionando y un script PHP que acepte los valores enviados por el PLC y mostrarlos de alguna forma (ya sea en una página web o almacenarlos en un archivo de texto).

Como la configuración de un servidor web y PHP excede esta nota de aplicación, solo expondremos un pequeño script PHP llamado **log.php** que podrá subir a su página web para hacer la prueba. Dicho script almacena los valores recibidos del PLC en un archivo de texto **log.txt**, de tal forma que allí podrá ver los datos recibidos cada vez que el PLC se conecte al servidor.

Recuerde en el proyecto Pawn especificar la dirección y puerto de su servidor web en la función **HttpSendInit()** y el PATH correcto al script en su servidor en la función **HttpSendGet()**, ya que el ejemplo basa la conexión en la dirección **ejemplo.com/data/log.php**, por ende la misma debe cambiarse por otra válida.

Script PHP **log.php**:

```
<?php

      // Obtener valores recibidos via HTTP GET.
      $field1 = $_GET["field1"];
      $field2 = $_GET["field2"];
      $field3 = $_GET["field3"];
      $field4 = $_GET["field4"];

      // Obtener cadena de fecha/hora actual.
      $datetime = date("Y/m/d H:i:s");

      // Armar cadena con datos recibidos.
      $logString = sprintf("Time: %s Field1: %s Field2: %s Field3: %s
                           Field4: %s\r\n", $datetime, $field1,
                           $field2, $field3, $field4);

      // Agregar cadena con datos recibidos al archivo log.txt.
      file_put_contents("log.txt", $logString, FILE_APPEND);

      // Responder con mensaje.
      echo "Log OK! Hello from PHP!\r\n"
?>
```

El script PHP anterior como podemos ver toma los valores de los campos **field** enviados por el PLC usando la instrucción **$_GET[“nombre_de_campo”]**.

Luego armamos una cadena en la variable **$logString** con la hora/fecha actual y los campos recibidos, ver función **sprintf()**.

Posteriormente con la función **file_put_contents()** guardamos el string **$logString** en el archivo “**log.txt**”, el cual cada vez que lo escribimos, agrega una línea de registro.

Finalmente le decimos al servidor que responda “**Log OK! Hello from PHP!**”. Si bien no leemos la respuesta en el PLC, podemos usarlo en el futuro para transmitir un código o mensaje.

Cuando el PLC haya enviado algunas conexiones al servidor, podremos leer el archivo **log.txt** y su contenido será similar al siguiente:

```
Time: 2016/10/11 13:39:44 Field1: 0 Field2: 0 Field3: 0 Field4: 0
Time: 2016/10/11 13:40:14 Field1: 1 Field2: 2 Field3: 3 Field4: 4
Time: 2016/10/11 13:40:44 Field1: 2 Field2: 4 Field3: 6 Field4: 8
Time: 2016/10/11 13:41:14 Field1: 3 Field2: 6 Field3: 9 Field4: 12
Time: 2016/10/11 13:41:44 Field1: 4 Field2: 8 Field3: 12 Field4: 16
Time: 2016/10/11 13:42:14 Field1: 5 Field2: 10 Field3: 15 Field4: 20
Time: 2016/10/11 13:42:45 Field1: 6 Field2: 12 Field3: 18 Field4: 24
Time: 2016/10/11 13:43:15 Field1: 7 Field2: 14 Field3: 21 Field4: 28
Time: 2016/10/11 13:43:45 Field1: 8 Field2: 16 Field3: 24 Field4: 32
Time: 2016/10/11 13:44:15 Field1: 9 Field2: 18 Field3: 27 Field4: 36
```

Notar que cada línea de **log.txt** es una conexión del PLC, a la cual se le agregó la fecha/hora de la misma y el valor de los campos Field transmitidos, que representan las variables **Value1**, **Value2**, **Value3** y **Value4**.

Observar como cada línea está separada por 30 segundos en el tiempo.

Si todo funciona bien, el archivo **log.txt** puede ser accedido con el navegador en la dirección:

http://ejemplo.com/data/log.txt

El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendGetPawn1.zip**, dentro de la carpeta "**Documentos**" del proyecto puede encontrar el archivo **log.php** y una copia de **log.txt** como referencia.

6.9 Código Completo del Ejemplo Inicial

```
// ----------------------------------------------------------------------------
// Archivo      : PlcMain.p
// Titulo       : Script principal del PLC.
//
// Creado por   : StxLadder Version 1.8.0.
// Fecha        : 21/10/2016 11:13:02
//
// Descripción  :
//
// Ejemplo para enviar datos web mediante petición GET.
// No se comprueban errores de transmisión.
//
// ----------------------------------------------------------------------------

// ----------------------------------------------------------------------------
// VARIABLES GLOBALES
// ----------------------------------------------------------------------------

//
// Variable de estado:
// Si su valor es "1" se inicia una transmision HTTP al servidor web.
//

new Send = 1

//
// Valores a transmitir por HTTP al servidor web.
//

new Value1, Value2, Value3, Value4

// ----------------------------------------------------------------------------
// FUNCIONES
// ----------------------------------------------------------------------------

// ****************************************************************************
// Funcion     : PlcMain()
// Descripcion : Punto de entrada principal del PLC.
// ****************************************************************************

PlcMain()
{
   //
   // Inicializar cliente HTTP.
   //
   // Parametros:
   // Servidor destino: ejemplo.com
   // Puerto TCP: 80
   // Opciones: Utilizar valores por defecto de libreria.
   // Timeout: 60 segundos
   //

   HttpSendInit("ejemplo.com", 80, HTTP_SEND_OPT_DEFAULT, 60)

   // Activar evento OnHttpSendCompleted()
   HttpSendSetCompletedEvent()
```

```
   // Inicializar eventos Timeout, ver OnTimeout().
   TimeoutInitEvent()

   // Configurar temporizador Timeout1 para generarse en
   // proximos 30 segundos.
   Timeout1SetEvent(30)

   //
   // LOOP PRINCIPAL DE PROGRAMA
   //

   for(;;)
   {
          // Delay 1 segundo.
          DelayS(1)

          // Conmutar Led Debug.
          LedToggle()
   }

   // Retorno.
   return 0
}

// ------------------------------------------------------------------------------
// EVENTOS
// ------------------------------------------------------------------------------

// ****************************************************************************
// @OnTimeout(): Manejador del evento @OnTimeout.
// Descripcion : Evento generado cuando un temporizador timeoput expira.
// ****************************************************************************

@OnTimeout()
{
   //
   // Comprobar si Timeout1 ha expirado.
   //

   if(Timeout1Check() == 1)
   {
      //
      // Comprobar si debemos iniciar una transmision HTTP.
      //

      if(Send == 1)
      {
         // Enviar 4 campos al servidor web.
         if(HttpSendGet("/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d",
                        Value1, Value2, Value3, Value4) == 0)
         {
            // Tranmision en curso, borrar flag.
            Send = 0
         }
      }

      // Recargar Timeout1 para generarse en proximos 30 segundos.
      Timeout1Reload(30)
   }
```

```
}

// ********************************************************************************
// Funcion     : OnHttpSendCompleted()
// Descripcion : Evento generado al finalizar transaccion HTTP.
// ********************************************************************************

@OnHttpSendCompleted()
{
   // Habilitar flag para inciar proxima transmision.
   Send = 1

   // Incrementar variables.
   Value1 += 1
   Value2 += 2
   Value3 += 3
   Value4 += 4
}
```

6.10 Enviar Petición POST

Enviar una petición POST no difiere demasiado de enviar una petición GET desde el punto de vista de programación.

Para ello nos basaremos en el ejemplo anterior **HttpSendGetPawn1.zip**, el cual puede descargar de la página web con el nombre: **HttpSendPostPawn1.zip**

El objetivo del ejemplo POST será el mismo que para el ejemplo GET (ver pág. 27) y será transmitir cada 30 segundos 4 números enteros mediante una petición POST a un servidor web de ejemplo, llamado **ejemplo.com**. Luego de una transmisión, se incrementan los valores de los 4 números enteros.

La dirección web completa del servidor de ejemplo será: **ejemplo.com/data/logpost.php**

Entonces para lograr dicho objetivo, en el ejemplo anterior, cambiamos la función **HttpSendGet()** por **HttpSendPost()** en el evento **OnTimeout()**:

```
@OnTimeout()
{
   //
   // Comprobar si Timeout1 ha expirado.
   //

   if(Timeout1Check() == 1)
   {
      //
      // Comprobar si debemos iniciar una transmision HTTP.
      //

      if(Send == 1)
      {
         // Enviar 4 campos con valores al servidor web.
         if(HttpSendPost("/data/logpost.php",
                         "field1=%d&field2=%d&field3=%d&field4=%d",
                         Value1, Value2, Value3, Value4) == 0)
         {
            // Tranmision en curso, borrar flag.
            Send = 0
         }
      }

      // Recargar Timeout1 para generarse en proximos 30 segundos.
      Timeout1Reload(30)
   }
}
```

La función **HttpSendPost()** se ejecuta si la variable "**Send**" es "1":

```
HttpSendPost("/data/logpost.php",
             "field1=%d&field2=%d&field3=%d&field4=%d",
              Value1, Value2, Value3, Value4)
```

Si la transacción HTTP fue aceptada para ser enviada por el PLC (lo cual no quiere decir que se haya enviado con éxito), el retorno de la función es "0" y la variable "**Send**" se pone a "0" dentro del "**if**".

Al finalizar la transacción HTTP (con o sin errores), se genera el evento **OnHttpSendCompleted**, el cual ejecuta la función del mismo nombre incrementando las variables **Value** y estableciendo **Send** a 1 para el próximo envío, como lo vimos en la sección 6.7.

La función **HttpSendPost()** acepta como parámetros la cadena "**PATH**", que contiene el nombre del recurso a acceder, el **QUERY STRING** que contiene los campos a enviar y los valores de los campos o de las 4 variables **Value1**, **Value2**, **Value3** y **Value4** para enviar, las cuales se especifican al final de la lista de argumentos de la función.

Para enviar los valores de las variables a la dirección: **ejemplo.com/data/logpost.php**

Debemos formar el siguiente **Query String** (ver sección 4.2 pág. 5 para información sobre el Query String) dentro del PATH:

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Donde los símbolos %d son códigos de formato que serán reemplazados por el valor de las variables **Value1**, **Value2**, **Value3** y **Value4**. Ver documentación de función **HttpSendPost()** en sección 8.

Como es una petición POST (ver sección 4.1, pág. 4), debemos colocar el Query String por separado del PATH (ruta del recurso), resultando:

```
PATH  = /data/logpost.php
QUERY = field1=%d&field2=%d&field3=%d&field4=%d
```

Entonces, supongamos que Value1=1, Value2=2, Value3=3 y Value4=4, la función **HttpSendPost()** enviará la siguiente cadena QUERY en la petición POST al servidor web:

```
field1=1&field2=2&field3=3&field4=4
```

Luego, el servidor desde el archivo **logpost.php** obtendrá los campos **field1**, **field2**, **field3** y **field4**, y sus respectivos valores mediante lenguaje PHP u otro. Esta es la forma que se envían datos a un servidor web mediante POST.

El resto del código del proyecto fue explicado en el ejemplo GET, por lo tanto el comportamiento es el mismo.

6.10.1 Prueba del Ejemplo POST con Servidor Web y PHP

Para probar el ejemplo necesita un servidor web funcionando y un script PHP que acepte los valores enviados por el PLC y mostrarlos de alguna forma (ya sea en una página web o almacenarlos en un archivo de texto).

Como la configuración de un servidor web y PHP excede esta nota de aplicación, solo expondremos un pequeño script PHP llamado **logpost.php** que podrá subir a su página web para hacer la prueba. Dicho script almacena los valores recibidos del PLC en un archivo de texto **log.txt**, de tal forma que allí podrá ver los datos recibidos cada vez que el PLC se conecte al servidor.

Recuerde en el proyecto Pawn especificar la dirección y puerto de su servidor web en la función **HttpSenInit()** y el **PATH / QUERY** correcto al script en su servidor en la función **HttpSendPost()**, ya que el ejemplo basa la conexión en la dirección **ejemplo.com/data/logpost.php**, por ende la misma debe cambiarse por otra válida.

Script PHP **logpost.php**:

```
<?php

      // Obtener valores recibidos via HTTP POST.
      $field1 = $_POST["field1"];
      $field2 = $_POST["field2"];
      $field3 = $_POST["field3"];
      $field4 = $_POST["field4"];

      // Obtener cadena de fecha/hora actual.
      $datetime = date("Y/m/d H:i:s");

      // Armar cadena con datos recibidos.
      $logString = sprintf("Time: %s Field1: %s Field2: %s Field3: %s
                           Field4: %s\r\n", $datetime, $field1,
                           $field2, $field3, $field4);
      // Agregar cadena con datos recibidos al archivo log.txt.
      file_put_contents("log.txt", $logString, FILE_APPEND);

      // Responder con mensaje.
      echo "Log OK! Hello from PHP!\r\n"
?>
```

El script PHP anterior como podemos ver toma los valores de los campos **field** enviados por el PLC usando la instrucción **$_POST[“nombre_de_campo”]**.

Luego armamos una cadena en la variable **$logString** con la hora/fecha actual y los campos recibidos, ver función **sprintf()**.

Posteriormente con la función **file_put_contents()** guardamos el string **$logString** en el archivo “**log.txt**”, el cual cada vez que lo escribimos, agrega una línea de registro.

Finalmente le decimos al servidor que responda “**Log OK! Hello from PHP!**”. Si bien no leemos la respuesta en el PLC, podemos usarlo en el futuro para transmitir un código o mensaje.

Cuando el PLC haya enviado algunas conexiones al servidor, podremos leer el archivo **log.txt** y su contenido será similar al siguiente:

```
Time: 2016/10/11 20:28:25 Field1: 0 Field2: 0 Field3: 0 Field4: 0
Time: 2016/10/11 20:28:53 Field1: 1 Field2: 2 Field3: 3 Field4: 4
Time: 2016/10/11 20:29:22 Field1: 2 Field2: 4 Field3: 6 Field4: 8
Time: 2016/10/11 20:29:53 Field1: 3 Field2: 6 Field3: 9 Field4: 12
Time: 2016/10/11 20:30:23 Field1: 4 Field2: 8 Field3: 12 Field4: 16
Time: 2016/10/11 20:30:53 Field1: 5 Field2: 10 Field3: 15 Field4: 20
Time: 2016/10/11 20:31:23 Field1: 6 Field2: 12 Field3: 18 Field4: 24
Time: 2016/10/11 20:31:53 Field1: 7 Field2: 14 Field3: 21 Field4: 28
Time: 2016/10/11 20:32:23 Field1: 8 Field2: 16 Field3: 24 Field4: 32
Time: 2016/10/11 20:32:53 Field1: 9 Field2: 18 Field3: 27 Field4: 36
```

Notar que cada línea de **log.txt** es una conexión del PLC, a la cual se le agregó la fecha/hora de la misma y el valor de los campos Field transmitidos, que representan las variables **Value1**, **Value2**, **Value3** y **Value4**.

Observar como cada línea está separada por 30 segundos (aproximadamente por retardos de red) en el tiempo.

Si todo funciona bien, el archivo **log.txt** puede ser accedido con el navegador en la dirección:

http://ejemplo.com/data/log.txt

El ejemplo completo puede bajarlo desde la página web con el nombre **HttpSendPostPawn1.zip**, dentro de la carpeta "**Documentos**" del proyecto puede encontrar el archivo **logpost.php** y una copia de **log.txt** como referencia.

6.11 Comprobar Errores

Para comprobar errores podemos utilizar la función **HttpSendCheckValidTransaction()** la cual retorna en su salida el valor “1” si la transacción fue exitosa y “0” si contiene algún error.

Partiendo de los ejemplos anteriores (**HttpSendGetPawn1.zip** o **HttpSendPostPawn1.zip**) agregamos una nueva variable global tipo bool llamada **SendError** la cual será “1” si la transacción contiene algún error y “0” si fue exitosa.

```
//
// Variable de estado:
// Si su valor es "1" indica error luego de una transacción HTTP.
//

new SendError = 0
```

Luego definimos el siguiente código para el evento:

```
@OnHttpSendCompleted()
{
   // Actualizar flag de error.
   // Si hay error en transaccion, hacer SendError = 1.
   SendError = !HttpSendCheckValidTransaction()

   // Comprobar si no hay errores.
   if(SendError == 0)
   {
      // Incrementar variables.
      Value1 += 1
      Value2 += 2
      Value3 += 3
      Value4 += 4
   }

   // Habilitar flag para iniciar próxima transmisión.
   Send = 1
}
```

La línea de código:

```
SendError = !HttpSendCheckValidTransaction()
```

Invierte con el operador “**!**” el valor retornado por **HttpSendCheckValidTransaction()** y lo almacena en la variable **SendError**. Por lo que **SendError** tendrá el valor “1” si la transacción fue errónea.

Si no existió error, **SendError=0**, se incrementan las variables Value1 a Value4.

La función **HttpSendCheckValidTransaction()** comprueba dos condiciones que deben cumplirse al mismo tiempo en una transacción HTTP normal para que sea totalmente exitosa:

1. El código de estado de la librería HTTP SEND es “0”
2. El código de respuesta del servidor HTTP es “200”.

En caso de error, si desea analizar en profundidad, puede obtener los códigos completos retornados por las funciones **HttpSendGetLibStatus()** y **HttpSendGetResponseCode()**, para así determinar la causa del error (ver sección 8 en pág. 66).

La variable **SendError** también puede utilizarse en otras partes del proyecto ya que su alcance es global.

Los ejemplos **HttpSendGet2.zip** y **HttpSendPost2.zip** disponibles en nuestro sitio web agregan comprobación de errores.

6.12 Obtener Mensaje Respuesta del Servidor

En algunas aplicaciones puede ser útil obtener el cuerpo del mensaje de respuesta del servidor web.

El cuerpo del mensaje de respuesta es el contenido del recurso solicitado, que puede ser una página web, una cadena de texto de un archivo TXT o una imagen.

Puede utilizar esta respuesta para procesar un mensaje o código del servidor web (por ejemplo si consultó una base de datos, la respuesta puede contener un valor que le sirva para validar acceso a la misma).

Note que una página web dinámica (por ejemplo alguna con contenido PHP) puede retornar diferentes valores en cada acceso, que dependen del servidor web y esto es muy útil para comunicarse desde la web con el PLC.

El PLC puede almacenar en un buffer interno cada respuesta del servidor web. Este buffer es limitado y solo almacena los primeros 308 bytes de contenido (valor que puede variar en algunos modelos).

Para que el PLC almacene el contenido en la página web, la función **HttpSendInit()** debe configurarse con la opción **HTTP_SEND_OPT_GET_RESPONSE**.

Por ejemplo en la configuración del Cliente Web en **PlcMain()**, para conectarse al servidor "**ejemplo.com**" podemos usar en Pawn el siguiente código:

```
HttpSendInit("ejemplo.com", 80, HTTP_SEND_OPT_GET_RESPONSE, 60)
```

Pero si deseamos conectarnos a un servidor web con la dirección IP 192.168.1.15 debemos utilizar las opciones **HTTP_SEND_OPT_USE_IP** y **HTTP_SEND_OPT_GET_RESPONSE** separadas con el operador |:

```
HttpSendInit("192.168.1.15", 80,
             HTTP_SEND_OPT_USE_IP| HTTP_SEND_OPT_GET_RESPONSE, 60)
```

Por otro lado, hay dos funciones que están asociados al buffer que contiene los datos recibidos, la función **HttpSendGetBodyData()** (que permite obtener en un array los datos almacenados en el buffer) y **HttpSendGetBodyLength()** (que permite obtener la cantidad de bytes almacenados en el buffer correspondiente a la ultima respuesta).

Partiendo de los ejemplos anteriores (**HttpSendGetPawn2.zip** o **HttpSendPostPawn2.zip**) agregamos un array global del tipo con 40 elementos llamado **BodyData** en el proyecto. Esto nos va a permitir recuperar 40 bytes del buffer interno del PLC cuando reciba respuesta del servidor web.

```
//
// Array que contendrá los datos recibidos del cuerpo de mensaje de respuesta.
//

new BodyData[40]
```

Supongamos que el servidor responda ante cada petición GET o POST el siguiente cuerpo de mensaje:

```
Bienvenidos al servidor web!
```

Nuestra misión va ser leer el primer carácter de la respuesta del cuerpo del mensaje y si corresponde a la letra “B” activaremos la salida **DOUT1** del PLC.

Entonces, en el evento **OnHttpSendCompleted()** el siguiente código

```
@OnHttpSendCompleted()
{
   // Actualizar flag de error.
   // Si hay error en transaccion, hacer SendError = 1.
   SendError = !HttpSendCheckValidTransaction()

   // Comprobar si no hay errores.
   if(SendError == 0)
   {
      // Incrementar variables.
      Value1 += 1
      Value2 += 2
      Value3 += 3
      Value4 += 4

      // Comprobar si recibimos datos.
      if(HttpSendGetBodyLength() > 0)
      {
         // Leer solo 40 bytes.
         HttpSendGetBodyData(BodyData, 0, 40, false)

         // Activar/Desactivar DOUT1 si primer byte recibido es
         // igual al caracter "B".
         if(BodyData[0] == 'B')
         {
            DoutSetOn(DOUT1)
         }
         else
         {
            DoutSetOff(DOUT1)
         }
      }
   }

   // Habilitar flag para iniciar próxima transmisión.
   Send = 1
}
```

La funcion **HttpSendGetBodyData()** copiará como máximo 40 bytes del buffer interno luego de una transacción y lo guardará en el array **BodyData[]**.

```
HttpSendGetBodyData(BodyData, 0, 40, false)
```

Esta copia se realizará si la transacción fue exitosa, porque comprobamos la variable **SendError** que tiene el valor “1” si hay error (ver sección 6.11). Antes también comprobamos si tenemos datos recibidos sin leer llamando a la función **HttpSendGetBodyLength()** que retorna mayor a 0 si hay datos sin leer:

```
if(HttpSendGetBodyLength() > 0)
```

Luego si el primer elemento de **BodyData**, es decir **BodyData[0]**, contiene el valor del carácter ‘B”’activará la salida **DOUT1**.

```
        if(BodyData[0] == 'B')
        {
           DoutSetOn(DOUT1)
        }
        else
        {
           DoutSetOff(DOUT1)
        }
```

Note que ‘B’ equivale al valor “66” en código ASCII, por lo tanto si el servidor retorna un cuerpo de mensaje con la primera letra igual a “B”, la salida **DOUT1** será activada.

Este es un ejemplo trivial, pero puede realizar mayores comprobaciones para operaciones complejas.

Los ejemplos **HttpSendGetPawn3.zip** y **HttpSendPostpawn3.zip** disponibles en nuestro sitio web procesamiento de cuerpo de mensaje de respuesta visto en esta sección.

Tip: En **StxLadder**, en el menú “**Herramientas > Conversión > ASCII**” tiene un conversor ASCII que puede serle de utilidad en sus proyectos.

6.13 Fin de Ejemplos

Estas son las nociones básicas para usar el Cliente Web. Puede usar estos ejemplos y otros disponibles en nuestro sitio de Internet como base para sus proyectos.

Recomendamos altamente:

- Leer Referencia de Funciones en Lenguaje Pawn, página 66
- Ir a Slicetex.com y descargar programas de ejemplos desde página web.

7 Referencia de Componentes en Lenguaje Ladder

En esta sección se detalla a modo general los componentes disponibles en lenguaje Ladder para utilizar el Cliente Web del PLC.

Recomendamos leer primero el **Ejemplo de Uso Inicial en Lenguaje Ladder** en página 7.

En lenguaje Ladder es muy simple utilizar el Cliente Web. Básicamente hay 4 clases de componentes disponibles:

- Componentes para configurar el cliente y obtener estado de librería.
- Componentes para activar eventos.
- Componentes para realizar peticiones GET y POST.
- Componentes para obtener datos o información proveniente del servidor web.

Importante:

Recuerde que la documentación detallada y actualizada de cada componente está disponible en el mismo entorno **StxLadder**. Para acceder a dicha documentación, solo tiene que insertar el componente, luego seleccionarlo con el botón-derecho del mouse, y posteriormente acceder al ítem "**Ver descripción del componente …**" del menú contextual desplegable del componente.

7.1 Componentes de Configuración y Estado

7.1.1 HTTP SEND – INIT

Agrega un componente que permite inicializar y configurar los parámetros del PLC para que pueda conectarse a un servidor web o HTTP (Hypertext Transfer Protocol) remoto y así realizar transacciones web con otros componentes de la librería.

Este componente debe llamarse solo cuando sea necesario configurar los parámetros HTTP de la librería HTTP Send. En general se llama desde el diagrama de inicio (Inicio.sld) o una sola vez desde otro diagrama.

Si el componente no es configurado correctamente, no podrá realizar transacciones web.

El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

El valor del puerto de salida "**ENO**" es "1" si pudo inicializarse correctamente la librería HTTP. De lo contrario retorna "0" (puede significar algún error en los parámetros de configuración del componente o algún error interno).

Configuración del Componente

Desde las propiedades del componente deben especificarse los siguientes campos:

- **Dirección servidor HTTP**: Nombre del servidor HTTP. Si es una dirección IP, tildar la casilla "**IP**" ubicada al costado del campo. Puede utilizar el botón "**Convertir**" para obtener la dirección IP de un servidor HTTP.
- **Puerto HTTP**: Numero de puerto TCP del servidor HTTP.
- **TImeout**: Segundos de espera para obtener respuesta de un servidor HTTP luego de una conexión.
- **Get Response**: Si se selecciona esta casilla, el cliente obtendrá el cuerpo del mensaje o página web recibida desde el servidor y la almacenará en un buffer interno que luego estará disponible para leer. Si solo desea transmitir datos GET/POST al servidor, no es necesario que tilde esta casilla.

Importante:

- Si la dirección del servidor HTTP se especifica como una IP, recuerde tildar la casilla "**IP**". Para utilizar un nombre de dominio debe tener configurada correctamente la dirección del servidor DNS en el PLC (ver menú "**PLC / Configurar PLC"** en StxLadder).

Ejemplo:

Para conectarse a **google** y no almacenar respuesta, podemos configurar el componente de la siguiente manera:

- **Dirección HTTP**: google.com
- **IP**: No seleccionar.
- **Puerto HTTP**: 80

- **Timeout**: 60
- **Get Response**: No seleccionar.

Para conectarse a un servidor local en puerto 82 con IP y almacenar respuesta, podemos configurar el componente de la siguiente manera:

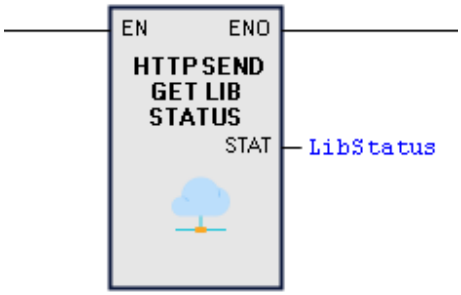
- **Dirección HTTP**: 192.168.1.15
- **IP**: Seleccionar.
- **Puerto HTTP**: 82
- **Timeout**: 60
- **Get Response**: Seleccionar.

Notas / Consejos:

- Este componente no realiza ninguna conexión, solo configura la librería HTTP para futuras transacciones HTTP que pueden realizarse con componentes **HttpSendGet** o **HttpSendPost** por ejemplo.
- Librería HTTP Send o cliente web, está pensada para enviar datos mediante peticiones GET/POST a un servidor, opcionalmente puede guardar la respuesta del servidor en un buffer interno con memoria limitada (ver componentes **HttpSendGetBodyLength** y **HttpSendGetBodyData**).

7.1.2 HTTP SEND - GET LIB STATUS

Agrega un componente que permite obtener el estado de la librería **HTTP Send** del PLC.



Este componente debe utilizarse para determinar el estado de una transacción **HTTP** o identificar errores.

Siempre debe comprobar el código de estado antes de realizar una operación. Un código de estado negativo, implica una condición de error. Un código positivo implica un estado particular. El valor cero significa que el cliente recibió respuesta del servidor.

Cuando realiza una transacción **HTTP**, el PLC comienza a negociar con el servidor **HTTP (WEB)** la transmisión de la petición (GET o POST) y la respuesta del mismo. Este proceso puede tomar desde decenas de milisegundos, hasta 125 segundos (o más) dependiendo del tiempo de respuesta de los servidores de internet.

Para evitar bloquear el PLC hasta que la respuesta del comando sea recibida (proceso que tarda decenas de segundos), puede utilizar este componente para leer el estado de la transacción. Usted puede leer el estado de transmisión cada X tiempo y luego setear una variable que indique su recepción. Bajo este escenario, es útil el componente **HttpSendGetLibStatus**.

Alternativamente, puede utilizar el evento **OnHttpSendCompleted** que le indica el fin de una transacción HTTP, pero para determinar si la misma fue exitosa, debe utilizar el componente **HttpSendGetLibStatus** dentro del evento para validar la misma.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- La salida "**ENO**" del componente es una copia del valor de la entrada "**EN**".
- La salida "**STAT**" del componente devuelve en una variable tipo **Int32** el estado de la librería. Un código de estado negativo, implica una condición de error. Un código positivo implica un estado particular. El valor cero significa que el cliente recibió respuesta del servidor **HTTP**.

  Valores de retorno:

  - (127): Requerimiento en proceso de envío.
  - (126): La librería no fue inicializada con el componente **HttpSendInit**.
  - (124): La librería fue inicializada.
  - (123): La conexión fue cerrada por el cliente.
  - (122): Datos transmitidos con éxito.
  - (0): Respuesta recibida con éxito del servidor, listo.
  - (-1): Error, conexión abortada.

- (-2): Error, timeout en conexión. Imposible conectar.
- (-3): Error, conexión cerrada por el host.
- (-4): Error, timeout en transacción HTTP.
- (-5): Error, la conexión no fue asignada.
- (-6): Error, ver código de retorno de función/componente.
- (-7): Error, no se puedo resolver el nombre del host.
- (-8): Error, estado de librería inválido.

Importante:

- Recuerde inicializar la librería con el componente **HttpSendInit** para evitar el código “126”.

Notas:

- Los códigos más importantes son el 127 (cuando hay una transacción en curso) y el 0 (cuando la respuesta del servidor está disponible). Para comprobar una situación de error, solo hay que verificar si el código devuelto es negativo.

- Utilice el componente **HttpSendCheckValidTransaction** para una detección de errores simple.

7.1.3 HTTP SEND – CHECK VALID TRANSACTION

Agrega un componente que comprueba si la transacción HTTP del tipo GET o POST fue válida o exitosa.

Este componente comprueba si el código de estado de la librería HTTP SEND es “0” y si el código de respuesta del servidor HTTP es “200”. Si ambas condiciones se dan luego de una transacción HTTP, indica que la conexión fue exitosa.

Esto es muy útil para comprobar errores fácilmente luego de una transacción HTTP, ya sea desde el evento **OnHttpSendCompleted** o desde cualquier otro diagrama.

En caso de error, puede obtener los códigos completos retornados por los componentes **HttpSendGetLibStatus** y **HttpSendGetResponseCode**, para así determinar la causa del error.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada “**EN**” es 1.

Salidas:

- La salida “**ENO**” del componente es “1” si la ultima transacción HTTP fue válida. De lo contrario es “0” si contiene algún error.

Notas:

- En caso de error, si desea conocer la causa, utilice los componentes **HttpSendGetLibStatus** y **HttpSendGetResponseCode** para obtener los códigos retornados.

7.2 Componentes Para Eventos

7.2.1 HTTP SEND – SET COMPLETED EVENT

Agrega un componente que permite activar el evento "**"OnHttpSendCompleted"**".

El evento **OnHttpSendCompleted** será llamado al finalizar una transacción HTTP tipo GET o POST. La transacción puede ser exitosa o no, para ello debe utilizar el componente **HttpSendGetLibStatus** y así determinar el estado de la misma.

Al dispararse el evento, el PLC llama al diagrama asignado para manejar el evento. Dicho evento se define desde el **Explorador de Proyecto** del entorno **StxLadder**. El programador procesará el evento con la lógica especificada en el diagrama.

El evento **OnHttpSendCompleted** es un mecanismo muy útil para determinar cuándo una transacción HTTP concluye.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- La salida "**STAT**" del componente es verdadera si el evento pudo configurarse con éxito.

Notas:

- Recuerde crear el diagrama para procesar el evento.
- Se recomienda utilizar una sola vez este componente, llamándolo desde el diagrama **Inicio.sld**.

Consejos:

- Puede utilizar el evento **OnHttpSendCompleted** para determinar el fin de una transacción HTTP y obtener datos de la misma, como código de respuesta del servidor, cuerpo de mensaje recibido, estado de la librería, etc.
- Si necesita comenzar una transacción nuevamente, desde el evento **OnHttpSendCompleted** puede activar una variable que señalice el fin de una transacción y así desde otro punto del programa, comenzar una transacción nueva.

7.2.2 HTTP SEND – CLR COMPLETED EVENT

Agrega un componente que permite desactivar el evento "**OnHttpSendCompleted**".

El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".

7.3 Componentes Para Realizar Peticiones GET y POST

7.3.1 HTTP SEND – GET

Agrega un componente que permite iniciar una transacción GET con un servidor web.

Este componente tiene dos finalidades principales:

- Enviar datos a una página web mediante un **Query String** codificado en el URL o **PATH**, por ejemplo: **http://ejemplo.com/log.php?field1=value1&field2=value2**.
- Pedir una página web simplemente: **http://ejemplo.com/pagina.html**

Mediante el envío de datos a través de un **Query String**, puede transferir datos desde el PLC al servidor web, como por ejemplo valores analógicos, valores enteros, mensajes de texto, etc. Dichos datos pueden ser almacenados o recibidos por el servidor web para generar páginas y visualizar la información o solo para registrarla, ya sea con PHP, HTML, Base de Datos SQL, etc.

Al enviar una petición GET, el servidor puede responder con un mensaje o contenido (pagina web). Si en el componente **HttpSendInit** se configuró con la opción "**Get Response**", el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño) para luego poder recuperarlo con el componente **HttpSendGetBodyData.** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **PATH** = Es el recurso a cargar en el servidor web, y debe empezar con la barra invertida "/". La misma puede contener un formato dinámico (ver sección **Formato**)
- Existen 4 entradas más, cuyo uso es opcional y permiten especificar las Variable a leer e imprimir en la cadena **PATH** antes de enviarla al servidor web.

  Los puertos de entrada son "**IN1**", "**IN2**", "**IN3**" e "**IN4**". Dentro de las propiedades del componente, puede especificar las variables apuntadas por dichos puertos.

  No es necesario utilizar todos los puertos al mismo tiempo, ni seguir un orden especial.

  El tipo de variable o constante que puede seleccionar es amplio, puede ser **Int32**, **Float**, **String**, etc.

  Lea la sección de **Formato** para imprimir valores de variables.

Salidas:

- El valor del puerto de salida "**ENO**" es "1" si la operación fue aceptada por el PLC y está en curso para su transmisión al servidor web (esto no quiere decir que la transmisión sea exitosa, para ello debe comprobar el estado de la librería con el componente **HttpSendGetLibStatus**). Si es "0" la salida **ENO**, la operación no fue aceptada por algún error (librería ocupada en otra transacción pendiente, error de memoria, librería no inicializada, etc).

Ejemplo 1:

Si desea obtener la pagina web de un servidor llamada "**estado.html**" del servidor **http://ejemplo.com/info/estado.html**, debe configurar el componente de la siguiente manera:

- **PATH**: /info/estado.html
- **Variables:** Vacio
- **Entradas**: No utilizar.

Evidentemente, antes con el componente **HttpSendInit** debe especificar **Direccion HTTP = ejemplo.com** y tildar la opción "**Get Response**". Luego con el componente **HttpSendGetBodyData** puede obtener el buffer con la respuesta.

Ejemplo 2:

Si desea enviar dos valores dinámicos (que dependen de una variable) a un servidor, por ejemplo **http://ejemplo.com/log.php?field1=value1&field2=value2**, donde **value1** y **value2** son dos cadenas de texto con cualquier valor numérico (1, 2, 3…etc) puede hacer:

- **PATH**: /log.php?field1=%d&field2=%d
- **Variables:** IN1,IN2
- **Entradas**: Usar IN1 e IN2, y conectar a una variable entera tipo **Int32** con valor numérico.

Si las variables conectadas a **IN1** e **IN2** tienen el valor 1 y 2 respectivamente, el PLC enviará la siguiente cadena **PATH** = /log.php?field1=1&field2=2

Notar como la cadena PATH puede contener formato con los símbolos "**%d**" (explicado más adelante) el cual le permite reemplazar por valores numéricos, y además como se utiliza la notación "URL encoded" o Query String para enviar los datos al servidor web:

Query String:

Un **Query String** consiste una cadena de la forma: **campo1=valor1&campo2=valor2&campoX=valorX**

El **Query String** se separa del recurso o pagina web para una petición **GET**, mediante el símbolo "**?**" y cada valor se especifica con un campo, separado por un "**=**". Si hay varios campos, los mismos se separan mediante el símbolo "**&**".

Ejemplo, enviar valor de RPM=598 y Temperatura=25.4:
RPM=598&Temperatura=25.4

Ejemplo, enviar Nombre "Ernesto Sabato" y Mail "sabato@gmail.com":
Nombre=Ernesto+Sabato&Mail=sabato%40gmail.com

Algunos caracteres (ya sea en el campo o valor) deben codificarse en “porcentaje”, también llamado “URL Encode”, estos caracteres son el espacio (por +) la arroba (por %40), como en el ejemplo anterior.

Las letras (A-Z y a-z), números (0-9), y caracteres “*”, “-“, “_”, “.”, “~” no necesitan ser codificados ya que son caracteres no reservados. Para el espacio puede usarse (+) o (%20), pero para el resto se usa el signo "%" seguido del número hexadecimal ASCII correspondiente al carácter codificado.

En **StxLadder**, en el menú “**Herramientas > Conversión > URL Encode**” hay un conversor para convertir cadenas a formato URL encoded, de tal forma que le sea fácil enviar un Query String complejo.

Formato:

Para enviar cadenas con formato, es necesario especificar “***códigos de formato***” en el campo “**PATH**” de las propiedades del componente. Las variables se especifican en el campo “**Variables**” del componente.

Los ***códigos de formato*** indican cómo debe interpretarse e imprimirse una variable.

Los ***códigos de formato*** están precedidos por un símbolo “%”. Por ejemplo “%d” indica que en ese lugar del texto, debe imprimirse una variable entera.

Cada “***código de formato***” lee la variable apuntada por algún puerto de entrada. Los puertos que deben leerse se especifican en las propiedades del componente, en una caja de texto adyacente a la cadena **PATH** a enviar.

Ejemplo, para imprimir los RPM de un motor en la cadena, cuyo valor se especifica en una variable entera apuntada por el puerto “**IN1**”, hacemos:

“RPM = %d” → “IN1”

En la cadena, se reemplazará el ***código de formato*** %d por el valor de la variable apuntada por el puerto “**IN1**”.

Si deseamos imprimir el valor de RPM y PRESION, podemos hacer:

“RPM = %d PRESION = %d” → “**IN1**,**IN2**”

Notemos, como separamos con comas los puertos **IN1** e **IN2**.

Si tenemos un valor de tensión y deseamos imprimirlo como punto flotante, podemos hacer:

“Tension: %.3f [V]” → “**IN1**”

La cadena resultante será la siguiente: “**Tension: 11.345 [V]”.**

Note como fue especificada la precisión, mostrando solo 3 dígitos decimales.

Códigos de formatos disponibles:

Nombre	Tipo	Descripción
%d	Entero	Imprime un entero con signo.
%u	Entero	Imprime un entero sin signo.
%x	Entero	Imprime un entero con notación hexadecimal.
%X	Entero	Imprime un entero con notación hexadecimal en mayúscula.
%f	Flotante	Imprime un numero decimal o punto flotante.
%s	Cadena	Imprime una cadena de caracteres.
%c	Caracter	Imprime un solo carácter.

Un Entero es un valor tipo **Int32**.
Un Flotante es un valor tipo **Float**
Una Cadena es un valor tipo **String**.

Alineación y Relleno:

Se antepone un numero entre el "%" y el código. El número especifica la cantidad de espacios que deberán ser rellenados antes de imprimir la variable. Si el digito es negativo "-", la alineación será hacia la izquierda.

Si antes del digito de alineación, se antepone el numero 0, en vez de rellenar con espacios, la alineación será rellenada con ceros.

Precisión:

En un número flotante, es posible especificar la cantidad de dígitos decimales a imprimir (por defecto es dos). Por ejemplo, "%.000f", imprimimos un numero decimal con tres dígitos luego del punto solamente

Ejemplos de Formatos:

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%d	3	3							
%4d	3				3				
%04d	3	0	0	0	3				
%-4d	3	3							
%04d	-3	-	0	0	3				
%4d	-3			-	3				
%03.000f	5.12345	0	5	.	1	2	3		
%f	5.12345	5	.	1	2				
%.4f	5.12345	5	.	1	2	3	4	5	
%.02f	3.1	3	.	1	0				

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%x	255	f	f						
%X	255	F	F						
%s	"HOLA"	H	O	L	A				
%c	66	B							
%c	'B'	B							
%s	0	(	n	u	l	l	)		

En las tablas, el término "Argumento" se refiere al valor del puerto (**IN1**, **IN2**, etc).

Notas / Consejos:

- Recuerde incluir el carácter "/" al inicio del PATH.
- Utilice el componente **HttpSendGetLibStatus** en conjunto con el evento **OnHttpSendCompleted** para determinar el fin de una transacción HTTP.

- Utilice el componente **HttpSendCheckValidTransaction** para una detección de errores simple.

- El componente **HttpSendPost** es similar a este componente, solo que envía los datos mediante una petición POST, esto significa que el Query String no se coloca en el URL o PATH. Su uso depende del servidor web.

- Si usted es un programador en lenguaje C, el uso de este componente le resultara similar a la función "**printf()"** de dicho lenguaje.

7.3.2 HTTP SEND – POST

Agrega un componente que permite iniciar una transacción POST con un servidor web.

Este componente tiene la finalidad principale de enviar datos a una página web mediante un **Query String**, por ejemplo: **field1=value1&field2=value2**.

Mediante el envío de datos a través de un **Query String**, puede transferir datos desde el PLC al servidor web, como por ejemplo valores analógicos, valores enteros, mensajes de texto, etc. Dichos datos pueden ser almacenados o recibidos por el servidor web para generar páginas y visualizar la información o solo para registrarla, ya sea con PHP, HTML, Base de Datos SQL, etc.

Al enviar una petición POST, el servidor puede responder opcionalmente con un mensaje o contenido (pagina web). Si en el componente **HttpSendInit** se configuró con la opción "**Get Response**", el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño) para luego poder recuperarlo con el componente **HttpSendGetBodyData.** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **PATH** = Es el recurso a cargar en el servidor web, y debe empezar con la barra invertida "/".
- **QUERY** = Es la cadena Query String a enviar al servidor web, que contiene los datos a transferir al servidor. La cadena debe codificarse en formato URL o URL Encoded. La misma puede contener un formato dinámico (ver sección **Formato**)
- Existen 4 entradas más, cuyo uso es opcional y permiten especificar las Variable a leer e imprimir en la cadena **PATH** antes de enviarla al servidor web.

  Los puertos de entrada son "**IN1**", "**IN2**", "**IN3**" e "**IN4**". Dentro de las propiedades del componente, puede especificar las variables apuntadas por dichos puertos.

  No es necesario utilizar todos los puertos al mismo tiempo, ni seguir un orden especial.

  El tipo de variable o constante que puede seleccionar es amplio, puede ser **Int32**, **Float**, **String**, etc.

  Lea la sección de **Formato** para imprimir valores de variables.

Salidas:

- El valor del puerto de salida "**ENO**" es "1" si la operación fue aceptada por el PLC y está en curso para su transmisión al servidor web (esto no quiere decir que la transmisión sea exitosa, para ello debe comprobar el estado de la librería con el componente **HttpSendGetLibStatus**). Si es "0" la salida **ENO**, la operación no fue aceptada por algún error (librería ocupada en otra transacción pendiente, error de memoria, librería no inicializada, etc).

Ejemplo 1:

Si desea enviar dos valores dinámicos (que dependen de una variable) a un servidor, por ejemplo **http://ejemplo.com/log.php,** con un **Query String** del tipo **field1=value1&field2=value2**, donde **value1** y **value2** son dos cadenas de texto con cualquier valor numérico (1, 2, 3…etc) puede hacer:

- **PATH**: /log.php
- **QUERY:** field1=%d&field2=%d
- **Variables:** IN1,IN2
- **Entradas**: Usar IN1 e IN2, y conectar a una variable entera tipo **Int32** con valor numérico.

Si las variables conectadas a **IN1** e **IN2** tienen el valor 1 y 2 respectivamente, el PLC enviará la siguiente cadena **QUERY** = field1=1&field2=2

Notar como la cadena QUERY puede contener formato con los símbolos "**%d**" (explicado más adelante) el cual le permite reemplazar por valores numéricos, y además como se utiliza la notación "URL encoded" o Query String para enviar los datos al servidor web:

Query String:

Un **Query String** consiste una cadena de la forma: **campo1=valor1&campo2=valor2&campoX=valorX**

En el **Query String** cada valor se especifica con un campo, separado por un "**=**". Si hay varios campos, los mismos se separan mediante el símbolo "**&**".

Ejemplo, enviar valor de RPM=598 y Temperatura=25.4:
RPM=598&Temperatura=25.4

Ejemplo, enviar Nombre "Ernesto Sabato" y Mail "sabato@gmail.com":
Nombre=Ernesto+Sabato&Mail=sabato%40gmail.com

Algunos caracteres (ya sea en el campo o valor) deben codificarse en "porcentaje", también llamado "URL Encode", estos caracteres son el espacio (por +) la arroba (por %40), como en el ejemplo anterior.

Las letras (A-Z y a-z), números (0-9), y caracteres "*", "-", "_", ".", "~" no necesitan ser codificados ya que son caracteres no reservados. Para el espacio puede usarse (+) o (%20), pero para el resto se usa el signo "%" seguido del número hexadecimal ASCII correspondiente al carácter codificado.

En **StxLadder**, en el menú "**Herramientas > Conversión > URL Encode**" hay un conversor para convertir cadenas a formato URL encoded, de tal forma que le sea fácil enviar un Query String complejo.

Formato:

Para enviar cadenas con formato, es necesario especificar "***códigos de formato***" en el campo "**QUERY**" de las propiedades del componente. Las variables se especifican en el campo "**Variables**" del componente.

Los ***códigos de formato*** indican cómo debe interpretarse e imprimirse una variable.

Los ***códigos de formato*** están precedidos por un símbolo "%". Por ejemplo "%d" indica que en ese lugar del texto, debe imprimirse una variable entera.

Cada "***código de formato***" lee la variable apuntada por algún puerto de entrada. Los puertos que deben leerse se especifican en las propiedades del componente, en una caja de texto adyacente a la cadena **QUERY** a enviar.

Ejemplo, para imprimir los RPM de un motor en la cadena, cuyo valor se especifica en una variable entera apuntada por el puerto "**IN1**", hacemos:

"RPM = %d" → "IN1"

En la cadena, se reemplazará el ***código de formato*** %d por el valor de la variable apuntada por el puerto "**IN1**".

Si deseamos imprimir el valor de RPM y PRESION, podemos hacer:

"RPM = %d PRESION = %d" → "**IN1**,**IN2**"

Notemos, como separamos con comas los puertos **IN1** e **IN2**.

Si tenemos un valor de tensión y deseamos imprimirlo como punto flotante, podemos hacer:

"Tension: %.3f [V]" → "**IN1**"

La cadena resultante será la siguiente: "**Tension: 11.345 [V]".**

Note como fue especificada la precisión, mostrando solo 3 dígitos decimales.

Códigos de formatos disponibles:

Nombre	Tipo	Descripción
%d	Entero	Imprime un entero con signo.
%u	Entero	Imprime un entero sin signo.
%x	Entero	Imprime un entero con notación hexadecimal.
%X	Entero	Imprime un entero con notación hexadecimal en mayúscula.
%f	Flotante	Imprime un numero decimal o punto flotante.
%s	Cadena	Imprime una cadena de caracteres.
%c	Caracter	Imprime un solo carácter.

Un Entero es un valor tipo **Int32**.
Un Flotante es un valor tipo **Float**
Una Cadena es un valor tipo **String**.

Alineación y Relleno:

Se antepone un numero entre el "%" y el código. El número especifica la cantidad de espacios que deberán ser rellenados antes de imprimir la variable. Si el digito es negativo "-", la alineación será hacia la izquierda.

Si antes del digito de alineación, se antepone el numero 0, en vez de rellenar con espacios, la alineación será rellenada con ceros.

Precisión:

En un número flotante, es posible especificar la cantidad de dígitos decimales a imprimir (por defecto es dos). Por ejemplo, "%.000f", imprimimos un numero decimal con tres dígitos luego del punto solamente

Ejemplos de Formatos:

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%d	3	3							
%4d	3				3				
%04d	3	0	0	0	3				
%-4d	3	3							
%04d	-3	-	0	0	3				
%4d	-3			-	3				
%03.000f	5.12345	0	5	.	1	2	3		
%f	5.12345	5	.	1	2				
%.4f	5.12345	5	.	1	2	3	4	5	
%.02f	3.1	3	.	1	0				

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%x	255	f	f						
%X	255	F	F						
%s	"HOLA"	H	O	L	A				
%c	66	B							
%c	'B'	B							
%s	0	(	n	u	l	l	)		

En las tablas, el término “Argumento” se refiere al valor del puerto (**IN1**, **IN2**, etc).

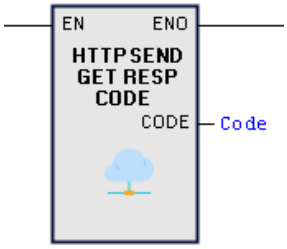
Notas / Consejos:

- Recuerde incluir el carácter “/” al inicio del PATH.
- Utilice el componente **HttpSendGetLibStatus** en conjunto con el evento **OnHttpSendCompleted** para determinar el fin de una transacción HTTP.
- Utilice el componente **HttpSendCheckValidTransaction** para una detección de errores simple.
- El componente **HttpSendGet** es similar a este componente, solo que envía los datos mediante una petición GET, esto significa que el Query String se coloca en el URL o PATH. Su uso depende del servidor web.
- Si usted es un programador en lenguaje C, el uso de este componente le resultara similar a la función “**printf()”** de dicho lenguaje.

7.4 Componentes Para Obtener Información y Datos

7.4.1 HTTP SEND - GET RESPONSE CODE

Agrega un componente que permite obtener el código de estado de respuesta devuelto por un servidor HTTP luego de una transacción HTTP exitosa tipo GET o POST.



El código de respuesta de un servidor permite determinar si el servidor aceptó o no nuestra petición, y cuál es la posible causa del error en caso que no la acepte.

No debe confundir el código de respuesta de un servidor HTTP con el código de estado devuelto por el componente **HttpSendGetLibStatus**, el cual es un código interno de la librería. Por ejemplo, podemos tener una transacción exitosa (la conexión se logró con el servidor y obtuvimos respuesta) pero el servidor pudo devolver un código de error 404 (pagina no encontrada). Por ello, en algunas situaciones, es útil conocer el código de respuesta de un servidor web.

El código de respuesta de un servidor web es un número entero que está definido por el protocolo HTTP (RFC 2616). El código de estado devuelto para peticiones correctas es el 200.

Algunos ejemplos:

- 200 : Respuesta estándar para peticiones correctas.
- 301 : Movido permanentemente.
- 400 : Error en sintaxis.
- 404 : Recurso no encontrado.
- 1xx : Respuestas informativas
- 2xx : Peticiones correctas
- 3xx : Redirecciones
- 4xx : Errores del cliente
- 5xx : Errores de servidor

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- La salida "**ENO**" del componente es una copia del valor de la entrada "**EN**".
- La salida "**CODE**" del componente devuelve en una variable tipo **Int32** el código de respuesta del servidor de la última transacción HTTP.

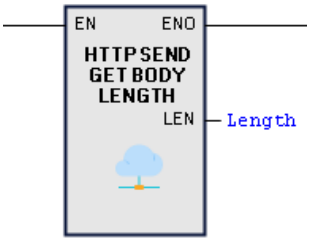
Notas:

- No es necesario que compruebe este código para transacciones normales, pero si usted debe asegurarse una transmisión correcta, debería leer dicho código en cada respuesta del servidor.

- Utilice el componente **HttpSendCheckValidTransaction** para una detección de errores simple.
- El código 200 es el más importante, indica éxito en la transacción en ambas partes, cliente y servidor.

HTTP SEND - GET BODY LENGTH

Agrega un componente que permite obtener la cantidad de bytes del cuerpo del mensaje recibido de un servidor luego de una transacción HTTP exitosa tipo GET o POST.



El cuerpo del mensaje de una respuesta HTTP es la página web o recurso solicitado al servidor.

Al enviar una petición GET o POST, el servidor puede responder con un mensaje o contenido (pagina web). Si en el componente **HttpSendInit** se configuró con la opción "**Get Response**", el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño) para luego poder recuperarlo con el componente **HttpSendGetBodyData.** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.

Si la longitud del mensaje recibido por el PLC excede su buffer interno, el resto de los datos recibidos son descartados, almacenándose solo la capacidad del buffer. Este componente devolverá la cantidad de bytes almacenados no leídos.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- La salida "**ENO**" del componente es una copia del valor de la entrada "**EN**".
- La salida "**LEN**" del componente devuelve en una variable tipo **Int32** la cantidad de bytes almacenados del cuerpo de mensaje recibido del servidor en la última transacción HTTP. Si el valor es 0, significa que no hay datos recibidos o sin leer.

  Los valores negativos tienen el siguiente significado:

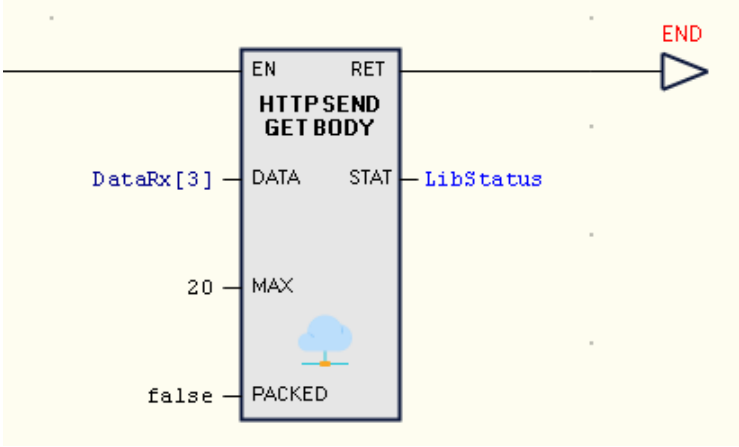
  - (-1) : Error, transacción invalida, no hay datos disponibles.
  - (-4) : Error, librería no inicializada.

Notas:

- Recuerde configurar **HttpSendInit** con la opción "**Get Response**" para obtener el cuerpo del de respuesta de un servidor HTTP y almacenarlo.
- La longitud máxima del buffer interno del PLC es 308 bytes, a menos que se indique lo contrario.

7.4.2 HTTP SEND - GET BODY DATA

Agrega un componente que permite obtener el cuerpo del mensaje recibido en la última transacción HTTP tipo GET o POST.



El cuerpo del mensaje de una respuesta HTTP es la página web o recurso solicitado al servidor.

El componente lee el buffer interno del PLC que almacena el cuerpo del mensaje recibido.

Al enviar una petición GET o POST, el servidor puede responder con un mensaje o contenido (pagina web). Si en el componente **HttpSendInit** se configuró con la opción "**Get Response**", el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño)**.** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.

Si la longitud del mensaje recibido por el PLC excede su buffer interno, el resto de los datos recibidos son descartados, almacenándose solo la capacidad del buffer.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- La entrada "**DATA**" debe apuntar a un array del tipo **Int32_Array**, donde se copiarán los bytes recibidos. El componente le preguntara a partir de que elemento se copiaran los bytes recibidos.
- La entrada "**MAX**" establece la cantidad de bytes que se copiaran a **DATA**, máximo 308. Si el valor es "0" se copiaran todos los bytes recibidos (debe asegurarse de no sobrepasar el tamaño de **DATA**).
- La entrada "**PACKED**", si es "true" se empaquetaran 4 bytes recibidos por cada elemento de **DATA**. Si es "false" se copiara un byte por cada elemento de **DATA**.

Salidas:

- La salida "**RET**" es verdadera o "**1**" si pudieron copiarse bytes a **DATA**. Si es "**0**" no se copiaron bytes porque existió un error o el buffer está vacío. Puede encontrar más información del error en el puerto "**STAT**".
- En la salida "**STAT**" se retorna un código de error en una variable tipo **Int32** (uso opcional):
    - El valor retornado es "**0**" si se copiaron bytes a **DATA**.
    - Si el valor retornado es "**-1**", no se copiaron bytes debido a que el buffer de recepción esta vacio.

- El valor es "**-2**" si el tamaño especificado en "**MAX**" es incorrecto.
- El valor es "**-4**" si la dirección de **DATA** no es válida.

Ejemplo:

En general, este componente se llama al inicio del diagrama que maneja el evento "**OnHttpSendCompleted**", que se llama cuando una transacción HTTP fue completada (exitosa o con errores).

Entonces, ubique este componente en el diagrama del evento "**OnHttpSendCompleted**", obtenga los datos recibidos en el puerto **DATA** y luego puede procesarlo o utilizarlo desde otros diagramas, compartiendo los datos con un array global.

Puede llamar a este componente constantemente y verificar si hay nuevos datos cuando la salida "**STAT**" sea igual a "**0**".

Datos Empaquetados (PACKED)

Es posible especificar si los datos a copiar en el puerto **DATA** están o no empaquetados (**PACKED**). En las propiedades del componente se puede tildar la casilla "**Packed data**" para declarar que los datos serán empaquetados previamente.

Cuando los datos están empaquetados en **DATA**, significa que hay 4 bytes por cada elemento **Int32** contenido en el array. De lo contrario solo se escribe un byte por cada elemento.

Si recibió 4 bytes, y **PACKED** = true, solo le hará falta un elemento **Int32** de **DATA** para almacenarlo. De lo contrario se utilizarán 4 elementos **Int32** del **ARRAY** para copiarlos.

El empaquetamiento es útil para ahorrar memoria RAM.

Consejos:

- Recibir el cuerpo del mensaje de una respuesta HTTP solo es necesario si desea obtener alguna información particular del servidor web, de lo contrario no necesita usar este componente.
- Puede usar en conjunto con el componente **HttpSendGetResponseCode** para determinar si el contenido recibido corresponde a un recurso valido dentro del servidor web.

Notas:

- Recuerde configurar **HttpSendInit** con la opción "**Get Response**" para obtener el cuerpo del de respuesta de un servidor HTTP y almacenarlo.
- La longitud máxima del buffer interno del PLC es 308 bytes, a menos que se indique lo contrario.
- Una vez que el buffer de recepción del PLC es leído (total o parcialmente), no será posible leerlo nuevamente hasta la próxima transacción, ya que le indicará que no hay datos sin leer.
- No se recomienda utilizar el argumento **MAX** = 0, a menos que se utilice un array **DATA** con la cantidad de elementos suficientes para almacenar todo el buffer de recepción completo.

8 Referencia de Funciones en Lenguaje Pawn

En esta sección se detalla a modo general las funciones disponibles en lenguaje Pawn para utilizar el Cliente Web del PLC.

Recomendamos leer primero el **Ejemplo de Uso Inicial en Lenguaje Pawn** en página 27.

En lenguaje Pawn es muy simple utilizar el Servidor Web. Básicamente hay 4 clases de funciones disponibles:

- Funciones para configurar el cliente y obtener estado de librería.
- Funciones para activar eventos.
- Funciones para realizar peticiones GET y POST.
- Funciones para obtener datos o información proveniente del servidor web.

En la página siguiente se describen dichas funciones en detalle.

8.1 Funciones de Configuración, Estado y Control

HttpSendInit(Address[ ], Port=80, Options=HTTP_SEND_OPT_DEFAULT, Timeout=30): Inicializa y configura los parámetros del Cliente Web.		
Argumentos	**Tipo**	**Descripción**
Address[ ]	E	Nombre del Servidor Web o dirección IP (si se utiliza la opción **HTTP_SEND_OPT_USE_IP**)
Port	E	Puerto TCP del servidor web. Por defecto 80.
Options	E	Opciones de configuración. Por defecto **HTTP_SEND_OPT_DEFAULT**. Opciones válidas: • **HTTP_SEND_OPT_DEFAULT:** Default, equivale a 0. • **HTTP_SEND_OPT_USE_IP**: Usar IP como **Address**. • **HTTP_SEND_OPT_GET_RESPONSE**: Obtener cuerpo de mensaje. • **HTTP_SEND_OPT_FORCE_RESOLV:** Resolver nombre de servidor en cada conexión. Útil para direcciones con IP dinámicas.
Timeout		Tiempo en segundos de timeout para esperar una respuesta cuando existe conexión al servidor (de lo contrario se espera 60 si no se puede conectar al servidor). Máximo 125 segundos. Por defecto 30 segundos.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, falla en inicialización, memoria insuficiente.
-10	S	Error, Cliente Web conectado actualmente al servidor, desconectar primero.
Notas		**Descripción**
1		Esta función debe llamarse una vez, antes de usar cualquier otra función del Cliente Web.

Ejemplo 1:

```
// Inicializar cliente HTTP.
// Servidor destino: 192.168.1.15
// Puerto TCP: 82
// Opciones: Utilizar dirección IP.
// Timeout: Usar valor por defecto.

HttpSendInit("192.168.1.15", 82, HTTP_SEND_OPT_USE_IP)
```

Ejemplo 2:

```
// Inicializar cliente HTTP.
// Servidor destino: ejemplo.com
// Puerto TCP: Usar valor por defecto.
// Opciones: Utilizar valores por defecto de librería.
// Timeout: Usar valor por defecto.

HttpSendInit("ejemplo.com")
```

Ejemplo 3:

```
// Inicializar cliente HTTP.
// Servidor destino: ejemplo.com
// Puerto TCP: 82
// Opciones: Utilizar valores por defecto de librería.
// Timeout: Usar valor por defecto.

HttpSendInit("ejemplo.com", 82)
```

Ejemplo 4:

```
// Inicializar cliente HTTP.
// Servidor destino: ejemplo.com
// Puerto TCP: 82
// Opciones: Utilizar valores por defecto de librería.
// Timeout: Usar 60 segundos.

HttpSendInit("ejemplo.com", 82, HTTP_SEND_OPT_DEFAULT, 60)
```

Ejemplo 5:

```
// Inicializar cliente HTTP.
// Servidor destino: ejemplo.com
// Puerto TCP: 80
// Opciones: Obtener respuesta del servidor.
// Timeout: 60 segundos.

HttpSendInit("ejemplo.com", 80, HTTP_SEND_OPT_GET_RESPONSE, 60)
```

Ejemplo 6:

```
// Inicializar cliente HTTP.
// Servidor destino: 192.168.1.15
// Puerto TCP: 82
// Opciones: Utilizar dirección IP y obtener respuesta del servidor.
// Timeout: 60 segundos.

HttpSendInit("192.168.1.15", 80, HTTP_SEND_OPT_USE_IP|HTTP_SEND_OPT_GET_RESPONSE, 60)
```

Ejemplo 7:

```
// Inicializar cliente HTTP.
// Servidor destino: 192.168.1.15
// Puerto TCP: 82
// Opciones: Utilizar dirección IP.
// Timeout: 60 segundos.

HttpSendInit("192.168.1.15", 80, HTTP_SEND_OPT_USE_IP, 60)
```

HttpSendClose(): Cierra la conexión actual al servidor web.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, cliente no está conectado.
-2	S	Error, no hay conexión asignada.
-4	S	Error, librería no fue inicializada.
Notas		**Descripción**
1		No es necesario utilizarla a menos que se quiera abortar inmediatamente la conexión.

Ejemplo:

```
PlcMain()
{
   // . . .

   // Forzar a desconectar del servidor web.
   HttpSendClose()

   // . . .
}
```

HttpSendGetLibStatus(): Retorna el estado de la librería.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
>0	S	Mayor a 0, código de estado informativo. Ver constantes **Tabla 1**, pág. 71.
0	S	Igual a 0, librería lista/ok. Ver constantes **Tabla 1**, pág. 71.
<0	S	Error en librería. Ver constantes **Tabla 1**, pág. 71.
Notas		**Descripción**
1		El estado de la librería indica una condición particular del cliente web. Por ejemplo si una transacción retorno con errores, no se pudo conectar a un servidor web o la transacción terminó sin novedades/exitosa. Pero no indica si un servidor nos responde con un código de error propio del protocolo, por ejemplo “la pagina no existe”.
2		La función **HttpSendGetResponseCode()** retorna el código de estado del servidor luego de una transacción.

Ejemplo 1:

El siguiente código comprueba si el cliente no esta ocupado en una transacción HTTP pendiente. Esto puede ser útil para comenzar nuevamente una transmisión luego que termine.

```
// Comprobar si el cliente web tiene una transacción HTTP pendiente.
if(HttpSendGetLibStatus() != HTTP_SEND_STAT_TRY_TO_SEND)
{
      // Cliente listo para comenzar una nueva transacción.
}
```

Ejemplo 2:

El siguiente código comprueba si la transacción fue exitosa del lado del cliente web una vez que fue completada.

```
new SendStat

// Obtener código de estado de librería.
SendStat = HttpSendGetLibStatus()

// Comprobar si el cliente web procesó la transacción exitosamente.
if(SendStat != HTTP_SEND_STAT_OK)
{
      // No, Error en transacción.
}
else
{
      // Si, Sin errores en transacción.
}
```

Tabla 1: Códigos de estados retornado por HttpSendGetLibStaus()

Nombre	Valor	Descripción
HTTP_SEND_STAT_TRY_TO_SEND	127	Requerimiento en proceso de envío.
HTTP_SEND_STAT_LIB_NOT_INIT	126	La librería no fue inicializada aun.
HTTP_SEND_STAT_INITIALIZED	124	La librería fue inicializada.
HTTP_SEND_STAT_CLOSED	123	La conexión fue cerrada por el cliente.
HTTP_SEND_STAT_SEND_OK	122	Datos transmitidos con éxito.
HTTP_SEND_STAT_OK	0	Librería lista / Transacción OK.
HTTP_SEND_STAT_ERR_ABORTED	-1	Error, conexión abortada.
HTTP_SEND_STAT_ERR_CONN_TO	-2	Error, timeout en conexión. Usualmente cuando el servidor no existe.
HTTP_SEND_STAT_ERR_CLOSED	-3	Error, conexión cerrada por el host.
HTTP_SEND_STAT_ERR_TO	-4	Error, timeout en transacción. Especificado por función de inicialización.
HTTP_SEND_STAT_ERR_ALLOC	-5	Error, la conexión no fue asignada.
HTTP_SEND_STAT_ERR_FUNC	-6	Error, ver código de retorno de función.
HTTP_SEND_STAT_ERR_RESOLV	-7	Error, no se puede resolver el nombre del host.

Ver otro ejemplo similar para comprobar errores en página 41, sección 6.11.

HttpSendCheckValidTransaction(): Comprueba si la ultima transacción fue válida. Dos condiciones que deben cumplirse al mismo tiempo en una transacción HTTP normal para que sea totalmente exitosa:

1. El código de estado de la librería HTTP SEND es “0”
2. El código de respuesta del servidor HTTP es “200”.

Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
1	S	Última transacción fue valida.
0	S	Última transacción fue invalida.
Notas		**Descripción**
1		Se utiliza generalmente en el evento **@OnHttpSendCompleted()** al finalizar la transacción HTTP.
2		En caso de error, si desea analizar en profundidad, puede obtener los códigos completos retornados por las funciones **HttpSendGetLibStatus()** y **HttpSendGetResponseCode()**, para así determinar la causa del error.
3		Tenga en cuenta que un servidor puede retornar un código 2XX y considerarse valida la transacción, pero esta función retornará 0 (invalida) porque espera el código 200. Consulte protocolo HTTP.

Ejemplo 1:

```
@OnHttpSendCompleted()
{
   // Actualizar flag de error.
   // Si hay error en transaccion, hacer SendError = 1.
   SendError = !HttpSendCheckValidTransaction()

   // Comprobar si no hay errores.
   if(SendError == 0)
   {
      // Incrementar variables.
      Value1 += 1
      Value2 += 2
      Value3 += 3
      Value4 += 4
   }

   // Habilitar flag para iniciar próxima transmisión.
   Send = 1
}
```

Ver otro ejemplo similar para comprobar errores en página 41, sección 6.11.

8.2 Funciones Para Eventos

HttpSendSetCompletedEvent(): Habilita el evento **@OnHttpSendCompleted().** El evento **OnHttpSendCompleted** será llamado al finalizar una transacción HTTP.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo habilitarse.
Notas		**Descripción**
1		El evento se ejecuta al finalizar la transacción HTTP, con o sin errores.
2		El evento **OnHttpSendCompleted** es un mecanismo muy útil para procesar el fin de una transacción, comprobar errores, obtener datos de respuesta web, etc.

Ejemplo:

```
PlcMain()
{
   // Activar evento OnHttpSendCompleted()
   HttpSendSetCompletedEvent()

   // . . .
}

@OnHttpSendCompleted()
{
   // Procesar Evento.
}
```

HttpSendClrCompletedEvent(): Deshabilita el evento **@OnHttpSendCompleted()**.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo deshabilitarse.
Notas		**Descripción**
-		-

Ejemplo:

```
   // Desactivar evento OnHttpSendCompleted()
   HttpSendClrCompletedEvent()
```

8.3 Funciones Para Realizar Peticiones GET y POST

HttpSendGet(Path[ ], …): Envía una petición GET al Servidor Web.		
Argumentos	**Tipo**	**Descripción**
Path[ ]	E	Cadena con dirección del recurso web a solicitar. Debe comenzar con el carácter "/" del directorio raíz. Puede contener formato (ver pág. 77). Habitualmente se transmite junto a un **Query String** (ver sección 4.2 en pág. 5) para la transmisión de datos. Recordar que el Query String debe estar codificado en URL-Encoded. Longitud máxima: 310 caracteres.
…	E	Lista de argumentos variables si se especifica formato en la cadena **Path**.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa. La transacción fue aceptada por el PLC. Pero no implica que haya finalizado.
-1	S	Error, librería ocupada (usualmente una transacción pendiente)
-2	S	Error, no se puede asignar conexión.
-4	S	Error, librería no inicializada.
-10	S	Error, error de argumento o dirección de cadena.
Notas		**Descripción**
1		Puede utilizar el evento **OnHttpSendCompleted()** para procesar el fin de la transacción.
2		Recuerde inicializar **HttpSendInit()** con la opción HTTP_SEND_OPT_USE_GET si desea obtener el cuerpo del mensaje de respuesta del servidor web.

Ejemplo 1:

Pedir pagina "index.html" al servidor web.

```
if(HttpSendGet("/index.html") == 0)
{
    // Transmisión en curso, borrar flag.
    Send = 0
}
```

Ejemplo 2:

Enviar 4 campos con valores de Value1, Value2, Value3 y Value4 al servidor web.

```
if(HttpSendGet("/data/log.php?field1=%d&field2=%d&field3=%d&field4=%d",
                Value1, Value2, Value3, Value4) == 0)
{
   // Transmisión en curso, borrar flag.
   Send = 0
}
```

Notar que el Query String enviado fue el siguiente, luego del carácter “?”:

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Ver otro ejemplo en página 29, sección 6.6.

HttpSendPost(Path[ ], Query[ ], ...): Envía una petición POST al Servidor Web.		
Argumentos	**Tipo**	**Descripción**
Path[ ]	E	Cadena con dirección del recurso web a solicitar. Debe comenzar con el carácter "/" del directorio raíz. Longitud máxima: 60 caracteres.
Query[ ]		Cadena con el **Query String** (ver sección 4.2 en pág. 5) para la transmisión de datos. Puede contener formato (ver pág. 77). Recordar que el Query String debe estar codificado en URL-Encoded. Longitud máxima: 250 caracteres.
…	E	Lista de argumentos variables si se especifica formato en la cadena **Query**.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa. La transacción fue aceptada por el PLC. Pero no implica que haya finalizado.
-1	S	Error, librería ocupada (usualmente una transacción pendiente)
-2	S	Error, no se puede asignar conexión.
-4	S	Error, librería no inicializada.
-10	S	Error, error de argumento o dirección de cadena.
Notas		**Descripción**
1		Puede utilizar el evento **OnHttpSendCompleted()** para procesar el fin de la transacción.
2		Recuerde inicializar **HttpSendInit()** con la opción HTTP_SEND_OPT_USE_GET si desea obtener el cuerpo del mensaje de respuesta del servidor web.

Ejemplo 1:

Enviar 4 campos con valores de Value1, Value2, Value3 y Value4 al servidor web.

```
if(HttpSendPost("/data/log.php","field1=%d&field2=%d&field3=%d&field4=%d",
                Value1, Value2, Value3, Value4) == 0)
{
   // Transmisión en curso, borrar flag.
   Send = 0
}
```

Notar que el **Query String** se especifica por separado al **Path**, y fue el siguiente:

```
field1=%d&field2=%d&field3=%d&field4=%d
```

Ver otro ejemplo en página 37, sección 6.10.

8.3.1 Formato Para las Cadenas de HttpSendGet() y HttpSendPost()

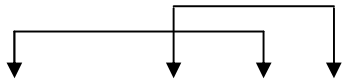
Las cadenas que contienen el **Query String** en las funciones **HttpSendGet()** y **HttpSendPost()** pueden contener formato, esto es, se pueden usar códigos de formato que luego serán reemplazados por los valores de variables antes de ser transmitida al servidor web. Esto le permite enviar cadenas cuyo contenido varía dinámicamente. Su utilidad es enorme. Por ello, haremos énfasis en explicar su funcionamiento.

Su implementación es una versión simplificada de la clásica función **printf()** del lenguaje C, por lo tanto no incluye todas sus características.

La cadena que admite formato en la función **HttpSendGet()** es **Path** y en la función **HttpSendPost()** es **Query** . El último argumento, denominado "…", significa que puede incluir un número de argumentos variable. Con argumentos variables, nos referimos a que el número de argumentos no está definido. De acuerdo al formato de la cadena a enviar, pueden o no existir argumentos.

Como las cadenas con formato transmiten el Query String, a continuación la usaremos en los ejemplos por comodidad:

Un ejemplo básico puede ser:



```
HttpSendXXX("field1=%f&field2=%d",  5.5, 9)
```

Imprimirá en la cadena el siguiente texto antes de enviarla:

```
field1=5.5&field2=9
```

Donde los códigos de formato, precedidos por un "%", son: "%f" y "%d". Indican que los argumentos que deben pasarse en la lista de argumentos variables, son un numero flotante (Float) y el otro un entero con signo, en ese orden. También pueden pasarse los valores como variables.

Códigos Formato:

Nombre	Tipo	Descripción
%d	Entero	Imprime un entero con signo.
%u	Entero	Imprime un entero sin signo.
%x	Entero	Imprime un entero con notación hexadecimal.
%X	Entero	Imprime un entero con notación hexadecimal en mayúscula.
%f	Flotante	Imprime un número decimal o punto flotante.
%s	Cadena	Imprime una cadena de caracteres.
%c	Carácter	Imprime un solo carácter.

También es posible especificar alineación, relleno y precisión (en un numero punto flotante).

Alineación y Relleno:

Se antepone un numero entre el "%" y el código. El número especifica la cantidad de espacios que deberán ser rellenados antes de imprimir la variable. Si el digito es negativo "-", la alineación será hacia la izquierda.

Si antes del digito de alineación, se antepone el numero 0, en vez de rellenar con espacios, la alineación será rellenada con ceros.

Precisión:

En un número flotante, es posible especificar la cantidad de dígitos decimales a imprimir en la cadena (por defecto es dos). Por ejemplo, "%.000f", imprimimos un numero decimal con tres dígitos luego del punto solamente

Ejemplos de formatos:

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%d	3	3							
%4d	3				3				
%04d	3	0	0	0	3				
%-4d	3	3							
%04d	-3	-	0	0	3				
%4d	-3			-	3				
%03.000f	5.12345	0	5	.	1	2	3		
%f	5.12345	5	.	1	2				
%.4f	5.12345	5	.	1	2	3	4	5	
%.02f	3.1	3	.	1	0				

Ejemplo	Argumento	Salida en Display (cada columna es un carácter)							
%x	255	f	f						
%X	255	F	F						
%s	"HOLA"	H	O	L	A				
%c	66	B							
%c	'B'	B							
%s	0	(	n	u	l	l	)		

Ejemplos prácticos:

```
// Imprimir cadena y número.
HttpSendXXX("Nombre=%s&Edad=%d",  "Rodion", 23)
```

```
Nombre=Rodion&Edad=23
```

```
// Imprimir decimal.
HttpSendXXX("Tanque=%f", 60.37)
```

```
Tanque=60.37
```

```
// Imprimir con relleno.
HttpSendXXX("Cuenta=%05d", 112)
```

```
Cuenta=00112
```

```
// Limitar precisión.
HttpSendXXX("Tension=%.3f", 11.34556)
```

```
Tension=11.345
```

8.4 Funciones Para Obtener Información y Datos

HttpSendGetResponseCode(): Obtiene el código de estado de respuesta devuelto por un servidor luego de una transacción HTTP exitosa tipo GET o POST.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Código invalido. Comprobar conexión.
-	S	Código de estado de respuesta de HTTP recibido. Valor de 3 dígitos. Algunos ejemplos: • 200 : Respuesta estándar para peticiones correctas. • 301 : Movido permanentemente. • 400 : Error en sintaxis. • 404 : Recurso no encontrado. • 1xx : Respuestas informativas • 2xx : Peticiones correctas • 3xx : Redirecciones • 4xx : Errores del cliente • 5xx : Errores de servidor
Notas		**Descripción**
1		El código de respuesta de un servidor permite determinar si el servidor aceptó o no nuestra petición, y cuál es la posible causa del error en caso que no la acepte.
2		No debe confundir el código de respuesta de un servidor HTTP con el código de estado devuelto por la función **HttpSendGetLibStatus()**, el cual es un código interno de la librería. Por ejemplo, podemos tener una transacción exitosa (la conexión se logró con el servidor y obtuvimos respuesta) pero el servidor devolvió un código de error 404 (pagina no encontrada). Por ello, en algunas situaciones, es útil conocer el código de respuesta de un servidor web.
3		El código de respuesta de un servidor web es un número entero que está definido por el protocolo HTTP (RFC 2616). El código de estado devuelto para peticiones correctas es el 200.
4		No es necesario que compruebe este código para transacciones normales, pero si usted debe asegurarse una transmisión correcta, debería leer dicho código en cada respuesta del servidor.
5		Utilice la función **HttpSendCheckValidTransaction()** para una detección de errores simple.
6		El código 200 es el más importante, indica éxito en la transacción en ambas partes, cliente y servidor.

Ejemplo:

```
@OnHttpSendCompleted()
{
   // Obtener Código de Respuesa del Servidor HTTP
   ResponseCode =  HttpSendGetResponseCode()

   // Comprobar si petición fue exitosa.
   if(ResponseCode == 220)
   {
      // Incrementar variables.
      Value1 += 1
      Value2 += 2
      Value3 += 3
      Value4 += 4
   }

   // Habilitar flag para iniciar próxima transmisión.
   Send = 1
}
```

Ver otro ejemplo similar para comprobar errores en página 41, sección 6.11.

HttpSendGetBodyLength(): Obtiene la cantidad de bytes del cuerpo del mensaje recibido de un servidor luego de una transacción HTTP exitosa tipo GET o POST.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
>0	S	Cantidad de bytes recibidos no leídos desde última transacción HTTP.
0	S	No hay datos recibidos o no leídos desde última transacción HTTP.
-1	S	Error, transacción invalida, no hay datos disponibles.
-4	S	Error, librería no inicializada.
Notas		**Descripción**
1		El cuerpo del mensaje de una respuesta HTTP es la página web o recurso solicitado al servidor.
2		Al enviar una petición GET o POST, el servidor puede responder con un mensaje o contenido (pagina web). Si la función **HttpSendInit()** se configuró con la opción "**HTTP_SEND_OPT_GET_RESPONSE**", el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño) para luego poder recuperarlo con la función **HttpSendGetBodyData().** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.
3		Si la longitud del mensaje recibido por el PLC excede su buffer interno, el resto de los datos recibidos son descartados, almacenándose solo la capacidad del buffer. Esta función devolverá la cantidad de bytes almacenados no leídos.
4		La longitud máxima del buffer interno del PLC es 308 bytes, a menos que se indique lo contrario.

Ejemplo:

```
@OnHttpSendCompleted()
{
   new RxLength
   new RxData[40]

   // Obtener numero de bytes recibidos.
   RxLength = HttpSendGetBodyLength()

   // Si hay datos disponibles sin leer, leer y almacenar.
   if(RxLength > 0)
   {
      // Leer 40 bytes y almacenar en array RxData[].
      HttpSendGetBodyData(RxData, 0, 40, false)
   }
}
```

Ver otro ejemplo en página 42, sección 6.12.

HttpSendGetBodyData (Data[ ], Index, Max, Pack): Obtiene el cuerpo del mensaje recibido en la última transacción HTTP tipo GET o POST.		
Argumentos	**Tipo**	**Descripción**
Data[ ]	S	Array donde se copiaran los bytes leídos del buffer. Ver nota 1.
Index	E	Índice del array Data[ ] donde se empezara a copiar los bytes. Ver nota 1.
Max	E	Numero de bytes a copiar desde el buffer de recepción (máximo 308). Si este valor es cero, se copiaran todos los bytes recibidos. Ver nota 1.
Pack	E	Si este valor es 1 (true), 4 bytes serán empaquetados (packed) en una celda (cell) del array. Si es 0 (false), los bytes no serán empaquetados (unpacked), y se copiara un byte por celda (cell) del array. Ver nota 2.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error. El buffer interno está vacío, no hay datos para copiar.
-2	S	Error. El numero de bytes a copiar (Max) es invalido.
-4	S	Error. Dirección del array “Data[ ]” es incorrecta.
Notas		**Descripción**
1		Es importante que los argumentos Index y Max, sean acordes a las dimensiones del array Data[ ] de destino. Es responsabilidad del programador, verificar que el valor Index + Max no exceda el tamaño del array.
2		Cuando el argumento Pack = true, es posible almacenar 4 bytes en una celda (empaquetamiento), por lo tanto, se ahorra espacio en memoria RAM. Se recomienda utilizar el empaquetamiento cuando Max es mayor a 12. De esta forma, 12 bytes pueden entrar en 3 celdas. Si Pack = false, cada byte leído es copiado en una celda. Si por ejemplo, leemos 5 bytes, necesitaremos un array de 5 celdas. Si bien se desperdicia memoria, la velocidad de lectura será mayor. El formato de los bytes empaquetados se explica en manual **STX80XX-MP-PLC_AX_CX_DX**.
3		El cuerpo del mensaje de una respuesta HTTP es la página web o recurso solicitado al servidor.
4		Al enviar una petición GET o POST, el servidor puede responder con un mensaje o contenido (pagina web). Si la función **HttpSendInit()** se configuró con la opción “**HTTP_SEND_OPT_GET_RESPONSE**”, el PLC almacenará la respuesta del servidor web en un buffer interno (limitado en tamaño) para luego poder recuperarlo con la función **HttpSendGetBodyData().** Esto puede ser útil para procesar algún mensaje o comunicación bidireccional entre PLC y servidor web.
5		Si la longitud del mensaje recibido por el PLC excede su buffer interno, el resto de los datos recibidos son descartados, almacenándose solo la capacidad del buffer. Esta función devolverá la cantidad de bytes almacenados no leídos.
6		La longitud máxima del buffer interno del PLC es 308 bytes, a menos que se indique lo contrario.

Ejemplo:

```
@OnHttpSendCompleted()
{
   new RxLength
   new RxData[40]

   // Obtener numero de bytes recibidos.
   RxLength = HttpSendGetBodyLength()

   // Si hay datos disponibles sin leer, leer y almacenar.
   if(RxLength > 0)
   {
      // Leer 40 bytes y almacenar en array RxData[].
      HttpSendGetBodyData(RxData, 0, 40, false)

      // Activar/Desactivar DOUT1 si primer byte recibido es
      // igual al caracter "B".

      if(BodyData[0] == 'B')
      {
         DoutSetOn(DOUT1)
      }
      else
      {
            DoutSetOff(DOUT1)
      }
   }
}
```

Ver otro ejemplo en página 42, sección 6.12.

Consejo:

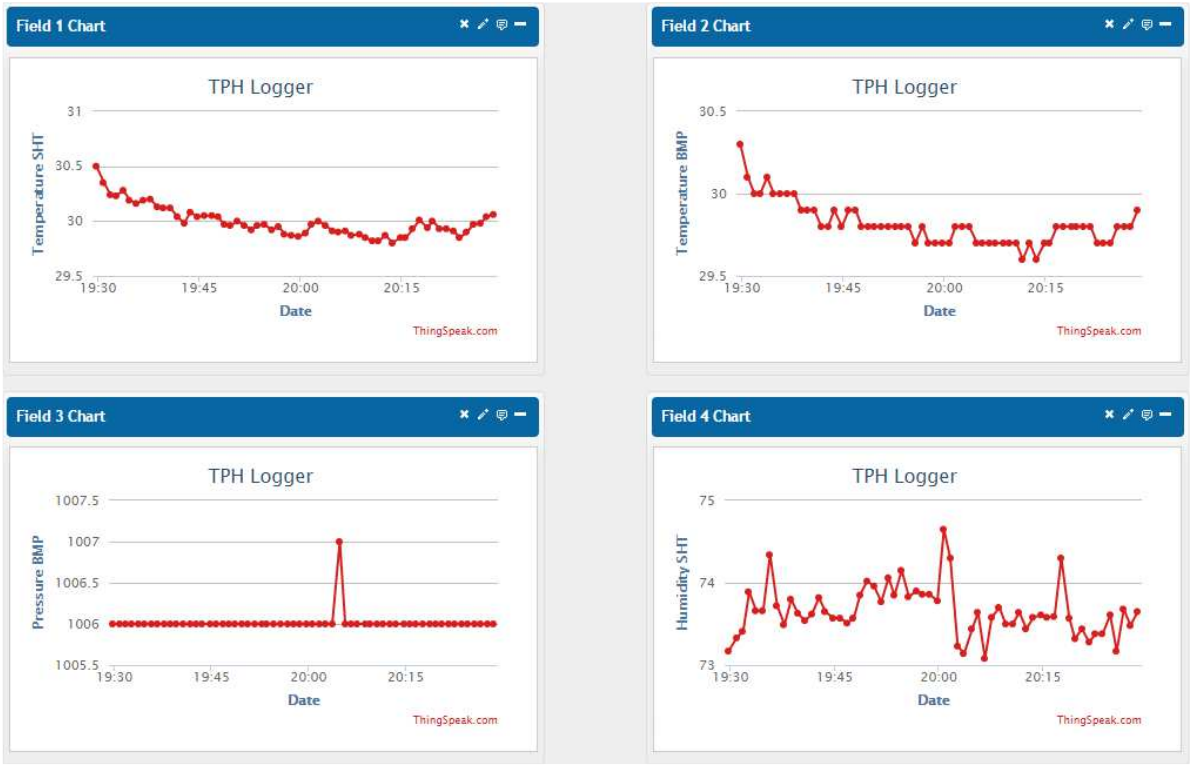
- Recibir el cuerpo del mensaje de una respuesta HTTP solo es necesario si desea obtener alguna información particular del servidor web, de lo contrario no necesita usar esta función.

9 Ejemplos Avanzados

Desde nuestra página web, específicamente en la sección de esta nota de aplicación:

www.slicetex.com/docs/an/an032

Puede descargar ejemplos para usar thingspeak.com (visualización de parámetros en gráficos en función del tiempo, muy útil para registrar temperaturas, humedad, y parámetros lentos) y ejemplos de script PHP para acceder a base de datos SQL.



Gráficos en Thingpeak.com que aceptan conexiones HTTP GET desde el PLC

10 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **IP**: Dirección Internet, conformada por cuatro octetos, por ejemplo 192.168.1.81.
- **HTTP:** Protocolo usado en la transferencia de páginas web.
- **HTML:** Lenguaje usado en las páginas web.
- **PHP:** Lenguaje de programación de lado servidor, útil para paginas dinámicas web.
- **Ethernet**: Red de computadoras, que generalmente se utilizan el protocolo de internet TCP/IP o UDP/IP.
- **Transacción**: Proceso de enviar un requerimiento y esperar respuesta de un Servidor Web.

11 Historial de Revisiones

Tabla 2: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
01 25/Oct/2016	1	1. Versión preliminar liberada.	Preliminar

12 Referencias

Ninguna.

13 Información Legal

13.1 Aviso de exención de responsabilidad

General: La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

El derecho a realizar cambios: Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

Idoneidad para el uso: Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

Aplicaciones: Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

Los valores límites o máximos: Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

Documento: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

14 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Para información y consultas, envíe un mail a: info@slicetex.com

Para soporte técnico ingrese a nuestro foro en: www.slicetex.com/foro

Slicetex Electronics
Córdoba, Argentina

15 Contenido

15.1 Índice de Tablas