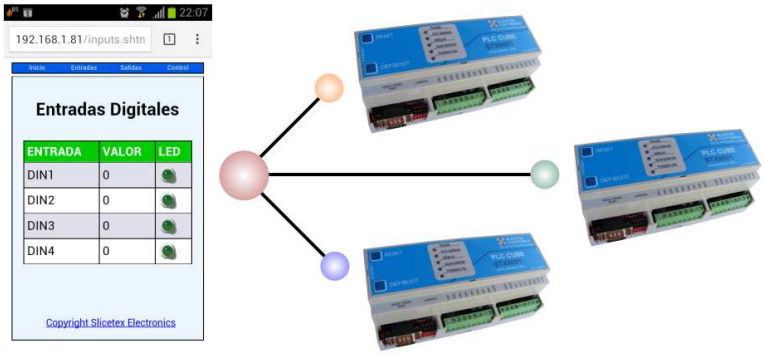
Slicetex Ladder Designer Studio

NOTA DE APLICACIÓN

AN030
Servidor Web del PLC

Autor: Ing. Boris Estudiez



Modelos Aplicables	AX, CX y DX

1 Descripción General

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

El PLC permite activar un Servidor Web, el cual mediante páginas Web creadas por el usuario muestra información y controles accesibles a través de cualquier navegador de internet.

La información en las páginas Web puede ser dinámica, es decir, es posible mostrar datos que varían en tiempo real, como valores de señales analógicas, valores digitales, etc.

A su vez, permite la recepción de datos por parte del usuario, permitiendo operaciones de control remoto.

Cada página Web puede ser libremente diseñada por el usuario en lenguaje HTML, haciendo de una alternativa visual fácil de utilizar y ampliamente accesible por cualquier dispositivo estándar remoto que tenga un navegador Web (como Chrome, Explorer, Firefox, Edge, etc), disponible en computadoras, dispositivos móviles, tablets, etc.

Las páginas web son cargadas al PLC como archivos independientes desde StxLadder, siendo posible establecer una contraseña para evitar accesos no autorizados.

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 **Servirdor 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 8. 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 34. 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 **Servidor Web** en el PLC.

Se recomienda estar familiarizado con los conceptos básicos del lenguaje HTML y con el mecanismo general de funcionamiento de los servidores Web.

Enlaces recomendados:

- http://www.w3schools.com/html: Información y ejemplos muy útiles para aprender HTML.

- http://www.w3schools.com/css: Información y ejemplos para aprender formato con CSS.

Software gratuito de edición Web (paginas HMTL, CSS, etc) recomendado:

- **Kompozer**: Editor HTML, http://www.kompozer.net/

4 Teoría de Funcionamiento

Los PLC de Slicetex Electronics permiten configurarse como **Servidor Web**, aceptando conexiones de clientes Web (navegadores). La comunicación se realiza en general por el puerto Ethernet disponible en el dispositivo, o alguna otra interfaz compatible con Internet.

Los navegadores (clientes) se comunican con el servidor Web utilizando el protocolo HTTP (Hypertext Transfer Protocol)

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

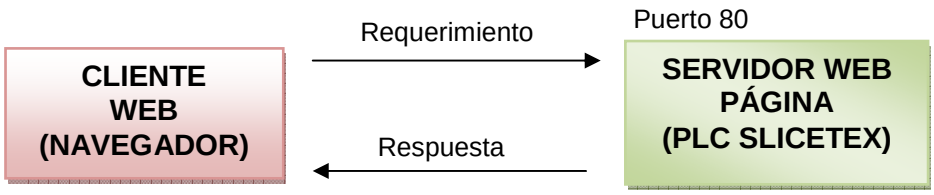


Fig. 1: Transacción HTTP.

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción HTTP para procesar un requerimiento de un cliente Web. Los pasos son los siguientes:

1. **Configuración**: El PLC debe configurar los parámetros para funcionar como servidor Web, por ejemplo numero de puerto, timeout de inactividad, páginas web, contraseña, etc.
2. **Requerimiento**: El cliente envía un requerimiento para obtener una página Web o enviar datos.
3. **Respuesta**: El servidor procesa el requerimiento del cliente, y devuelve una respuesta que depende de la pagina Web, ya sea si esta es estática o dinámica (con información variable). Si la página es dinámica, el servidor Web incluirá información actualizada en la respuesta que depende de los datos a mostrar (como valor de variables, parámetros, etc).
4. **Recepción:** El cliente recibe la respuesta, procesa la pagina HTML y la muestra en el navegador Web (con texto, imágenes, etc).
5. **Fin:** El cliente puede realizar otro requerimiento al servidor si es necesario, volviendo al punto 2.

4.1 Características del Servidor Web

En la tabla siguiente se listan algunas de las características del servidor Web.

Tabla 1: Características del Servidor Web

Característica	Valor	Descripción
Cantidad de Archivos	X	No hay un límite establecido, pero depende de la cantidad de memoria flash del dispositivo.
Memoria disponible para archivos	Espacio libre de memoria FLASH	Los archivos Web, se localizan en el espacio libre asignado para la memoria de programa del PLC mediante un sistema de archivos. Es decir, se carga el programa Ladder o Pawn primero, y en el espacio libre remanente se almacenan los archivos. Por lo tanto, el espacio para las páginas es relativo a la cantidad de memoria libre disponible del dispositivo. Consultar hoja de datos del dispositivo. El entorno StxLadder informa la cantidad de memoria utilizada.
Número máximo de conexiones en simultáneo	4	Valor mínimo, puede variar en diferentes dispositivos.
Extensiones de archivos admitidas		html, shtml, js, txt, gif, png, jpg, css, ico, xml (ver detalles en página 5).
Tamaño máximo de nombre de archivo	20	Caracteres
Soporte de paginas dinámicas	Si	Mediante sistema de scripts en archivos HTML, comandos CGI. Eventos disponibles en PLC.
Soporte de envío de datos y control	Si	Mediante formularios web y método POST. Eventos disponibles en PLC.
Soporte protección de accesos	Si	Mediante usuario y contraseña.
Compresión de datos	Si	Ver más información en sección **Comprimir Archivos** de página 104.

4.2 Detalles de Extensiones Soportadas

En la tabla siguiente se listan las extensiones soportadas por servidor Web y su descripción. El uso correcto de los mismos, le permitirá optimizar el rendimiento del servidor.

Tabla 2: Extensiones de archivos soportadas

Extensión	Ejemplo	Descripción
html	info.html	Archivo con código HTML **estático**. El contenido de la página se muestra tal cual como fue diseñada.
shtml	index.shtml	Archivo con código HTML **dinámico** mediante script. El contenido de la página puede variar de acuerdo a los datos procesados por el PLC. Este tipo de página permiten la inclusión de comandos que son ejecutados por el PLC al cargarse la pagina y pueden devolver información imprimible, como texto, valores, imágenes, etc. La página es interpretada por el PLC antes de enviarse al navegador, por lo tanto su carga es más lenta que las páginas estáticas.
txt	info.txt	Archivo de texto plano.
gif	grafico.gif	Archivo de imagen GIF. Limitado a 256 colores, es un formato de archivo con compresión sin pérdida de información. Es una buena elección para almacenar líneas, dibujos, texto, gráficos pequeños e iconos en un tamaño pequeño. Admite transparencias. Ampliamente soportado.
png	grafico.png	Archivo de imagen PNG. Es un formato de archivo con compresión sin pérdida de información. Es una buena elección para almacenar líneas, dibujos, texto, gráficos pequeños, fondos e iconos en un tamaño pequeño. No está limitado en colores. No se recomienda para fotografías. Admite transparencias.
jpg	foto.jpg	Archivo de imagen JPG. Es un formato de archivo con compresión y **pérdida de información** (en detrimento de la calidad). Es una buena elección para almacenar fotografías o imágenes realistas. Para gráficos artísticos como líneas o iconos, es mejor utilizar GIF o PNG, ya que no pierden información. Sin embargo, es mejor para imágenes con muchos colores, ya que GIF está limitado y PNG no es eficiente en memoria para imágenes realistas. El formato JPG no admite transparencias.
ico	favicon.ico	Archivo de imagen de icono ICO. Introducido por Microsoft. Se usa principalmente en el servidor para establecer el icono de la pagina Web mostrado en la pestaña del navegador (16x16 pixeles). Es de uso opcional y no se recomienda para reemplazar los archivos GIF, PNG y JPG.
css	style.css	Archivo de estilos. Si está definido, permite definir y crear un documento de presentación de estilos para el archivo HTML/SHTML. Es decir, el navegador formateará las páginas presentadas de acuerdo a la información de este archivo.

xml	config.xml	Archivo XML, con lenguaje de marcas que define un set de reglas de codificación y formato, que puede ser leído por humanos como por maquinas. En el servidor está pensado para aplicaciones o clientes remotos específicos que desean obtener información del servidor (como datos, valores, configuración, etc).
js	script.js	Archivo JavaScript, el cual permite programar en un lenguaje script interpretado por el navegador el comportamiento de la página de forma dinámica. El script se ejecuta desde el lado cliente. Muy útil para crear librerías graficas o realizar lógicas en paginas web.

4.3 Archivos Web Obligatorios

Los siguientes archivos son obligatorios en el servidor Web. El entorno **StxLadder** comprueba la existencia de los mismos antes de cargarlos en el PLC.

Tabla 3: Archivos obligatorios

Archivo	Descripción
index.html o index.shml	Página principal del servidor, es el punto de entrada cuando se accede a la dirección del mismo. Puede ser con extensión estática o dinámica.
404.html o 404.shtml	Página de recursos no encontrados. Cuando un archivo requerido no es encontrado en el servidor, esta página será mostrada. Puede ser con extensión estática o dinámica.
401.html	Página que se muestra cuando el servidor está protegido por contraseña y el acceso fue denegado.
413.html	Página que se muestra cuando se envía un formulario web con método POST con una longitud de datos que supera el límite del servidor.

4.4 Consideraciones para Optimizar el Funcionamiento

Algunas ideas para mejorar el desempeño del servidor Web y optimizar recursos:

- Comprenda el lenguaje HTML, evite usar diseñadores de pagina Web sofisticados ya que generan páginas con gran cantidad de información y por lo tanto ocupan más memoria.
- Se recomienda utilizar nombres de archivos cortos, y también limitar la cantidad de los mismos (no es obligatorio, pero mejora el desempeño).
- Utilizar páginas dinámicas solo para mostrar información variable.
- Es posible mediante comandos CGI reutilizar código HTML. Esto permite ahorrar espacio en memoria, agrupando código común a varias páginas Web (como menús, cabeceras, etc).
- Usar imágenes de tamaño reducido, limitando la resolución de las mismas con un programa de edición gráfico. Solo utilizarlas cuando sea necesario, ya que en general ocupan mucho espacio. Se recomienda usar una extensión adecuada como se detalla en **Tabla 2: Extensiones de archivos soportadas** de la página 5.

- Para reducir memoria utilizada por archivos web, no incluya comentarios, tabulaciones con espacios en blanco y código que no utilizará (especialmente en archivos CSS).
- Puede ahorrar memoria comprimiendo los archivos con la opción "**Comprimir archivos para reducir tamaño**" desde la configuración del servidor web en StxLadder. Ver más información en sección **Comprimir Archivos** de página 104.
- Utilice un archivo CSS para definir el formato de visualización de paginas HTML.
- En eventos de páginas web dinámicas, retorne lo antes posible desde el programa del PLC.

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 servidor Web rápidamente.

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

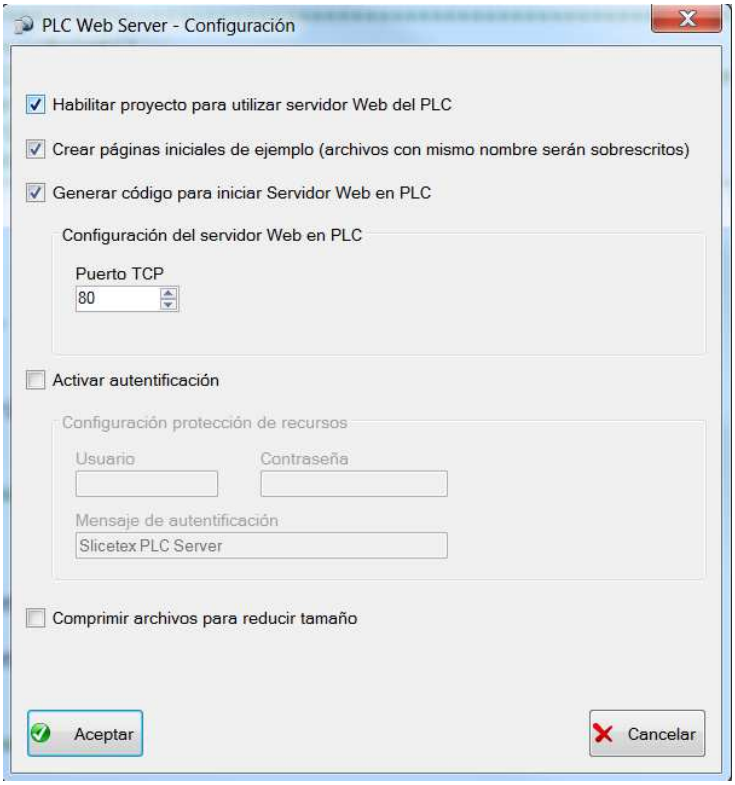
En esta sección expondremos un ejemplo básico en lenguaje Ladder, si está usando lenguaje Pawn, puede ir directamente al ejemplo Pawn en la página 34.

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

5.1 Crear el Proyecto

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

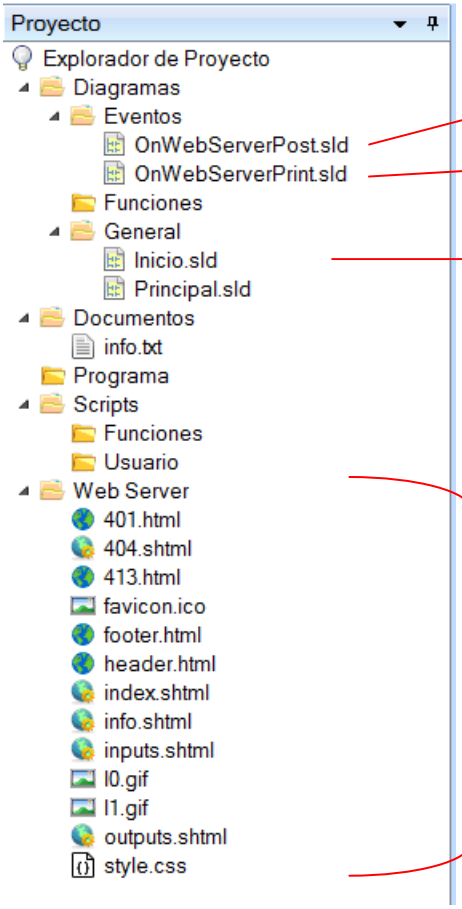
Active el uso del servidor Web en el proyecto seleccionando el menú "**Herramientas > PLC Web Server**". La siguiente ventana aparecerá:



Tilde las primeras tres opciones, para activar el servidor, crear páginas web iníciales y generar código Ladder en el proyecto, como muestra la figura superior.

Haga click en el botón "**Aceptar**" para efectuar los cambios.

Si el Servidor Web pudo configurarse con éxito en el proyecto, los siguientes archivos se agregaron al **Explorador del Proyecto**:



Evento **OnWebServerPost**. Procesa formularios Web.

Evento **OnWebServerPrint**. Imprime cadenas de texto en páginas Web dinámicas.

Diagrama Inicio: En network **N001** se agregan componentes para configurar el Servidor Web en el PLC.

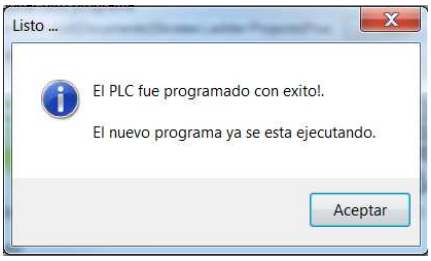
Archivos Web de ejemplo.
Este es el sistema de archivos accesibles a través de un navegador Web que se conecte al PLC.
Los archivos aquí depositados luego serán copiados a la memoria libre del PLC.

Notar como hay múltiples archivos con diferentes extensiones, SHTML, HMTL, GIF, CSS, etc.

Cuando dichos archivos son accedidos por un navegador Web, pueden disparar ciertos eventos en el PLC, y de esta forma interactuar con el código del proyecto.

Ahora que ya generó automáticamente los archivos básicos necesarios para utilizar el servidor Web en el PLC, puede compilar y transferirlos al PLC.

Entonces, conecte el PLC a la red Ethernet y seleccione el menú "**PLC > Cargar rápidamente**" en la barra de menús en **StxLadder**. El siguiente cartel se mostrará si el proyecto fue transferido al PLC:



5.2 Acceda a la Página Web del PLC

El próximo paso para probar el servidor Web del PLC, es accederlo con su navegador Web favorito (como Chrome, Explorer, Firefox, Edge, etc) desde la PC o desde una tablet / smartphone.

Para acceder con el navegador Web, debe colocar la dirección IP del PLC en la barra de direcciones del navegador. Por ejemplo, si el PLC se encuentra en la IP **192.168.1.81** de su red LAN, coloque en el navegador la siguiente dirección:

http://192.168.1.81

Si su red LAN está configurada correctamente y el proyecto fue cargado en el PLC, el navegador debe mostrar la siguiente página Web de inicio:

Si puede ver la pagina Web, ¡ya configuró con éxito el servidor Web del PLC!.

La pagina principal del PLC, corresponde al archivo **index.html** o **index.shtml** (de acuerdo a cual este cargado) y se carga por defecto al ingresar la dirección del PLC en el navegador.

Puede navegar por el servidor Web haciendo click en las páginas disponibles en el menú de la parte superior de la página Principal.

Si hace click en "**Entradas**" se mostrará la siguiente página que permite ver el estado de la entrada digital **DIN1** del PLC:

Note que la dirección en la barra de herramientas del navegador se transformó en:

http//192.168.1.81/inputs.shtml

Ya que se está accediendo al archivo "**inputs.shtml**".

Si polariza la entrada "**DIN1**" del PLC, y luego refresca o recarga la pagina "**inputs.shtml**", la pagina debería reflejar el nuevo valor de **DIN1** de la siguiente manera:

Como se puede notar, ahora **VALOR=1** y el **LED=Activado**. Esto significa que usted está cargando una página con contenido dinámico, es decir, la información mostrada puede cambiar según las condiciones de un programa cargado en el PLC. Las páginas dinámicas tienen extensiones **.shtml** en el PLC.

Si hace click en "**Salidas**" se mostrará la siguiente página que permite controlar la salida digital **DOUT1** del PLC:

Note que la dirección en la barra de herramientas del navegador se transformó en:

http//192.168.1.81/outputs.shtml

Ya que se está accediendo al archivo "**outputs.shtml**".

Esta página permite enviar datos desde el navegador Web (cliente) al PLC (servidor Web). El mecanismo que se utiliza es el método POST, que veremos más adelante.

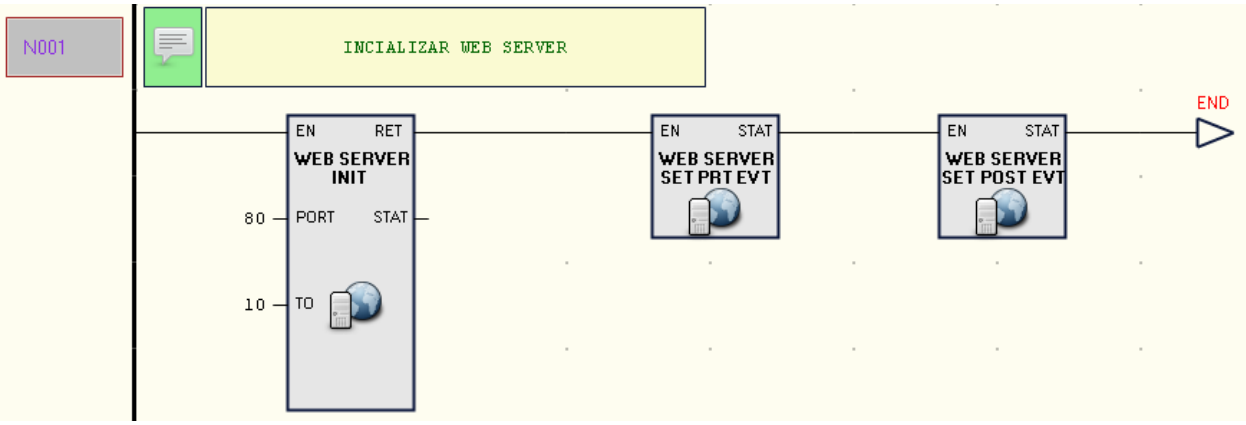
Básicamente consiste en un formulario Web (que puede contener strings, números, o elementos de control como botones) y los valores de los mismos se envían al PLC mediante campos delimitados.

En este caso, si tocamos en "**ON**" y luego damos click en el botón "**Actualizar**" de la página, podremos activar la salida digital DOUT1 del PLC.

Esta página también es dinámica porque en el campo VALOR se selecciona OFF / ON automáticamente de acuerdo al valor de la salida **DOUT1** cuando se carga la pagina. Esto se hace por código.

5.3 Configuración del Servidor Web

Para utilizar el servidor Web en el PLC, el mismo debe ser activado y configurado. Si abrimos el diagrama "**Inicio.sld**" podremos ver en la network **N001** el siguiente código:



El componente "**WEB SERVER INIT**" activa el servidor Web, especificando el puerto "PORT" donde se escucharán las conexiones TCP por defecto cuando un cliente se conecte. En este caso se eligió el puerto **80** que es el estándar. Más información del componente en pagina 62.

Si selecciona otro puerto, por ejemplo el numero **88** (ya sea por seguridad o para diferenciar de otros PLC conectados en una misma red), para acceder con un navegador deberá escribir el numero de puerto separado por ":" en la barra de direcciones, por ejemplo: http://192.168.1.81:88

El puerto "**TO**" del componente especifica el numero de segundos que un cliente TCP puede permanecer conectado al servidor de forma inactiva (sin enviar datos) antes de ser desconectado. Esto no tiene que ver con interactuar con la pagina Web desde el navegador, sino con el funcionamiento interno del navegador Web que utilice, por ejemplo, si el mismo se conecta al servidor pero no envía datos o peticiones, es un error, y el servidor del PLC lo desconectará pasado un tiempo para que no consuma recursos. En general 10 segundos es un tiempo adecuado.

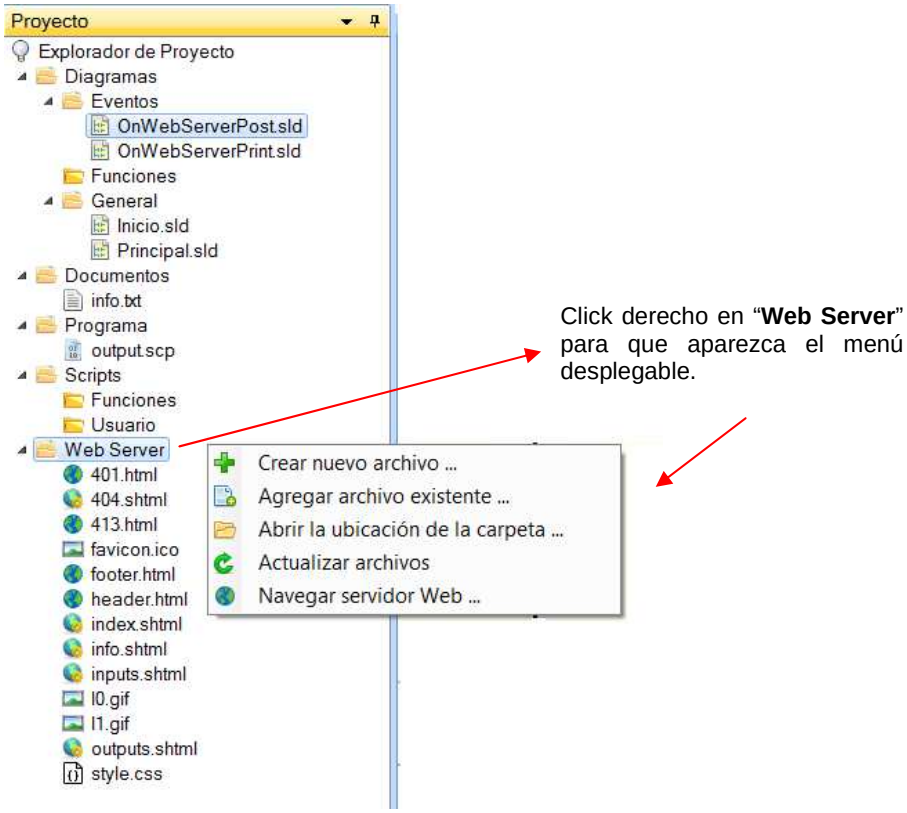
El componente "**WEB SERVER PRT EVT**", activa el evento **OnWebServerPrint**, el cual sirve para imprimir desde el PLC texto en las páginas Web dinámicas (archivos con extensión **.shtml**) cuando el servidor lo requiere.
Por ejemplo, si el usuario cargó desde el navegador la página **inputs.shml** que muestra el estado de la entrada digital **DIN1**, el servidor le pedirá al PLC que devuelva el estado de la entrada en tiempo real. Esto lo hace generando el evento **OnWebServerPrint**, el cual provoca que el diagrama **OnWebServerPrint.sld** se ejecute en el PLC, y en el mismo se procesa el requerimiento para devolver una cadena de texto con el estado de la entrada **DIN1**.

El componente "**WEB SERVER POST EVT**", activa el evento **OnWebServerPost**, el cual sirve para procesar desde el PLC formularios Web, los cuales envían datos desde el navegador al PLC. Son muy útiles para realizar tareas de control o enviar parámetros al PLC.
Por ejemplo, cuando el navegador accede a la página **outputs.html** y se hace click en el botón "**Actualizar**" para conmutar el estado de la salida **DOUT1**, el navegador envía el dato con el nuevo valor de **DOUT1** al servidor del PLC y el mismo genera el evento **OnWebServerPost.** Dicho evento dispara la ejecución del diagrama **OnWebServerPrint.sld**, que permite procesar el dato recibido y controlar la salida **DOUT1**.

5.4 Creando Archivos y Páginas Web

Desde el entorno **StxLadder** es muy fácil editar y cargar contenido Web en el servidor del PLC.

Para cargar contenido, desde el **Explorador de Proyecto** en **StxLadder** haga click derecho sobre la carpeta "**Web Server**". El siguiente menú desplegable aparecerá:



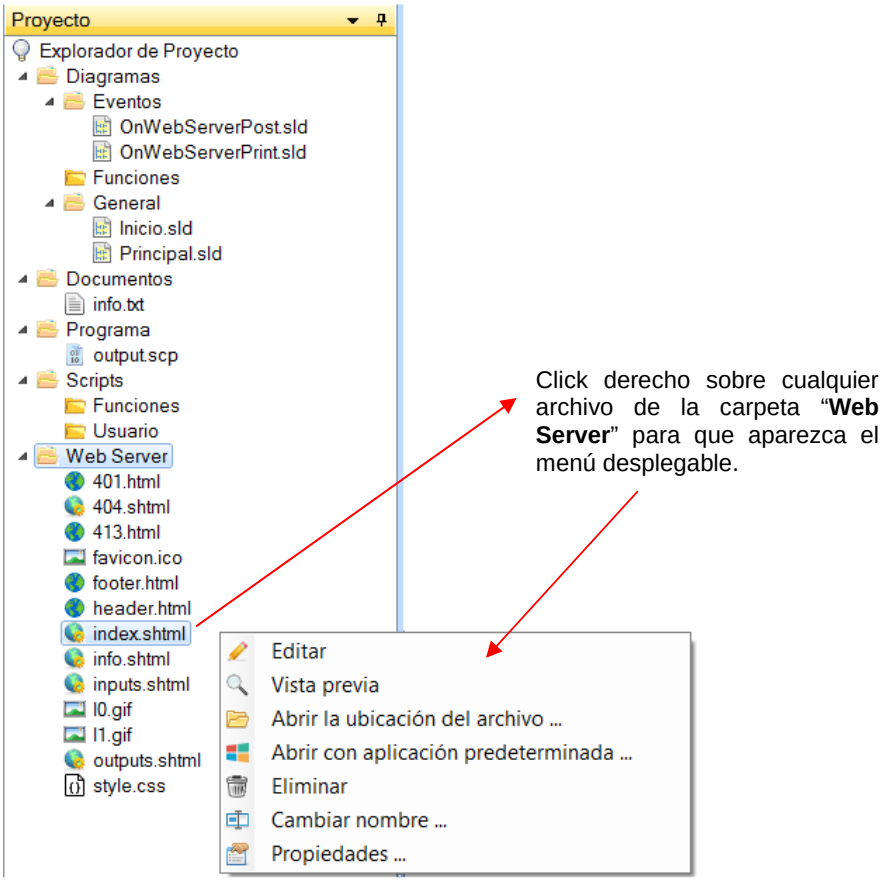
Este menú permite agregar nuevos archivos y archivos existentes al PLC. La forma más simple es utilizar "**Crear nuevo archivo …"** o "**Agregar nuevo archivo existente …**".

Si necesita acceder directamente a la carpeta contenedora de los archivos del PLC (ya sea para editarlos externamente, o agregar muchos al mismo tiempo, etc) seleccione "**Abrir la ubicación de la carpeta …**". Recuerde luego seleccionar "**Actualizar archivos**" si agrego o borro archivos externamente.

Si desea explorar el servidor Web desde el navegador de Web de StxLadder seleccione "**Navegar servidor Web …**" y por defecto se navega en la dirección IP del PLC configurada por el proyecto.

La lista de extensiones de archivos permitidos y obligatorios se puede consultar en la **Tabla 2** y **Tabla 3**, de las páginas 5 y 6.

Para editar contenido, desde el **Explorador de Proyecto** en **StxLadder** haga click derecho sobre cualquier archivo de la carpeta "**Web Server**". El siguiente menú desplegable aparecerá:



Las opciones del menú desplegable se explican por sí solas, las mismas permiten: editar, eliminar, cambiar nombre, etc al archivo seleccionado.

Si hace doble-click sobre el archivo, se ejecuta la primera opción por defecto, en este caso "**Editar**". Para un archivo con extensión .html, .shtml, .css, .txt o .xml, es posible editarlo desde **StxLadder**. Lo cual es muy recomendado y cómodo.

Alternativamente, puede usar su editor externo favorito, para ello seleccione "**Abrir la ubicación del archivo …**", y una carpeta de Windows se abrirá donde puede editar el archivo con su aplicación externa. No se recomienda editar un archivo desde StxLadder y al mismo tiempo desde una aplicación externa. En ese caso, realice dicha operación por separado.

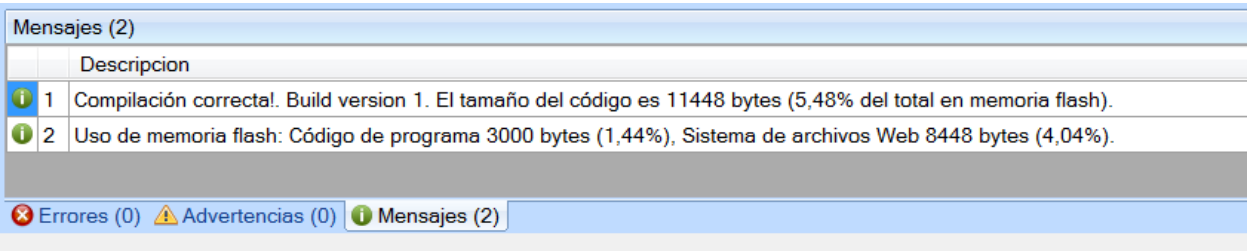
La opción "**Abrir con aplicación predeterminada …"** abre el archivo con la aplicación de Windows predeterminada para dicha extensión.

La opción "**Propiedades** …" es muy útil para consultar el tamaño en bytes de un archivo, de esta manera saber cuáles son mas "pesados" en caso que tenga poco espacio disponible en el PLC.

5.5 Memoria Utilizada

El PLC tiene una memoria abundante para código de programa, sin embargo, en general los archivos Web consumen mucha memoria si no se utilizan con cuidado. Para optimizar la memoria utilizada, siga los consejos en la sección "**Consideraciones para Optimizar el Funcionamiento"** de la página 6.

Para consultar la memoria utilizada por el los archivos Web en el PLC, debe compilar el proyecto desde el menú "**Proyecto > Compilar**" y observar el área de mensajes de **StxLadder**:



El mensaje numero 2, nos dice que en este proyecto **8448** bytes de memoria flash del PLC se utilizarán para almacenar los archivos Web, es decir un **4.04%** del total disponible para el modelo de PLC usado en el proyecto.

Esto nos deja un 95.96% de espacio para código de programa. Lo cual es bastante.

En este ejemplo, el programa solo ocupa 3000 bytes de memoria flash, el 1.44% del total para el modelo de PLC usado en el proyecto.

Tenga en cuenta que los archivos Web del servidor, se alojan en la memoria flash libre no ocupada por código de programa, esto quiere decir, que a mayor programa, menor espacio para pagina Web tenemos, y viceversa.

Puede visualizar desde el **Explorador de Proyecto** en StxLadder el tamaño de los archivos Web cargados, seleccionando la opción "**Propiedades …**" mediante click derecho sobre el archivo en cuestión. Esto puede ser útil para control el tamaño de los mismos a la hora de reducir espacio.

Es recomendable usar la menor cantidad de código HTML o imágenes en las páginas Web. Esto se logra conociendo la tecnología HTML y evitando usar diseñadores o editores Web complejos. En otras palabras, estudie el código HTML y edítelo manualmente con StxLadder.

5.6 Proceso de Carga de una Página Dinámica

Las páginas dinámicas en el servidor tienen la extensión **.shtml**, y se diferencian de otros archivos por la capacidad de modificar su contenido dinámicamente de acuerdo ciertas condiciones generadas por el programa de control del PLC.

Estas páginas son ideales para mostrar datos que puedan cambiar con el tiempo, como parámetros, señales analógicas, valores de entradas, etc.

Las páginas estáticas, aquellas con extensión **.html**, solo pueden mostrar un contenido fijo en el PLC, y son muy útiles para mostrar información constante en el tiempo (como por ejemplo datos de contacto, información de ayuda, encabezados, etc).

Las paginas dinámicas **.shtml** son procesadas por el PLC en el momento que se cargan por el navegador Web que las solicita, por lo tanto requieren más tiempo uso del procesador del PLC, respecto a las paginas estáticas **.html**.

Analicemos el contenido del archivo **index.shtml** del ejemplo actual (hacemos doble-click en el nombre para editarlo dentro del **Explorador de Proyecto**):

```
$!+ /header.html $

<h2>Principal</h2>

<center>

<p>¡Bienvenido al servidor Web del PLC!.</p>
<p>Click sobre el menú de arriba para navegar las páginas.</p>

</center>

$!+ /footer.html $
```

Como se puede apreciar, el archivo tiene contenido HTML, el cual es fácilmente identificable porque el lenguaje HTML utiliza TAGs, que son palabras claves (únicas) entre ángulos **< >** y que generalmente vienen de a pares encerrando un contenido:

```
<tag> contenido </tag>
```

Por ejemplo, en la página **index.shtml**, se usa el siguiente tag **<h2>** (párrafo tipo header o titulo) para describir el texto “Principal” y cerrando al final con el tag terminador **</h2>**:

```
<h2>Principal</h2>
```

Para escribir texto común tipo párrafo usamos ** **:

```
<p>¡Bienvenido al servidor Web del PLC!.</p>
<p>Click sobre el menú de arriba para navegar las páginas.</p>
```

Y para que quede todo centrado al medio, usamos el tag ** ** …. ** ** encerrando los párrafos con tags ** ** … ** .**

Existen muchísimos tags, que permiten crear textos en negrita, cursiva, tablas, marcos, etc… El lenguaje HTML permite hacer diseños muy logrados de forma simple. Recomendamos entrar en el siguiente link para comprender las nociones básicas del lenguaje HTML y los tags disponibles:

http://www.w3schools.com/html

También en la misma página proveen un editor para probar código HTML de forma simple y visualizarlo en tiempo real:

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_default

Hasta aquí el HTML para mostrar contenido estático, pero ¿ cómo hacemos en una página dinámica para mostrar contenido variable ?.

La respuesta es utilizar comandos o códigos especiales que solo el servidor Web del PLC pueda interpretar al cargar una página. Cuando los comandos son detectados por el servidor Web al ser cargada una página por el navegador Web del cliente, el servidor realiza una acción (como activar un evento o ejecutar el comando) antes de seguir cargando la página. Dicha acción, puede devolver contenido, que se suma al flujo de respuesta de la página y entonces el navegador muestra el nuevo contenido mientras carga la pagina.

Los comandos que son ejecutados del lado servidor, se conocen como CGI (Common Gateway Interface) y en el caso del servidor Web del PLC de Slicetex, los comandos CGI se delimitan por los signos **$!** y **$**. Un comando CGI permite interactuar y/o transferir datos entre el cliente Web y el Servidor Web.

En este ejemplo introduciremos los comandos CGI a medida que avancemos, sin embargo, la lista completa de **Comandos CGI Soportados por el Servidor** puede verla en la página 100.

Volviendo al ejemplo, en la página **index.shtml** tenemos el siguiente comando CGI:

```
$!+ /header.html $
```

Como vemos, está delimitado por los caracteres **$!** y **$**.

En este caso, el comando se llama "**+**" y permite incluir dentro de la página **index.shtml** el contenido de otra página Web, como **header.html**. Notar que debe anteponer la barra invertida "**/**" al nombre del archivo a incluir.

Este comando es muy útil, ya que nos permite ahorrar memoria, incluyendo el contenido constante de la página **header.html (**que contiene los encabezados HTML) y que se repite en cada página Web del servidor. Si este comendo no existiera, deberíamos copiar manualmente el contenido de **header.html** en la página **index.shtml** y en todas las paginas que compartan el mismo encabezado, aumentando su tamaño en memoria.

Al final de **index.shtml** nos topamos con el mismo comando, pero esta vez incluye otra página HTML con contenido común a todas las páginas del servidor, la pagina "**footer.html**", que contiene el pie de página:

```
$!+ /footer.html $
```

Cuando el servidor envía la página **index.shtml** al cliente y encuentra el comando **$!+ /header.html $**, automáticamente incluye el siguiente contenido (archivo **header.html**) en el flujo HTML de respuesta al cliente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="es">

<head>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
	<meta name="format-detection" content="telephone=no">
	<meta name = "viewport" content = "width = 320, initial-scale = 1.0, user-scalable = no">

	<title>Slicetex PLC Web Server!</title>
	<link rel="stylesheet" type="text/css" href="/style.css">
</head>

<body bgcolor="#fffeec" text="black">

<div class="menu">
	<div class="menubox"><a href="/">Inicio</a></div>
	<div class="menubox"><a href="/inputs.shtml">Entradas</a></div>
	<div class="menubox"><a href="/outputs.shtml">Salidas</a></div>
	<div class="menubox"><a href="/info.shtml">Info</a></div>
	<br>
</div>

<div class="contentblock">
```

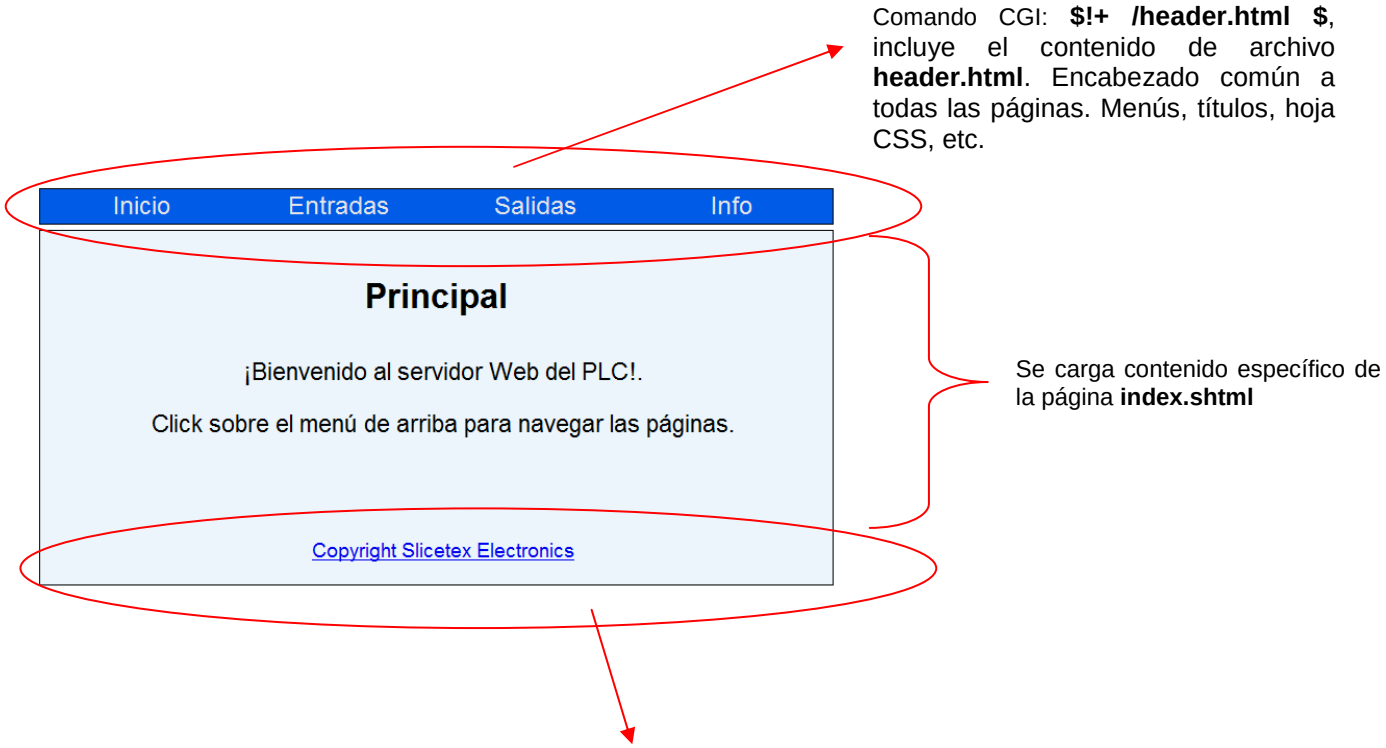
El cual define un titulo de pagina, tipo de documento HTML, idioma de la pagina Web, hoja de estilos CSS, icono del sitio Web y define un menú. Esto es completamente configurable de acuerdo al diseñador. Puede ver más información sobre este contenido en la sección **Recomendaciones de Diseño Web** en página 107.

Luego el servidor carga el restante contenido de **index.shml** hasta que se topa con **$!+ /footer.html $**, e incluye el siguiente código HTML:

```
<br>
<br>

<center>
	<p style="font-size:15px">
	<a  href="http://www.slicetex.com"  target="_blank">Copyright  Slicetex
Electronics</a>
	</p>
</center>

</div>
</body>
</html>
```

El cual no hace más que agregar información de copyright y cerrar ciertos TAGs abiertos en **header.html**.

Visualmente, el proceso de carga de **index.shtml** es el siguiente:



Este es el proceso de carga básico de una página dinámica **.shtml**, donde lo importante es comprender que las mismas son procesadas por el servidor en busca de comandos CGI que permiten modificar el contenido de la página en tiempo real, es decir mientras el cliente ordena la carga de la pagina en su navegador.

Las paginas estaticas **.html** no tienen esta particularidad, siempre muestran el mismo contenido y el servidor no las analiza en busca de comandos CGI.

A continuación veremos cómo modificar mediante los comandos CGI y el programa del PLC el contenido de una página dinámica.

5.7 Modificar Contenido de Página desde Programa en PLC

En esta sección explicaremos como modificar el contenido que se visualiza en una página dinámica **.shtml** desde el programa cargado en el PLC, usando para ello comandos CGI.

Este mecanismo nos da una gran potencialidad para representar información en tiempo real de acuerdo a la lógica implementada por el programa en el PLC. Por ejemplo valor de entradas, señales de control, etc.

Analicemos el contenido de la página **inputs.shtml**, la cual muestra el valor de la entrada **DIN1**:

```
$!+ /header.html $

<h2>Entradas Digitales</h2>

<center>

<table id="tInputs">
  <tr>
    <th>ENTRADA</th>
    <th>VALOR</th>
    <th>LED</th>
  </tr>
  <tr>
    <td>DIN1</td>
    <td>$! PrintStr 10 $</td>
    <td><img src="$! PrintStr 11 $"></td>
  </tr>
</table>

<br>
</center>

$!+ /footer.html $
```

Los comandos CGI: **$!+ /header.html $** y **$!+ /footer.html $,** ya fueron analizados en la sección 5.6, pagina 17. Solo incluyen el contenido de encabezado y pie de página común a todas la paginas.

El código HTML **<table> … </table>** crea una tabla donde cada fila se define con los tags **<tr> … </tr>,** y dentro de cada fila el valor de cada columna se define con los tags **<th> … </th>>**, mientras que para los títulos de cada columna se usa **<th> … </th>**, dando el siguiente aspecto:

ENTRADA	VALOR	LED
DIN1	0	

Si analizamos el código HTML, la parte donde se encuentran los comandos CGI es la siguiente:

```
<tr>
  <td>DIN1</td>
  <td>$! PrintStr 10 $</td>
  <td><img src="$! PrintStr 11 $"></td>
</tr>
```

El comando CGI en cuestión es "**PrintStr**" y el mismo se encuentra delimitado por los símbolos **$!** y **$**, símbolos únicos del servidor Web de los PLC de Slicetex.

Hay dos comandos CGI utilizados:

- **$! PrintStr 10 $**
- **$! PrintStr 11 $**

El comando **PrintStr** sirve para imprimir cadenas de texto provenientes del programa del PLC en la página Web y toma dos argumentos de la siguiente forma:

Sintaxis: **PrintStr Num Arg1**

- **Num:** Número de cadena a imprimir, valor entero.
- **Arg1:** Valor entero numérico de argumento, uso opcional.

La lista completa de **Comandos CGI Soportados por el Servidor** puede verla en la página 100.

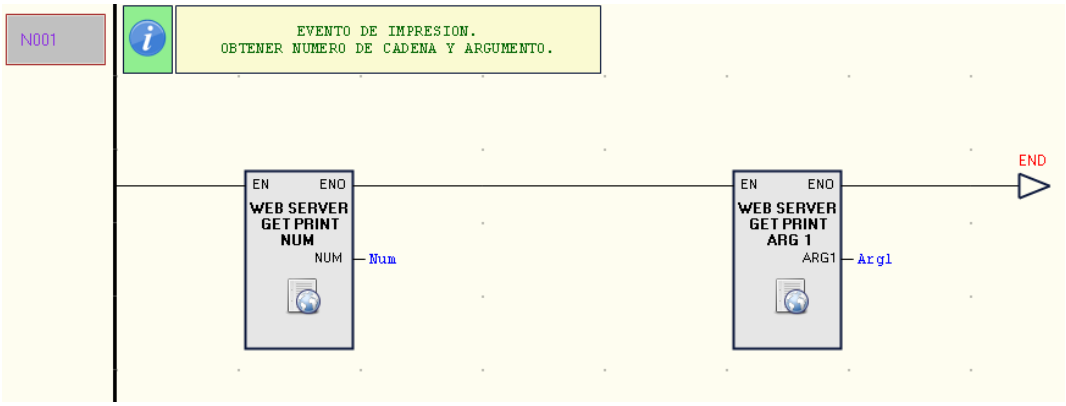
Cuando un navegador (cliente web) pide la pagina **inputs.shtml**, el servidor la carga, y la procesa hasta encontrar el comando **$! PrintStr 10 $**, punto en el cual realiza las siguientes acciones:

1. El servidor web pausa la carga de la página web.
2. Obtiene los valores numéricos de **Num** y **Arg1**.
3. Procede a ejecutar código del programa del PLC
4. Activa el evento **OnWebServerPrint**, ejecutando el diagrama **OnWebServerPrint.sld**.
5. El programa del usuario en el PLC procesa desde el diagrama **OnWebServerPrint.sld** los valores de los argumentos **Num** y **Arg1** recibidos.
6. De acuerdo al valor de **Num** y **Arg1**, el usuario desde el PLC devuelve una cadena con texto, la cual reemplazará completamente la etiqueta del comando "**$! PrintStr Num Arg1 $**". Como resultado neto, el cliente web con el navegador, solo recibirá la cadena de texto devuelta por el programa del PLC, pero no el texto del comando CGI. Esta cadena se almacena temporalmente en un buffer interno.
7. El usuario finaliza la ejecución del diagrama **OnWebServerPrint.sld**.
8. El servidor Web reanuda la carga de la pagina Web, pero reemplaza la etiqueta del comando "**$! PrintStr Num Arg1 $**" por el valor de la cadena de texto devuelta por el programa de usuario del PLC.
9. El servidor continúa con la carga del código HTML y procesamiento de los otros comandos CGI restantes en la página.

Los puntos importantes a tener en cuenta son del 5 al 7, ya que es la parte donde el programa del PLC toma control y la página web detiene su carga hasta que el programa del PLC devuelva el control al

servidor. Por lo tanto, es muy importante que retorne lo más rápido posible desde el evento **OnWebServerPrint**. De esta manera, la carga de la página web será más rápida. Evite realizar operaciones complejas que tomen mucho tiempo de procesador.

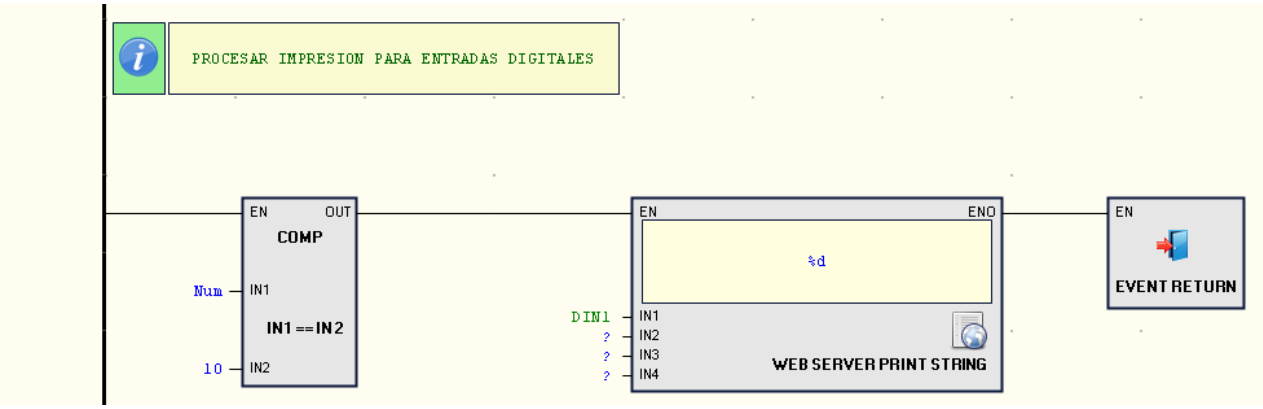
Volviendo al ejemplo en lenguaje Ladder, abramos el diagrama **OnWebServerPrint.sld** en **StxLadder**. Cuando el comando CGI con el valor **$! PrintStr 10 $** sea encontrado, el siguiente código Ladder es ejecutado al activarse el evento **OnWebServerPrint**:



Los componentes "**WEB SERVER GET PRINT NUM**" y "**WEB SERVER GET PRINT ARG1**" obtienen último valor de los argumentos **Num** y **Arg1** recibidos desde un comando CGI **PrintNum**. Ambos valores los copiamos en variables locales con el mismo nombre.

Conviene localizar estos componentes al principio del diagrama, de tal manera que se ejecuten lo antes posible para utilizarlos en código posterior. Por ello los colocamos en la Network N001.

Luego procesamos el valor **Num** recibido, para determinar el número de cadena que el Servidor Web solicita para imprimir.



El componente "**COMP**" compara si **Num == 10**, en caso afirmativo, se ejecuta el componente "**WEB SERVER PRINT STRING**". Este último componente nos permite imprimir una cadena de texto con formato.

En este caso se imprime la cadena "**%d**", la cual obtiene el valor entero de una variable de un puerto de entrada del componente. En este caso de IN1=**DIN1**. Ver documentación de componente para más información de uso.

Por último, en el código Ladder se ejecuta el componente "**EVENT RETURN**", el cual hace que inmediatamente se deje de procesar código en el diagrama Ladder del evento **OnWebServerPrint** y se retorne el control al Servidor Web. Código posterior a este componente no será ejecutado, ya que no es necesario porque ya logramos imprimir la cadena solicitada por el Servidor Web.

Entonces, si DIN1 es 0, la cadena resultante es "0" y en la página Web queda:

```
<td>0</td>
```

ENTRADA	VALOR	LED
DIN1	0	

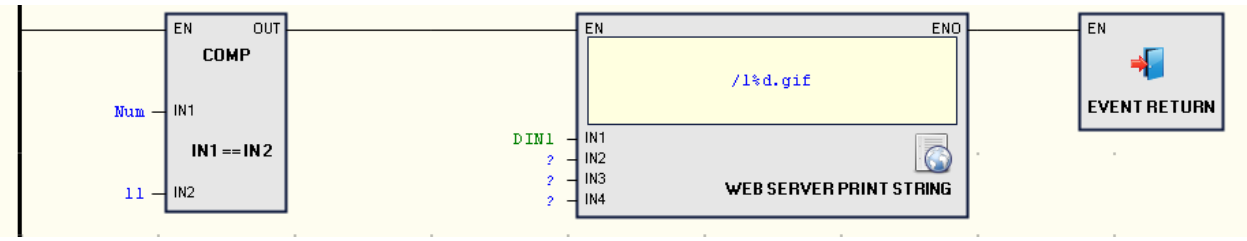
Si DIN1 es 1, la cadena resultante es "1" y en la página Web queda:

```
<td>1</td>
```

ENTRADA	VALOR	LED
DIN1	1	

Notar cómo se reemplaza el texto del comando **$! PrintStr 10 $** por el valor de la cadena retornada por el programa del PLC en la pagina dinámica.

Volviendo al diagrama, cuando el comando CGI sea **$! PrintStr 11 $**, el siguiente código Ladder se procesará:



En este caso si Num=11, el componente "**WEB SERVER PRINT STRING**" imprime la siguiente cadena:

"/l%d.gif"

La cual depende del valor reemplazado en el código de formato %d, que toma su valor del puerto IN1=DIN1. En otras palabras, puede retornar: "**/l0.gif**" o "**/l1.gif**".

Si nos fijamos en el **Explorador de Proyecto**, tenemos dos archivos de imágenes GIF cargados:

Archivo: l0.gif

Archivo: l1.gif

Ambas imágenes representan un LED apagado y un LED encendido. Ideal para representar el estado de una entrada digital como DIN1.

Entonces, si DIN1 es 0, la cadena resultante es "**/l0.gif**" y en la página Web queda:

```
<td><img src="/l0.gif"></td>
```

ENTRADA	VALOR	LED
DIN1	0	

Si DIN1 es 1, la cadena resultante es "**/l1.gif**" y en la página Web queda:

```
<td><img src="/l1.gif"></td>
```

ENTRADA	VALOR	LED
DIN1	1	

El código HMTL **<img scr="**url-imagen**">** le dice al navegador "cargar una imagen localizada en url-imagen", en este caso especificamos el camino hasta el archivo gif, por ejemplo **/l1.gif**.

```
<img src="/l1.gif">
```

El navegador al encontrar ese tag HTML, hace una petición extra al Servidor Web para descargar y mostrar la imagen.

Visualmente, este es el proceso cuando el servidor encuentra los comandos CGI:

Activa evento y ejecuta el diagrama **OnWebServerPrint.sld**

Se procesa argumentos **Num** y **Arg1**, para devolver la cadena de texto solicitada por el servidor.

```
<tr>
  <td>DIN1</td>
  <td>$! PrintStr 10 $</td>
  <td><img src="$! PrintStr 11 $"></td>
</tr>
```

5.8 Procesar Formularios Web y Recibir Datos del Navegador

En secciones previas vimos como modificar contenido dinámicamente en una página web para mostrar información que puede variar con el tiempo, como entradas, variables, etc.

Ahora veremos cómo recibir datos desde un Navegador Web, es decir desde el cliente al servidor. Esto nos permitirá interactuar con el servidor para pasar valores y controlar el funcionamiento del PLC remotamente.

El método utilizado para este fin en los Servidores Web de los PLC de Slicetex Electronics, es a través de formularios web (form) usando el método "**POST**".

La pagina **outputs.shtml** del ejemplo permite controlar la salida digital **DOUT1** del PLC. La misma consta de un formulario (con una tabla) y entradas de control tipo "**radio**" (permiten selección un valor entre varias opciones). Finalmente, un botón "**Actualizar**" que envía desde el navegador los valores seleccionado por los controles "**radio**".

Cuando en el navegador se haga click en "**Actualizar**", se enviarán los valores seleccionados en el formulario al Servidor Web.

Esto desencadena las siguientes acciones en el Servidor Web:

1. Activa el evento **OnWebServerPost**, ejecutando el diagrama **OnWebServerPost.sld**.
2. El programa del usuario en el PLC procesa desde el diagrama **OnWebServerPost.sld** comprobando los valores del formulario recibidos.
3. De acuerdo a valores del formulario recibidos, el usuario desde el PLC ejecuta una acción, como por ejemplo actualizar el valor de una salida digital.
4. El usuario finaliza la ejecución del diagrama **OnWebServerPost.sld**.

Analicemos el contenido de la página **outputs.shtml**:

```
$!+ /header.html $

<center>
<h2>Salidas Digitales</h2>

<form action="/outputs.shtml" method="post">
<input type="hidden" name="Form" value="Outputs">
      <table id="tOutputs">
        <tr>
          <th>SALIDA</th>
          <th>VALOR</th>
        </tr>
        <tr>
          <td>DOUT1</td>
          <td>
            <input type="radio" name="DO1" value="0" $! PrintStr 20 $> OFF
            <br>
            <input type="radio" name="DO1" value="1" $! PrintStr 21 $> ON
          </td>
        </tr>
      </table>
      <br>
      <input type="submit" value="Actualizar" class="update">
      <br>
</form>

<br>
</center>

$!+ /footer.html $
```

Dejemos de lado los comandos CGI delimitados por los caracteres **$!** y **$**, y comprendamos el código.

Un formulario HTML (form) se define como:

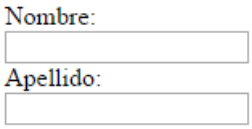
```
<form>
…
elementos form
…
</form>
```

El elemento **<input>** es el más importante de los elementos **form**, ya que el mismo nos permite de acuerdo a su tipo definir entradas para texto, controles radio, botones, etc.

Ejemplo:

```
<form>
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
</form>
```

Produce la siguiente salida en un navegador:



Este formulario tiene solo dos entradas “input” del tipo “text”. Ambas nos permiten introducir texto.

```
<input type="text" name="Nombre">
```

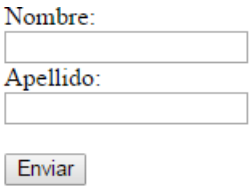
```
<input type="text" name="Apellido">
```

El atributo “**name**” nos permite agregar un nombre a las “**input**” para luego poder identificarlas cuando recibamos los valores en el Servidor Web.

Sin embargo, nos falta el botón para enviar los datos al Servidor Web. Esto se puede agregar con otro “input” del tipo “submit”.

```
<form>
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

Resultando la siguiente visualización en el navegador Web:



Ya es posible enviar los datos al Servidor Web haciendo click en el botón “Enviar”, sin embargo para el Servidor Web del PLC necesitamos agregar lo siguiente en cada formulario creado:

```
<form action="/pagina_destino.html" method="post">
<input type="hidden" name="Form" value="Nombres">
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

Agregamos a <form> los siguientes atributos:

- **action="/pagina_destino.html":** Es de uso opcional, pero nos permite especificar una página de destino que se cargará cuando se haga click en el botón "**submit**". Puede ser la misma página del formulario u otra diferente. Dinámica o estática.

- **method="post"**: Este es el método a utilizar por el navegador cuando se envíen los datos del formulario al Servidor Web. Utilizamos el método POST.

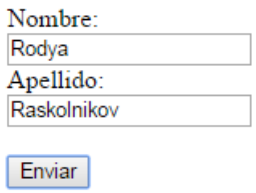
Finalmente, agregamos otro "**input**" al formulario para identificarlo de otros formularios en diferentes páginas web del Servidor.

```
<input type="hidden" name="Form" value="Nombre_Formulario">
```

Donde:

- **type="hidden"**: Hace que esta entrada del formulario no sea visible en el navegador.
- **name="Form"**: Nombre de la "**input**", la llamamos **Form** por convención.
- **value="Nombre_Formulario"**: Nombre del formulario, puede ser cualquiera pero único para cada formulario del servidor.

En los servidores de Slicetex, el valor de "**name**" de cada "**input**" no puede contener espacios, ni caracteres especiales. Solo texto o números, y es buena práctica no hacerlos demasiados extensos. También no pueden repetirse los valores de "**names**" de los "**input**" de un mismo formulario.



Ahora cuando hagamos click en el botón "**Enviar**" del formulario, se envía la siguiente información al Servidor Web en forma de campos (fields), los cuales siempre están de a pares con sus valores (values) en el siguiente orden:

1. Form = Nombres
2. Nombre = Rodya
3. Apellido = Raskolnikov

Cada dato esta en el formato "**Field = Value**", dentro del Servidor Web podemos obtener los valores de cada campo. Para ello debemos especificar el nombre del campo, por ejemplo "**Nombre**" y una función nos devolverá la cadena "**Rodya**", obteniendo de esta forma dicho valor.

Es evidente, que podemos de esta forma enviar números, texto, etc remotamente al servidor.

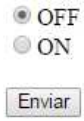
Cuando el Servidor Web recibe los datos de un formulario web con el método POST, genera un evento del tipo **OnWebServerPost**. Allí podemos procesar los datos recibidos.

Si queremos utilizar un formulario con un control tipo "**radio**" como el del ejemplo, podemos escribir el siguiente código HTML:

```
<form action="/outputs.shtml" method="post>
<input type="hidden" name="Form" value="Outputs">
  <input type="radio" name="DO1" value="0" checked>OFF
  <br>
  <input type="radio" name="DO1" value="1">ON
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

En color rojo, pusimos el código importante a observar. En los elementos "**input**" utilizamos el atributo "**type=radio**", usando el mismo "**name=DO1**" para cada input ya que este tipo de control solo envía una opción de varias entradas. Cada entrada "input" tiene un "**value**" diferente, en este caso "1" o "0". Finalmente el atributo "**checked**" permite seleccionar el radio por defecto.

En un navegador resulta la siguiente visualización:



Si hacemos click en "Enviar", los siguientes campos se envían:

1. Form = Outputs
2. DO1 = 0

Se envía el campo DO1=0, porque el valor seleccionado es "OFF".

Volvamos al código de **outputs.shtml**, la parte del código del formulario la marcamos en rojo**:**

```
<form action="/outputs.shtml" method="post">
<input type="hidden" name="Form" value="Outputs">
      <table id="tOutputs">
        <tr>
          <th>SALIDA</th>
          <th>VALOR</th>
        </tr>
        <tr>
          <td>DOUT1</td>
          <td>
            <input type="radio" name="DO1" value="0" $! PrintStr 20 $> OFF
            <br>
            <input type="radio" name="DO1" value="1" $! PrintStr 21 $> ON
          </td>
        </tr>
      </table>
      <br>
      <input type="submit" value="Actualizar" class="update">
      <br>
</form>
```

Como podemos observar, es muy similar a los formularios que vimos de ejemplo, solo que esta insertado dentro de una tabla HTML (<table>) para darle una mejor visualización.

Por otro lado hay dos comandos CGI adentro del código del formulario:

- $! PrintStr 20 $
- $! PrintStr 21 $

Ambos comandos CGI son procesados en el diagrama "**OnWebServerPrint.sld**" cuando se carga la pagina, y solo insertan el texto "**checked**" de acuerdo al valor de la salida digital DOUT1, permitiendo seleccionar el radio por default según corresponda (OFF u ON) automáticamente. Esto lo veremos más adelante.

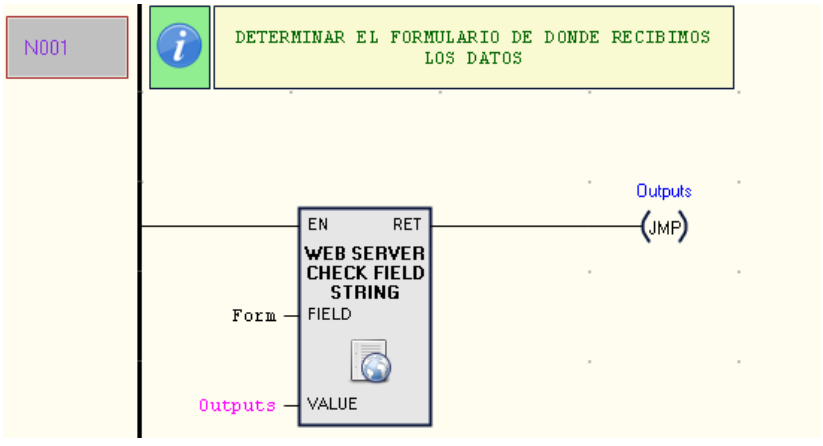
SALIDA	VALOR
DOUT1	(•) OFF ( ) ON

Actualizar

Si hacemos click en el botón "**Actualizar**" los siguientes campos se envían al Servidor Web:

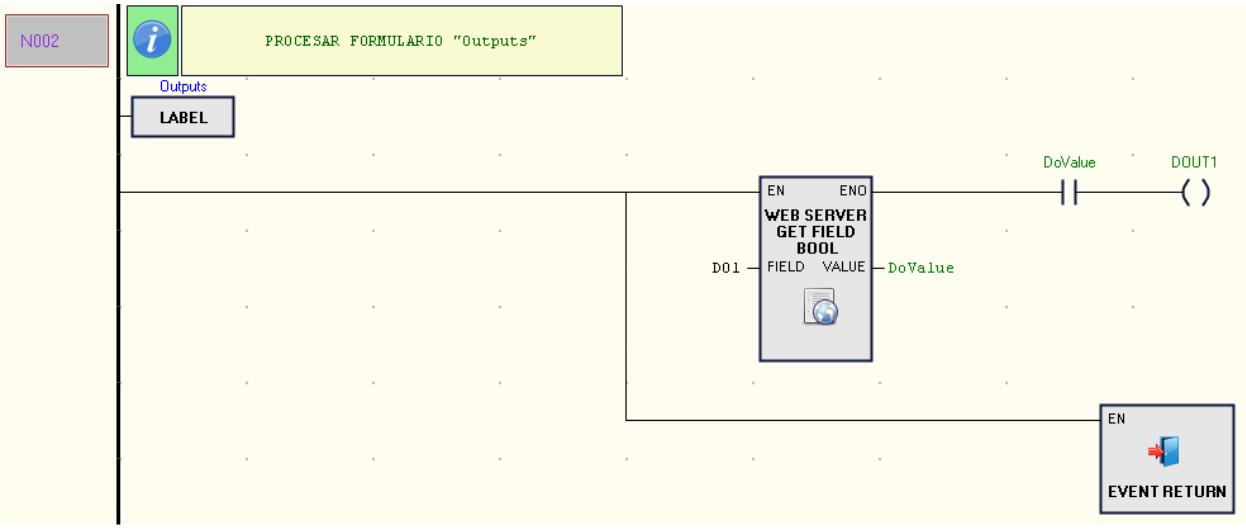
1. Form = Outputs
2. DO1 = 0

Cuando el Servidor Web recibe los datos del formulario web con el método POST, genera un evento del tipo **OnWebServerPost**. Se ejecuta el diagrama Ladder "**OnWebServerPost.sld**" e inicialmente se ejecuta la **Network N001** con el siguiente código:



El componente "**WEB SERVER CHECK FIELD STRING**", busca dentro de los datos recibidos desde el formulario, el campo FIELD = "Form" y si tiene el valor VALUE = "Outputs", hace su salida RET=1 y ejecuta la bobina JMP hasta la etiqueta "**Outputs**" en el diagrama. Esto nos permite discriminar entre diferentes formularios.

Una vez producido el salto hacia "**Outputs**" en el diagrama, se ejecuta el siguiente código:



Aquí se ejecuta el componente "**WEB SERVER GET FIELD BOOL**", que busca el campo **FIELD = DO1** y obtiene su valor del tipo **Bool**, es decir "1" o "0" y lo escribe en la variable local "**DoValue**" si lo encuentra.

Luego leemos el valor de la variable "**DoValue**" y de acuerdo a su valor escribimos un "0" o "1" en la salida digital DOUT1 con el componente bobina.

Finalmente, se ejecuta el componente "**EVENT RETURN**" para retornar lo antes posible del diagrama evento, así el PLC puede seguir procesando otras operaciones.

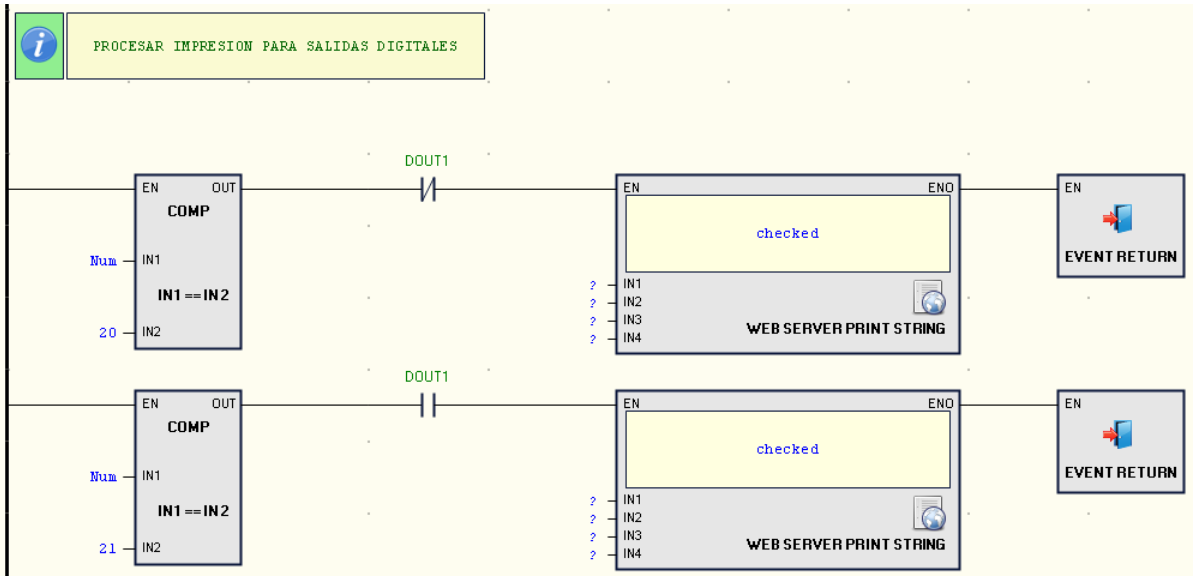
Hasta aquí vimos como controlar y enviar datos remotamente al PLC desde un Navegador Web.

Los formularios son un método muy potente para operaciones de control, aconsejamos investigar más sobre los mismos en el siguiente link:

http://www.w3schools.com/html/html_forms.asp

Luego de hacer click en el botón "Actualizar", el formulario especifica `action="/outputs.shtml"`, esto quiere decir que el navegador nuevamente pedirá cargar la pagina "**outputs.shtml**", provocando que el Servidor Web la procese nuevamente. Y como teníamos dos comandos CGI del tipo **PrintStr** se ejecutará dos veces el diagrama **OnWebServerPrint.sld**, porque el evento **OnWebServerPrint** será disparado dos veces.

El siguiente código del diagrama "**OnWebServerPrint.sld**" será ejecutado de acuerdo a los argumentos del comando CGI del tipo **PrintStr**:



Cuando **Num** sea igual a 20, si el valor de **DOUT1** es "0" se devuelve la cadena "**checked**" a la página Web. De lo contrario, no se devuelve nada, una cadena vacía.

Cuando **Num** sea igual a 21, si el valor de **DOUT1** es "1" se devuelve la cadena "**checked**" a la página Web. De lo contrario, no se devuelve nada, una cadena vacía.

El valor "**checked**" permite especificar un atributo para seleccionar el "**radio**" correcto en el formulario. De esta forma, indicamos el valor actual de la salida **DOUT1** indirectamente.

El comando CGI del tipo **PrintStr** lo vimos en la sección **Modificar Contenido de Página desde Programa en PLC** de la página 21.

Una nota final, si los datos enviados por el formulario al PLC son excesivos, fuera del límite soportado, el PLC carga la página **413.html** mostrándole al usuario el error.

5.9 Fin del Ejemplo

Estas son las nociones básicas para usar el Servidor Web. Puede usar este ejemplo y otros disponibles en nuestro sitio de Internet como base para sus proyectos.

Lea las descripciones de cada componente desde el entorno StxLadder antes de usarlos, para mejorar la comprensión de los mismos.

Recomendamos altamente que siga leyendo las siguientes secciones:

- Referencia de Componentes en Lenguaje Ladder, página 61
- Comandos CGI Soportados por el Servidor, página 100
- Opciones de Configuración del Servidor, página 103
- Recomendaciones de Diseño Web, página 107

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 servidor Web rápidamente.

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

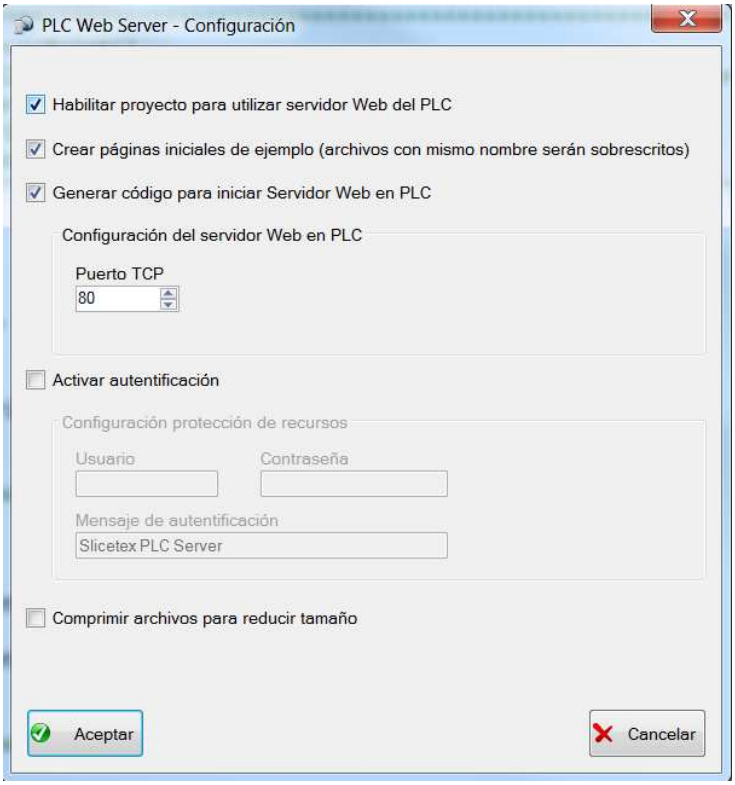
En esta sección expondremos un ejemplo básico en lenguaje Pawn, si está usando lenguaje Ladder, puede ir directamente al ejemplo Ladder en la página 8.

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

6.1 Crear el Proyecto

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

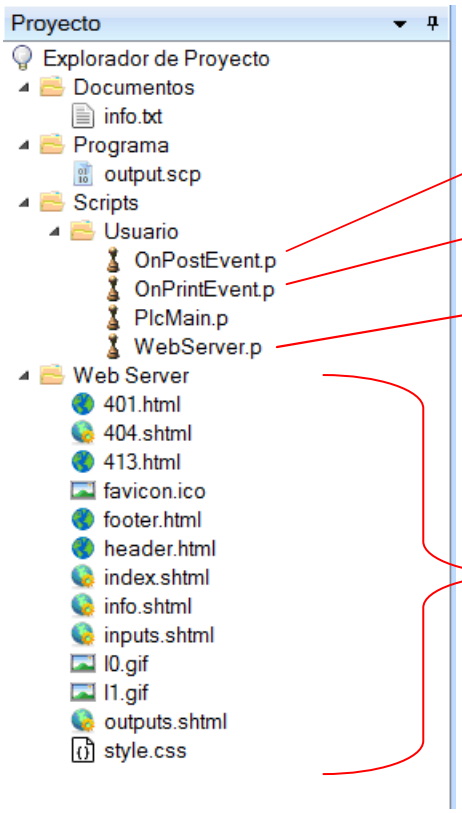
Active el uso del servidor Web en el proyecto seleccionando el menú "**Herramientas > PLC Web Server**". La siguiente ventana aparecerá:



Tilde las primeras tres opciones, para activar el servidor, crear páginas web iníciales y generar código Pawn en el proyecto, como muestra la figura superior.

Haga click en el botón "**Aceptar**" para efectuar los cambios.

Si el Servidor Web pudo configurarse con éxito en el proyecto, los siguientes archivos se agregaron al **Explorador del Proyecto**:



Rutinas para evento **OnWebServerPost**. Procesa formularios Web.

Rutinas para evento **OnWebServerPrint**. Imprime cadenas de texto en páginas Web dinámicas.

Se agregan rutinas para configurar el Servidor Web en el PLC.

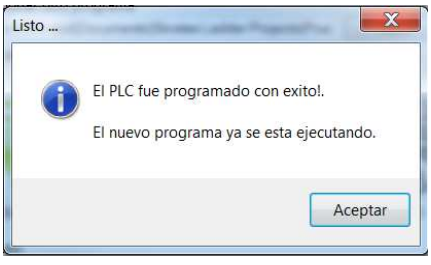
Archivos Web de ejemplo.
Este es el sistema de archivos accesibles a través de un navegador Web que se conecte al PLC.
Los archivos aquí depositados luego serán copiados a la memoria libre del PLC.

Notar como hay múltiples archivos con diferentes extensiones, SHTML, HMTL, GIF, CSS, etc.

Cuando dichos archivos son accedidos por un navegador Web, pueden disparar ciertos eventos en el PLC, y de esta forma interactuar con el código del proyecto.

Ahora que ya generó automáticamente los archivos básicos necesarios para utilizar el servidor Web en el PLC, puede compilar y transferirlos al PLC.

Entonces, conecte el PLC a la red Ethernet y seleccione el menú "**PLC > Cargar rápidamente**" en la barra de menús en **StxLadder**. El siguiente cartel se mostrará si el proyecto fue transferido al PLC:



6.2 Acceda a la Página Web del PLC

El próximo paso para probar el servidor Web del PLC, es accederlo con su navegador Web favorito (como Chrome, Explorer, Firefox, Edge, etc) desde la PC o desde una tablet / smartphone.

Para acceder con el navegador Web, debe colocar la dirección IP del PLC en la barra de direcciones del navegador. Por ejemplo, si el PLC se encuentra en la IP **192.168.1.81** de su red LAN, coloque en el navegador la siguiente dirección:

http://192.168.1.81

Si su red LAN está configurada correctamente y el proyecto fue cargado en el PLC, el navegador debe mostrar la siguiente página Web de inicio:

Si puede ver la pagina Web, ¡ya configuró con éxito el servidor Web del PLC!.

La pagina principal del PLC, corresponde al archivo **index.html** o **index.shtml** (de acuerdo a cual este cargado) y se carga por defecto al ingresar la dirección del PLC en el navegador.

Puede navegar por el servidor Web haciendo click en las páginas disponibles en el menú de la parte superior de la página Principal.

Si hace click en "**Entradas**" se mostrará la siguiente página que permite ver el estado de la entrada digital **DIN1** del PLC:

Note que la dirección en la barra de herramientas del navegador se transformó en:

http//192.168.1.81/inputs.shtml

Ya que se está accediendo al archivo "**inputs.shtml**".

Si polariza la entrada "**DIN1**" del PLC, y luego refresca o recarga la pagina "**inputs.shtml**", la pagina debería reflejar el nuevo valor de **DIN1** de la siguiente manera:

Como se puede notar, ahora **VALOR=1** y el **LED=Activado**. Esto significa que usted está cargando una página con contenido dinámico, es decir, la información mostrada puede cambiar según las condiciones de un programa cargado en el PLC. Las páginas dinámicas tienen extensiones **.shtml** en el PLC.

Si hace click en "**Salidas**" se mostrará la siguiente página que permite controlar la salida digital **DOUT1** del PLC:

Note que la dirección en la barra de herramientas del navegador se transformó en:

http//192.168.1.81/outputs.shtml

Ya que se está accediendo al archivo "**outputs.shtml**".

Esta página permite enviar datos desde el navegador Web (cliente) al PLC (servidor Web). El mecanismo que se utiliza es el método POST, que veremos más adelante.

Básicamente consiste en un formulario Web (que puede contener strings, números, o elementos de control como botones) y los valores de los mismos se envían al PLC mediante campos delimitados.

En este caso, si tocamos en "**ON**" y luego damos click en el botón "**Actualizar**" de la página, podremos activar la salida digital DOUT1 del PLC.

Esta página también es dinámica porque en el campo VALOR se selecciona OFF / ON automáticamente de acuerdo al valor de la salida **DOUT1** cuando se carga la pagina. Esto se hace por código.

6.3 Configuración del Servidor Web

Para utilizar el servidor Web en el PLC, el mismo debe ser activado y configurado. Si abrimos el archivo "**WebServer.p**" podremos ver el siguiente código:

```
// ****************************************************************************
// Funcion     : WebServerInitialize()
// Descripcion : Inicialización para utilizar el Web Server en PLC.
// ****************************************************************************

WebServerInitialize()
{
   // Inicializar Web Server.
   // Puerto: 80
   WebServerInit(80)

   // Activar evento  @OnWebServerPrint() para imprimir
   // cadenas en paginas dinamicas.
   WebServerSetPrintEvent()

   // Activar evento  @OnWebServerPost() para procesar datos
   // recibidos de formularios..
   WebServerSetPostEvent()
}
```

La función "**WebServerInit()**" activa el servidor Web, especificando el puerto donde se escucharán las conexiones TCP por defecto cuando un cliente se conecte. En este caso se eligió el puerto **80** que es el estándar. Más información de la función en pagina 79.

Si selecciona otro puerto, por ejemplo el numero **88** (ya sea por seguridad o para diferenciar de otros PLC conectados en una misma red), para acceder con un navegador deberá escribir el numero de puerto separado por ":" en la barra de direcciones, por ejemplo: http://192.168.1.81:88

La función "**WebServerSetPrintEvent()**", activa el evento **OnWebServerPrint**, el cual sirve para imprimir desde el PLC texto en las páginas Web dinámicas (archivos con extensión **.shtml**) cuando el servidor lo requiere.

Por ejemplo, si el usuario cargó desde el navegador la página **inputs.shml** que muestra el estado de la entrada digital **DIN1**, el servidor le pedirá al PLC que devuelva el estado de la entrada en tiempo real. Esto lo hace generando el evento **OnWebServerPrint**, el cual provoca que la función **@OnWebServerPrint()** en archivo **OnPrintEvent.p** se ejecute en el PLC, y en el mismo se procesa el requerimiento para devolver una cadena de texto con el estado de la entrada **DIN1**.

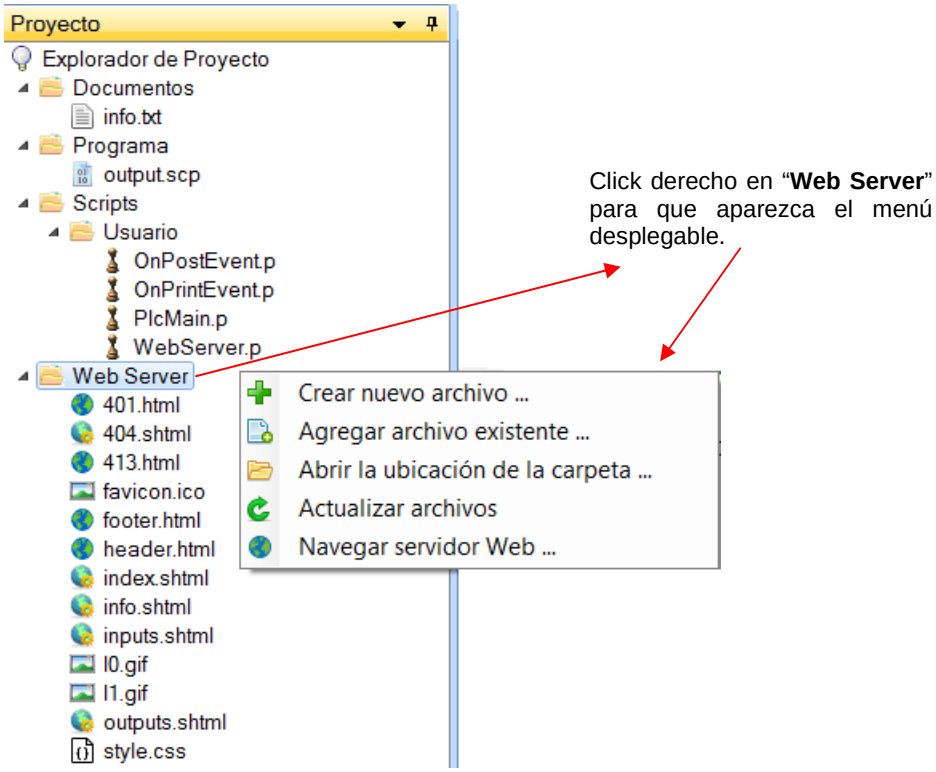
La función "**WebServerSetPostEvent()**", activa el evento **OnWebServerPost**, el cual sirve para procesar desde el PLC formularios Web, los cuales envían datos desde el navegador al PLC. Son muy útiles para realizar tareas de control o enviar parámetros al PLC.

Por ejemplo, cuando el navegador accede a la página **outputs.html** y se hace click en el botón "**Actualizar**" para conmutar el estado de la salida **DOUT1**, el navegador envía el dato con el nuevo valor de **DOUT1** al servidor del PLC y el mismo genera el evento **OnWebServerPost.** Dicho evento dispara la ejecución de la función **@OnWebServerPost()** en archivo **OnPostEvent.p**, que permite procesar el dato recibido y controlar la salida **DOUT1**.

6.4 Creando Archivos y Páginas Web

Desde el entorno **StxLadder** es muy fácil editar y cargar contenido Web en el servidor del PLC.

Para cargar contenido, desde el **Explorador de Proyecto** en **StxLadder** haga click derecho sobre la carpeta "**Web Server**". El siguiente menú desplegable aparecerá:



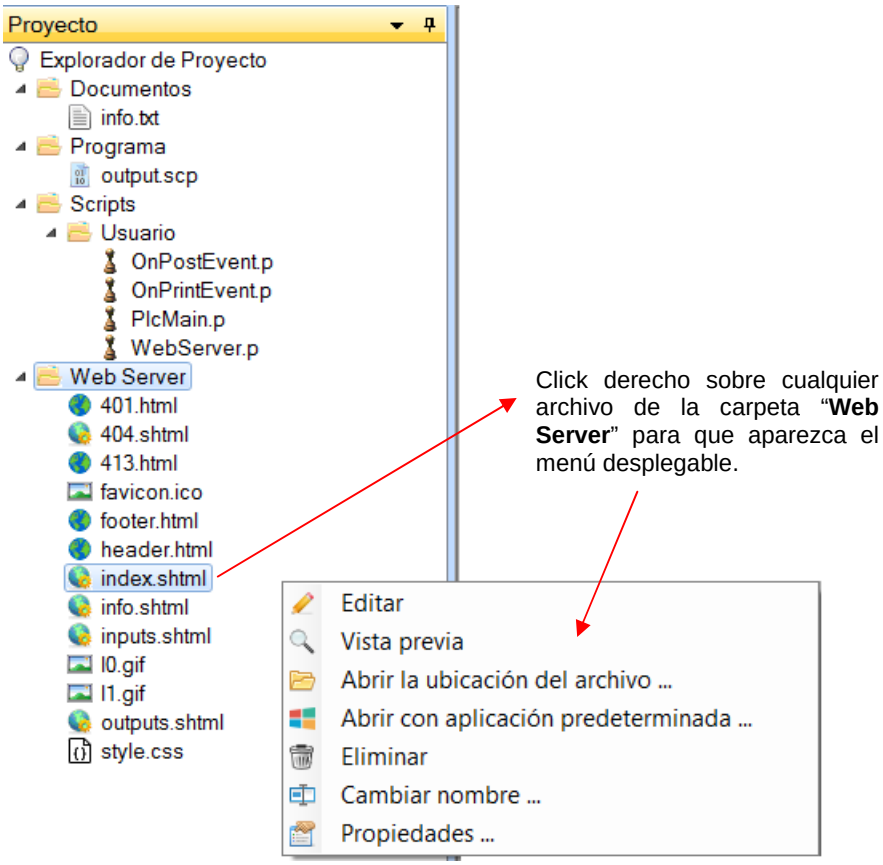
Este menú permite agregar nuevos archivos y archivos existentes al PLC. La forma más simple es utilizar "**Crear nuevo archivo …"** o "**Agregar nuevo archivo existente …**".

Si necesita acceder directamente a la carpeta contenedora de los archivos del PLC (ya sea para editarlos externamente, o agregar muchos al mismo tiempo, etc) seleccione "**Abrir la ubicación de la carpeta …**". Recuerde luego seleccionar "**Actualizar archivos**" si agrego o borro archivos externamente.

Si desea explorar el servidor Web desde el navegador de Web de StxLadder seleccione "**Navegar servidor Web …**" y por defecto se navega en la dirección IP del PLC configurada por el proyecto.

La lista de extensiones de archivos permitidos y obligatorios se puede consultar en la **Tabla 2** y **Tabla 3**, de las páginas 5 y 6.

Para editar contenido, desde el **Explorador de Proyecto** en **StxLadder** haga click derecho sobre cualquier archivo de la carpeta "**Web Server**". El siguiente menú desplegable aparecerá:



Las opciones del menú desplegable se explican por sí solas, las mismas permiten: editar, eliminar, cambiar nombre, etc al archivo seleccionado.

Si hace doble-click sobre el archivo, se ejecuta la primera opción por defecto, en este caso "**Editar**". Para un archivo con extensión .html, .shtml, .css, .txt o .xml, es posible editarlo desde **StxLadder**. Lo cual es muy recomendado y cómodo.

Alternativamente, puede usar su editor externo favorito, para ello seleccione "**Abrir la ubicación del archivo …**", y una carpeta de Windows se abrirá donde puede editar el archivo con su aplicación externa. No se recomienda editar un archivo desde StxLadder y al mismo tiempo desde una aplicación externa. En ese caso, realice dicha operación por separado.

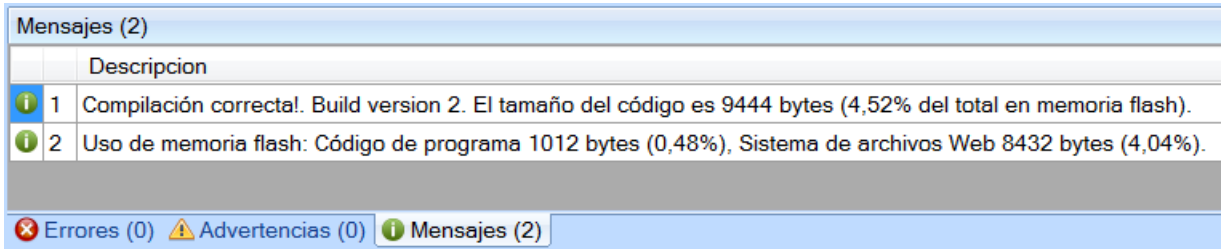
La opción "**Abrir con aplicación predeterminada …"** abre el archivo con la aplicación de Windows predeterminada para dicha extensión.

La opción "**Propiedades** …" es muy útil para consultar el tamaño en bytes de un archivo, de esta manera saber cuáles son mas "pesados" en caso que tenga poco espacio disponible en el PLC.

6.5 Memoria Utilizada

El PLC tiene una memoria abundante para código de programa, sin embargo, en general los archivos Web consumen mucha memoria si no se utilizan con cuidado. Para optimizar la memoria utilizada, siga los consejos en la sección "**Consideraciones para Optimizar el Funcionamiento"** de la página 6.

Para consultar la memoria utilizada por el los archivos Web en el PLC, debe compilar el proyecto desde el menú "**Proyecto > Compilar**" y observar el área de mensajes de **StxLadder**:



El mensaje numero 2, nos dice que en este proyecto **8432** bytes de memoria flash del PLC se utilizarán para almacenar los archivos Web, es decir un **4.04%** del total disponible para el modelo de PLC usado en el proyecto.

Esto nos deja un 95.96% de espacio para código de programa. Lo cual es bastante.

En este ejemplo, el programa solo ocupa 1012 bytes de memoria flash, el 0.48% del total para el modelo de PLC usado en el proyecto.

Tenga en cuenta que los archivos Web del servidor, se alojan en la memoria flash libre no ocupada por código de programa, esto quiere decir, que a mayor programa, menor espacio para pagina Web tenemos, y viceversa.

Puede visualizar desde el **Explorador de Proyecto** en StxLadder el tamaño de los archivos Web cargados, seleccionando la opción "**Propiedades …**" mediante click derecho sobre el archivo en cuestión. Esto puede ser útil para control el tamaño de los mismos a la hora de reducir espacio.

Es recomendable usar la menor cantidad de código HTML o imágenes en las páginas Web. Esto se logra conociendo la tecnología HTML y evitando usar diseñadores o editores Web complejos. En otras palabras, estudie el código HTML y edítelo manualmente con StxLadder.

6.6 Proceso de Carga de una Página Dinámica

Las páginas dinámicas en el servidor tienen la extensión **.shtml**, y se diferencian de otros archivos por la capacidad de modificar su contenido dinámicamente de acuerdo ciertas condiciones generadas por el programa de control del PLC.

Estas páginas son ideales para mostrar datos que puedan cambiar con el tiempo, como parámetros, señales analógicas, valores de entradas, etc.

Las páginas estáticas, aquellas con extensión **.html**, solo pueden mostrar un contenido fijo en el PLC, y son muy útiles para mostrar información constante en el tiempo (como por ejemplo datos de contacto, información de ayuda, encabezados, etc).

Las paginas dinámicas **.shtml** son procesadas por el PLC en el momento que se cargan por el navegador Web que las solicita, por lo tanto requieren más tiempo uso del procesador del PLC, respecto a las paginas estáticas **.html**.

Analicemos el contenido del archivo **index.shtml** del ejemplo actual (hacemos doble-click en el nombre para editarlo dentro del **Explorador de Proyecto**):

```
$!+ /header.html $

<h2>Principal</h2>

<center>

<p>¡Bienvenido al servidor Web del PLC!.</p>
<p>Click sobre el menú de arriba para navegar las páginas.</p>

</center>

$!+ /footer.html $
```

Como se puede apreciar, el archivo tiene contenido HTML, el cual es fácilmente identificable porque el lenguaje HTML utiliza TAGs, que son palabras claves (únicas) entre ángulos **< >** y que generalmente vienen de a pares encerrando un contenido:

```
<tag> contenido </tag>
```

Por ejemplo, en la página **index.shtml**, se usa el siguiente tag **<h2>** (párrafo tipo header o titulo) para describir el texto “Principal” y cerrando al final con el tag terminador **</h2>**:

```
<h2>Principal</h2>
```

Para escribir texto común tipo párrafo usamos ** **:

```
<p>¡Bienvenido al servidor Web del PLC!.</p>
<p>Click sobre el menú de arriba para navegar las páginas.</p>
```

Y para que quede todo centrado al medio, usamos el tag ** ** …. ** ** encerrando los párrafos con tags ** ** … ** .**

Existen muchísimos tags, que permiten crear textos en negrita, cursiva, tablas, marcos, etc… El lenguaje HTML permite hacer diseños muy logrados de forma simple. Recomendamos entrar en el siguiente link para comprender las nociones básicas del lenguaje HTML y los tags disponibles:

http://www.w3schools.com/html

También en la misma página proveen un editor para probar código HTML de forma simple y visualizarlo en tiempo real:

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_default

Hasta aquí el HTML para mostrar contenido estático, pero ¿ cómo hacemos en una página dinámica para mostrar contenido variable ?.

La respuesta es utilizar comandos o códigos especiales que solo el servidor Web del PLC pueda interpretar al cargar una página. Cuando los comandos son detectados por el servidor Web al ser cargada una página por el navegador Web del cliente, el servidor realiza una acción (como activar un evento o ejecutar el comando) antes de seguir cargando la página. Dicha acción, puede devolver contenido, que se suma al flujo de respuesta de la página y entonces el navegador muestra el nuevo contenido mientras carga la pagina.

Los comandos que son ejecutados del lado servidor, se conocen como CGI (Common Gateway Interface) y en el caso del servidor Web del PLC de Slicetex, los comandos CGI se delimitan por los signos **$!** y **$**. Un comando CGI permite interactuar y/o transferir datos entre el cliente Web y el Servidor Web.

En este ejemplo introduciremos los comandos CGI a medida que avancemos, sin embargo, la lista completa de **Comandos CGI Soportados por el Servidor** puede verla en la página 100.

Volviendo al ejemplo, en la página **index.shtml** tenemos el siguiente comando CGI:

```
$!+ /header.html $
```

Como vemos, está delimitado por los caracteres **$!** y **$**.

En este caso, el comando se llama "**+**" y permite incluir dentro de la página **index.shtml** el contenido de otra página Web, como **header.html**. Notar que debe anteponer la barra invertida "**/**" al nombre del archivo a incluir.

Este comando es muy útil, ya que nos permite ahorrar memoria, incluyendo el contenido constante de la página **header.html (**que contiene los encabezados HTML) y que se repite en cada página Web del servidor. Si este comendo no existiera, deberíamos copiar manualmente el contenido de **header.html** en la página **index.shtml** y en todas las paginas que compartan el mismo encabezado, aumentando su tamaño en memoria.

Al final de **index.shtml** nos topamos con el mismo comando, pero esta vez incluye otra página HTML con contenido común a todas las páginas del servidor, la pagina "**footer.html**", que contiene el pie de página:

```
$!+ /footer.html $
```

Cuando el servidor envía la página **index.shtml** al cliente y encuentra el comando **$!+ /header.html $**, automáticamente incluye el siguiente contenido (archivo **header.html**) en el flujo HTML de respuesta al cliente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="es">

<head>
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
	<meta name="format-detection" content="telephone=no">
	<meta name = "viewport" content = "width = 320, initial-scale = 1.0, user-scalable = no">

	<title>Slicetex PLC Web Server!</title>
	<link rel="stylesheet" type="text/css" href="/style.css">
</head>

<body bgcolor="#fffeec" text="black">

<div class="menu">
	<div class="menubox"><a href="/">Inicio</a></div>
	<div class="menubox"><a href="/inputs.shtml">Entradas</a></div>
	<div class="menubox"><a href="/outputs.shtml">Salidas</a></div>
	<div class="menubox"><a href="/info.shtml">Info</a></div>
	<br>
</div>

<div class="contentblock">
```

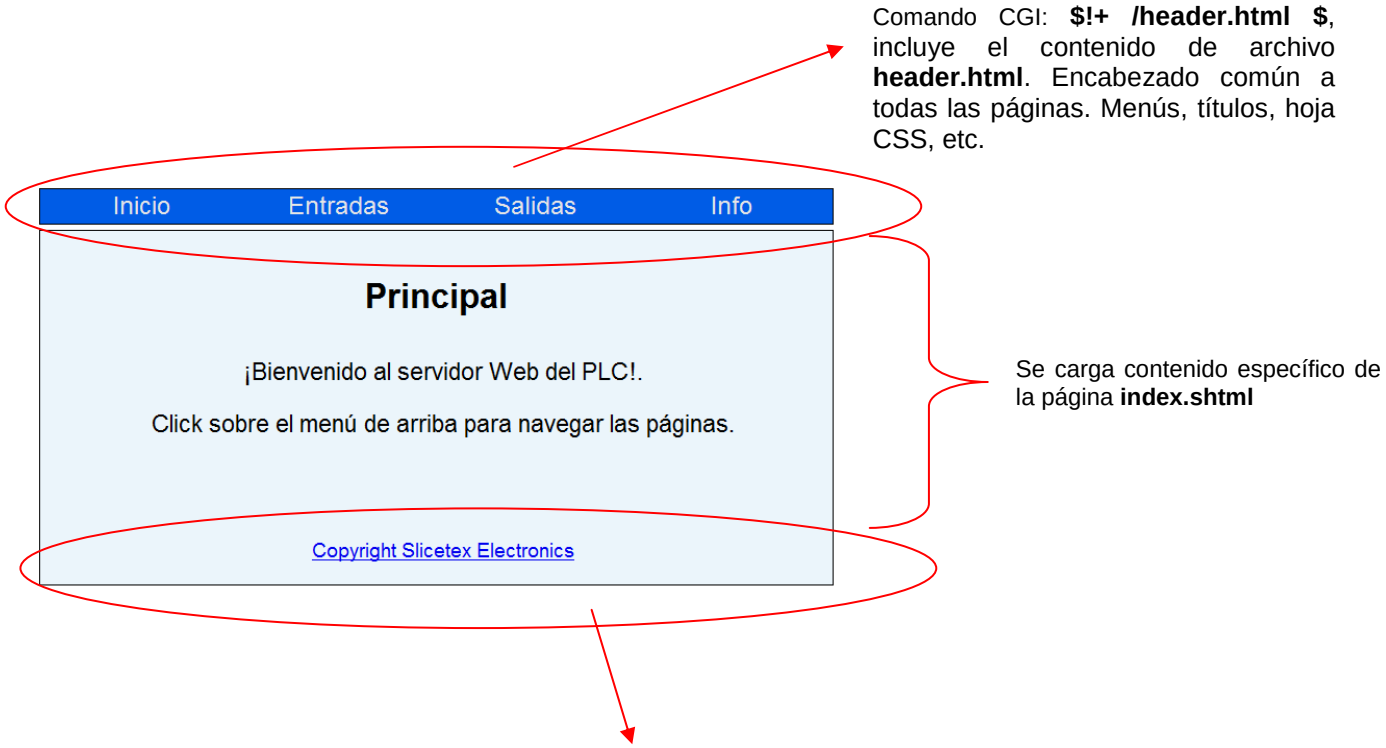
El cual define un titulo de pagina, tipo de documento HTML, idioma de la pagina Web, hoja de estilos CSS, icono del sitio Web y define un menú. Esto es completamente configurable de acuerdo al diseñador. Puede ver más información sobre este contenido en la sección **Recomendaciones de Diseño Web** en página 107.

Luego el servidor carga el restante contenido de **index.shml** hasta que se topa con **$!+ /footer.html $**, e incluye el siguiente código HTML:

```
<br>
<br>

<center>
	<p style="font-size:15px">
	<a  href="http://www.slicetex.com"  target="_blank">Copyright  Slicetex
Electronics</a>
	</p>
</center>

</div>
</body>
</html>
```

El cual no hace más que agregar información de copyright y cerrar ciertos TAGs abiertos en **header.html**.

Visualmente, el proceso de carga de **index.shtml** es el siguiente:



Este es el proceso de carga básico de una página dinámica **.shtml**, donde lo importante es comprender que las mismas son procesadas por el servidor en busca de comandos CGI que permiten modificar el contenido de la página en tiempo real, es decir mientras el cliente ordena la carga de la pagina en su navegador.

Las paginas estaticas **.html** no tienen esta particularidad, siempre muestran el mismo contenido y el servidor no las analiza en busca de comandos CGI.

A continuación veremos cómo modificar mediante los comandos CGI y el programa del PLC el contenido de una página dinámica.

6.7 Modificar Contenido de Página desde Programa en PLC

En esta sección explicaremos como modificar el contenido que se visualiza en una página dinámica **.shtml** desde el programa cargado en el PLC, usando para ello comandos CGI.

Este mecanismo nos da una gran potencialidad para representar información en tiempo real de acuerdo a la lógica implementada por el programa en el PLC. Por ejemplo valor de entradas, señales de control, etc.

Analicemos el contenido de la página **inputs.shtml**, la cual muestra el valor de la entrada **DIN1**:

```
$!+ /header.html $

<h2>Entradas Digitales</h2>

<center>

<table id="tInputs">
  <tr>
    <th>ENTRADA</th>
    <th>VALOR</th>
    <th>LED</th>
  </tr>
  <tr>
    <td>DIN1</td>
    <td>$! PrintStr 10 $</td>
    <td><img src="$! PrintStr 11 $"></td>
  </tr>
</table>

<br>
</center>

$!+ /footer.html $
```

Los comandos CGI: **$!+ /header.html $** y **$!+ /footer.html $,** ya fueron analizados en la sección 5.6, pagina 17. Solo incluyen el contenido de encabezado y pie de página común a todas la paginas.

El código HTML **<table> … </table>** crea una tabla donde cada fila se define con los tags **<tr> … </tr>,** y dentro de cada fila el valor de cada columna se define con los tags **<td> … </td>**, mientras que para los títulos de cada columna se usa **<th> … </th>**, dando el siguiente aspecto:

ENTRADA	VALOR	LED
DIN1	0	

Si analizamos el código HTML, la parte donde se encuentran los comandos CGI es la siguiente:

```
<tr>
  <td>DIN1</td>
  <td>$! PrintStr 10 $</td>
  <td><img src="$! PrintStr 11 $"></td>
</tr>
```

El comando CGI en cuestión es "**PrintStr**" y el mismo se encuentra delimitado por los símbolos **$!** y **$**, símbolos únicos del servidor Web de los PLC de Slicetex.

Hay dos comandos CGI utilizados:

- **$! PrintStr 10 $**
- **$! PrintStr 11 $**

El comando **PrintStr** sirve para imprimir cadenas de texto provenientes del programa del PLC en la página Web y toma dos argumentos de la siguiente forma:

Sintaxis: **PrintStr Num Arg1**

- **Num:** Número de cadena a imprimir, valor entero.
- **Arg1:** Valor entero numérico de argumento, uso opcional.

La lista completa de **Comandos CGI Soportados por el Servidor** puede verla en la página 100.

Cuando un navegador (cliente web) pide la pagina **inputs.shtml**, el servidor la carga, y la procesa hasta encontrar el comando **$! PrintStr 10 $**, punto en el cual realiza las siguientes acciones:

1. El servidor web pausa la carga de la página web.
2. Obtiene los valores numéricos de **Num** y **Arg1**.
3. Procede a ejecutar código del programa del PLC
4. Activa el evento **OnWebServerPrint**, ejecutando la función **@OnWebServerPrint()**.
5. El programa del usuario en el PLC procesa desde la función **@OnWebServerPrint()** los valores de los argumentos **Num** y **Arg1** recibidos.
6. De acuerdo al valor de **Num** y **Arg1**, el usuario desde el PLC devuelve una cadena con texto, la cual reemplazará completamente la etiqueta del comando "**$! PrintStr Num Arg1 $**". Como resultado neto, el cliente web con el navegador, solo recibirá la cadena de texto devuelta por el programa del PLC, pero no el texto del comando CGI. Esta cadena se almacena temporalmente en un buffer interno.
7. El usuario finaliza la ejecución de la función **@OnWebServerPrint()**.
8. El servidor Web reanuda la carga de la pagina Web, pero reemplaza la etiqueta del comando "**$! PrintStr Num Arg1 $**" por el valor de la cadena de texto devuelta por el programa de usuario del PLC.
9. El servidor continúa con la carga del código HTML y procesamiento de los otros comandos CGI restantes en la página.

Los puntos importantes a tener en cuenta son del 5 al 7, ya que es la parte donde el programa del PLC toma control y la página web detiene su carga hasta que el programa del PLC devuelva el control al

servidor. Por lo tanto, es muy importante que retorne lo más rápido posible desde el evento **OnWebServerPrint**. De esta manera, la carga de la página web será más rápida. Evite realizar operaciones complejas que tomen mucho tiempo de procesador.

Volviendo al ejemplo en lenguaje Ladder, abramos el archivo **OnPrintEvent.p** en **StxLadder** para ver la función **@OnWebServerPrint()**. Cuando el comando CGI con el valor **$! PrintStr 10 $** sea encontrado, el siguiente código Pawn es ejecutado al activarse el evento **OnWebServerPrint**:

```
@OnWebServerPrint()
{
	//
	// Obtener e imprimir el numero de cadena solicitada por el servidor.
	//

	switch(WebServerGetPrintNum())
	{
		//
		// Procesar cadenas relacionadas a inputs.shtml
		//

		case 10:
			WebServerPrintStr("%d", DinValue(DIN1))
		case 11:
			WebServerPrintStr("/l%d.gif", DinValue(DIN1))

		//
		// …
		//
	}
}
```

La función **WebServerGetPrintNum()** obtiene el último valor del argumento **Num** recibido desde un comando CGI **PrintNum**. Dicho valor lo usamos para la sentencia **switch()** de Pawn.

Nota: Puede usar también la función **WebServerGetPrintArg1()** para obtener el valor de Arg1.

Luego procesamos el valor **Num** recibido, para determinar el número de cadena que el Servidor Web solicita imprimir:

```
		case 10:
			WebServerPrintStr("%d", DinValue(DIN1))
```

Si **Num == 10**, se ejecuta la función **WebServerPrintStr()**. Este último nos permite imprimir una cadena de texto con formato en la página web.

En este caso se imprime la cadena "**%d**", la cual obtiene el valor entero de la entrada digital DIN1 a través de la función **DinValue(DIN1)**. Ver documentación de función para información de uso.

Una vez que procesamos el valor de **Num**, se recomienda retornar del evento así el Servidor Web puede continuar la carga de la pagina web.

Entonces, si DIN1 es 0, la cadena resultante es “0” y en la página Web queda:

```
    <td>0</td>
```

ENTRADA	VALOR	LED
DIN1	0	

Si DIN1 es 1, la cadena resultante es “1” y en la página Web queda:

```
    <td>1</td>
```

ENTRADA	VALOR	LED
DIN1	1	

Notar cómo se reemplaza el texto del comando **$! PrintStr 10 $** por el valor de la cadena retornada por el programa del PLC en la pagina dinámica.

Volviendo al código Pawn, cuando el comando CGI sea **$! PrintStr 11 $**, el siguiente código se procesará:

```
            case 11:
                 WebServerPrintStr("/l%d.gif", DinValue(DIN1))
```

En este caso si Num=11, la función **WebServerPrintStr()** imprime la siguiente cadena:

“/l%d.gif”

La cual depende del valor reemplazado en el código de formato %d, que toma su valor del puerto DIN1. En otras palabras, puede retornar: “**/l0.gif**” o “**/l1.gif**”.

Si nos fijamos en el **Explorador de Proyecto**, tenemos dos archivos de imágenes GIF cargados:

Archivo: l0.gif

Archivo: l1.gif

Ambas imágenes representan un LED apagado y un LED encendido. Ideal para representar el estado de una entrada digital como DIN1.

Entonces, si DIN1 es 0, la cadena resultante es "**/l0.gif**" y en la página Web queda:

```
<td><img src="/l0.gif"></td>
```

ENTRADA	VALOR	LED
DIN1	0	

Si DIN1 es 1, la cadena resultante es "**/l1.gif**" y en la página Web queda:

```
<td><img src="/l1.gif"></td>
```

ENTRADA	VALOR	LED
DIN1	1	

El código HMTL **<img scr="**url-imagen**">** le dice al navegador "cargar una imagen localizada en url-imagen", en este caso especificamos el camino hasta el archivo gif, por ejemplo **/l1.gif**.

```
<img src="/l1.gif">
```

El navegador al encontrar ese tag HTML, hace una petición extra al Servidor Web para descargar y mostrar la imagen.

Visualmente, este es el proceso cuando el servidor encuentra los comandos CGI:

Activa evento y ejecuta el evento **@OnWebServerPrint()**

Se procesa argumentos **Num** y **Arg1**, para devolver la cadena de texto solicitada por el servidor.

```
<tr>
  <td>DIN1</td>
  <td>$! PrintStr 10 $</td>
  <td><img src="$! PrintStr 11 $"></td>
</tr>
```

6.8 Procesar Formularios Web y Recibir Datos del Navegador

En secciones previas vimos como modificar contenido dinámicamente en una página web para mostrar información que puede variar con el tiempo, como entradas, variables, etc.

Ahora veremos cómo recibir datos desde un Navegador Web, es decir desde el cliente al servidor. Esto nos permitirá interactuar con el servidor para pasar valores y controlar el funcionamiento del PLC remotamente.

El método utilizado para este fin en los Servidores Web de los PLC de Slicetex Electronics, es a través de formularios web (form) usando el método "**POST**".

La pagina **outputs.shtml** del ejemplo permite controlar la salida digital **DOUT1** del PLC. La misma consta de un formulario (con una tabla) y entradas de control tipo "**radio**" (permiten selección un valor entre varias opciones). Finalmente, un botón "**Actualizar**" que envía desde el navegador los valores seleccionado por los controles "**radio**".

Cuando en el navegador se haga click en "**Actualizar**", se enviarán los valores seleccionados en el formulario al Servidor Web.

Esto desencadena las siguientes acciones en el Servidor Web:

1. Activa el evento **OnWebServerPost**, ejecutando la función **@OnWebServerPost()**.
2. El programa del usuario en el PLC procesa el evento desde el archivo **OnPostEvent.p** comprobando los valores del formulario recibidos.
3. De acuerdo a valores del formulario recibidos, el usuario desde el PLC ejecuta una acción, como por ejemplo actualizar el valor de una salida digital.
4. El usuario finaliza la ejecución del evento **OnWebServerPost** retornando de la función **@OnWebServerPost()**.

Analicemos el contenido de la página **outputs.shtml**:

```
$!+ /header.html $

<center>
<h2>Salidas Digitales</h2>

<form action="/outputs.shtml" method="post">
<input type="hidden" name="Form" value="Outputs">
      <table id="tOutputs">
        <tr>
          <th>SALIDA</th>
          <th>VALOR</th>
        </tr>
        <tr>
          <td>DOUT1</td>
          <td>
            <input type="radio" name="DO1" value="0" $! PrintStr 20 $> OFF
            <br>
            <input type="radio" name="DO1" value="1" $! PrintStr 21 $> ON
          </td>
        </tr>
      </table>
      <br>
      <input type="submit" value="Actualizar" class="update">
      <br>
</form>

<br>
</center>

$!+ /footer.html $
```

Dejemos de lado los comandos CGI delimitados por los caracteres **$!** y **$**, y comprendamos el código.

Un formulario HTML (form) se define como:

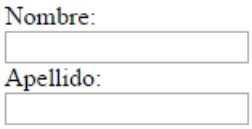
```
<form>
…
elementos form
…
</form>
```

El elemento **<input>** es el más importante de los elementos **form**, ya que el mismo nos permite de acuerdo a su tipo definir entradas para texto, controles radio, botones, etc.

Ejemplo:

```
<form>
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
</form>
```

Produce la siguiente salida en un navegador:



Este formulario tiene solo dos entradas “input” del tipo “text”. Ambas nos permiten introducir texto.

```
<input type="text" name="Nombre">
```

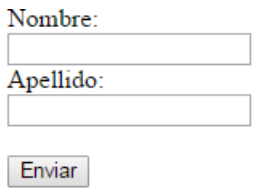
```
<input type="text" name="Apellido">
```

El atributo “**name**” nos permite agregar un nombre a las “**input**” para luego poder identificarlas cuando recibamos los valores en el Servidor Web.

Sin embargo, nos falta el botón para enviar los datos al Servidor Web. Esto se puede agregar con otro “input” del tipo “submit”.

```
<form>
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

Resultando la siguiente visualización en el navegador Web:



Ya es posible enviar los datos al Servidor Web haciendo click en el botón “Enviar”, sin embargo para el Servidor Web del PLC necesitamos agregar lo siguiente en cada formulario creado:

```
<form action="/pagina_destino.html" method="post">
<input type="hidden" name="Form" value="Nombres">
  Nombre:<br>
  <input type="text" name="Nombre">
  <br>
  Apellido:<br>
  <input type="text" name="Apellido">
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

Agregamos a <form> los siguientes atributos:

- **action="/pagina_destino.html":** Es de uso opcional, pero nos permite especificar una página de destino que se cargará cuando se haga click en el botón "**submit**". Puede ser la misma página del formulario u otra diferente. Dinámica o estática.

- **method="post"**: Este es el método a utilizar por el navegador cuando se envíen los datos del formulario al Servidor Web. Utilizamos el método POST.

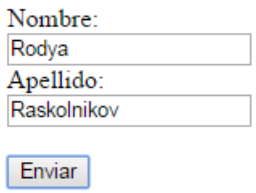
Finalmente, agregamos otro "**input**" al formulario para identificarlo de otros formularios en diferentes páginas web del Servidor.

```
<input type="hidden" name="Form" value="Nombre_Formulario">
```

Donde:

- **type="hidden"**: Hace que esta entrada del formulario no sea visible en el navegador.
- **name="Form"**: Nombre de la "**input**", la llamamos **Form** por convención.
- **value="Nombre_Formulario"**: Nombre del formulario, puede ser cualquiera pero único para cada formulario del servidor.

En los servidores de Slicetex, el valor de "**name**" de cada "**input**" no puede contener espacios, ni caracteres especiales. Solo texto o números, y es buena práctica no hacerlos demasiados extensos. También no pueden repetirse los valores de "**names**" de los "**input**" de un mismo formulario.



Ahora cuando hagamos click en el botón "**Enviar**" del formulario de la imagen anterior, se envía la siguiente información al Servidor Web en forma de campos (fields), los cuales siempre están de a pares con sus valores (values) en el siguiente orden:

4. Form = Nombres
5. Nombre = Rodya
6. Apellido = Raskolnikov

Cada dato esta en el formato "**Field = Value**", dentro del Servidor Web podemos obtener los valores de cada campo. Para ello debemos especificar el nombre del campo, por ejemplo "**Nombre**" y una función nos devolverá la cadena "**Rodya**", obteniendo de esta forma dicho valor.

Es evidente, que podemos de esta forma enviar números, texto, etc remotamente al servidor.

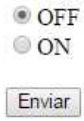
Cuando el Servidor Web recibe los datos de un formulario web con el método POST, genera un evento del tipo **OnWebServerPost**. Allí podemos procesar los datos recibidos.

Si queremos utilizar un formulario con un control tipo "**radio**" como el del ejemplo, podemos escribir el siguiente código HTML:

```
<form action="/outputs.shtml" method="post>
<input type="hidden" name="Form" value="Outputs">
  <input type="radio" name="DO1" value="0" checked>OFF
  <br>
  <input type="radio" name="DO1" value="1">ON
  <br><br>
  <input type="submit" value="Enviar">
</form>
```

En color rojo, pusimos el código importante a observar. En los elementos "**input**" utilizamos el atributo "**type=radio**", usando el mismo "**name=DO1**" para cada input ya que este tipo de control solo envía una opción de varias entradas. Cada entrada "input" tiene un "**value**" diferente, en este caso "1" o "0". Finalmente el atributo "**checked**" permite seleccionar el radio por defecto.

En un navegador resulta la siguiente visualización:



Si hacemos click en "Enviar", los siguientes campos se envían:

3. Form = Outputs
4. DO1 = 0

Se envía el campo DO1=0, porque el valor seleccionado es "OFF".

Volvamos al código de **outputs.shtml**, la parte del código del formulario la marcamos en rojo**:**

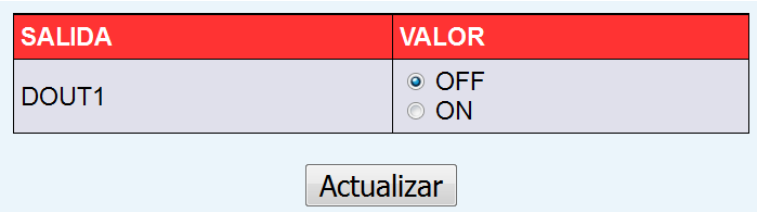
```
<form action="/outputs.shtml" method="post">
<input type="hidden" name="Form" value="Outputs">
      <table id="tOutputs">
        <tr>
          <th>SALIDA</th>
          <th>VALOR</th>
        </tr>
        <tr>
          <td>DOUT1</td>
          <td>
            <input type="radio" name="DO1" value="0" $! PrintStr 20 $> OFF
            <br>
            <input type="radio" name="DO1" value="1" $! PrintStr 21 $> ON
          </td>
        </tr>
      </table>
      <br>
      <input type="submit" value="Actualizar" class="update">
      <br>
</form>
```

Como podemos observar, es muy similar a los formularios que vimos de ejemplo, solo que esta insertado dentro de una tabla HTML (<table>) para darle una mejor visualización.

Por otro lado hay dos comandos CGI adentro del código del formulario:

- $! PrintStr 20 $
- $! PrintStr 21 $

Ambos comandos CGI son procesados en el evento "**@OnWebServerPrint()**" cuando se carga la pagina, y solo insertan el texto "**checked**" de acuerdo al valor de la salida digital DOUT1, permitiendo seleccionar el radio por default según corresponda (OFF u ON) automáticamente. Esto lo veremos más adelante.



Si hacemos click en el botón "**Actualizar**" los siguientes campos se envían al Servidor Web:

3. Form = Outputs
4. DO1 = 0

Cuando el Servidor Web recibe los datos del formulario web con el método POST, genera un evento del tipo **OnWebServerPost**. Se ejecuta la funcion "**@OnWebServerPost.sld()**" en archivo **OnPostEvent.p** e inicialmente se ejecuta el siguiente código Pawn:

```
@OnWebServerPost()
{
	//
	// Determinar el formulario de donde recibimos los datos.
	//

	if(WebServerCheckFieldValueStr("Form", "Outputs") == 1)
	{
		// Procesar formulario "Outputs" en outputs.shtml.
		ProcessFormOutputs()

		// Retornar de evento inmediatamente.
		return
	}
}
```

La función **WebServerCheckFieldValueStr()** busca dentro de los datos recibidos desde el formulario, el campo Field = "**Form**" y si tiene el valor Value = "**Outputs**", retorna 1. Ejecutando la función **ProcessFormOutputs()** que procesa el formulario "**Outputs**". Esto nos permite discriminar entre diferentes formularios.

La función **ProcessFormOutputs()** es definida por el usuario y tiene el siguiente código Pawn:

```
ProcessFormOutputs()
{
	//
	// Activar/Desactivar salida DOUT1 segun valor del campo DO1 recibido.
	//

	// Obtener el valor Bool del campo DO1.
	if(WebServerGetFieldValueBool("DO1"))
	{
		DoutSetOn(DOUT1)
	}
	else
	{
		DoutSetOff(DOUT1)
	}
}
```

Aquí se ejecuta la función **WebServerGetFieldValueBool()** que busca el campo **Field = DO1** y obtiene su valor del tipo **Bool**, es decir “1” o “0”.

De acuerdo al valor retornado por **WebServerGetFieldValueBool()** escribimos un “0” o “1” en la salida digital DOUT1 con las funciones de acceso a salidas digitales (DoutSetOn() y DoutSetOff()).

Se recomienda retornar lo antes posible del evento, así el PLC puede seguir procesando otras operaciones.

Hasta aquí vimos como controlar y enviar datos remotamente al PLC desde un Navegador Web.

Los formularios son un método muy potente para operaciones de control, aconsejamos investigar más sobre los mismos en el siguiente link:

http://www.w3schools.com/html/html_forms.asp

Luego de hacer click en el botón “Actualizar”, el formulario especifica `action="/outputs.shtml"`, esto quiere decir que el navegador nuevamente pedirá cargar la pagina “**outputs.shtml**”, provocando que el Servidor Web la procese nuevamente. Y como teníamos dos comandos CGI del tipo **PrintStr** se ejecutará dos veces la función **@OnWebServerPrint()**, porque el evento **OnWebServerPrint** será disparado dos veces.

El siguiente código del archivo "**OnPrintEvent.p**" será ejecutado de acuerdo a los argumentos del comando CGI del tipo **PrintStr**:

```
@OnWebServerPrint()
{
	//
	// Obtener e imprimir el numero de cadena solicitada por el servidor.
	//

	switch(WebServerGetPrintNum())
	{
		//
		// …
		//

		//
		// Procesar cadenas relacionadas a outputs.shtml
		//

		case 20:
			if(DoutValue(DOUT1) == 0)
			{
				WebServerPrintStr("checked")
			}
		case 21:
			if(DoutValue(DOUT1) == 1)
			{
				WebServerPrintStr("checked")
			}
	}
}
```

Cuando **Num** sea igual a 20, si el valor de **DOUT1** es "0" se devuelve la cadena "**checked**" a la página Web. De lo contrario, no se devuelve nada, una cadena vacía.

Cuando **Num** sea igual a 21, si el valor de **DOUT1** es "1" se devuelve la cadena "**checked**" a la página Web. De lo contrario, no se devuelve nada, una cadena vacía.

El valor "**checked**" permite especificar un atributo para seleccionar el "**radio**" correcto en el formulario. De esta forma, indicamos el valor actual de la salida **DOUT1** indirectamente.

El comando CGI del tipo **PrintStr** lo vimos en la sección **Modificar Contenido de Página desde Programa en PLC** de la página 47.

Una nota final, si los datos enviados por el formulario al PLC son excesivos, fuera del límite soportado, el PLC carga la página **413.html** mostrándole al usuario el error.

6.9 Fin del Ejemplo

Estas son las nociones básicas para usar el Servidor Web. Puede usar este ejemplo y otros disponibles en nuestro sitio de Internet como base para sus proyectos.

Lea las descripciones de cada componente desde el entorno StxLadder antes de usarlos, para mejorar la comprensión de los mismos.

Recomendamos altamente que siga leyendo las siguientes secciones:

- Referencia de Funciones en Lenguaje Pawn, página 78
- Comandos CGI Soportados por el Servidor, página 100
- Opciones de Configuración del Servidor, página 103
- Recomendaciones de Diseño Web, página 107

7 Referencia de Componentes en Lenguaje Ladder

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

Recomendamos leer primero el **Ejemplo de Uso Inicial en Lenguaje Ladder** en página 8.

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

- Componentes para configurar el servidor y la conexión.
- Componentes para activar eventos.
- Componentes para imprimir datos de páginas dinámicas y modificar su contenido.
- Componentes para obtener datos provenientes de formularios web.

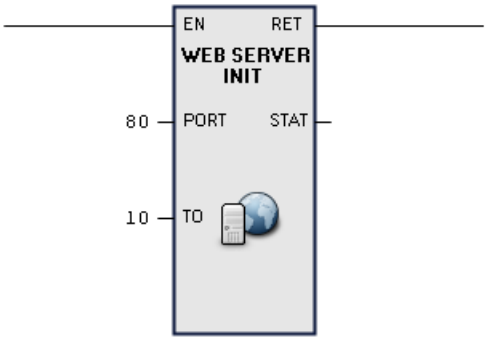
Importante:

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

WEB SERVER - INIT

Agrega un componente que permite inicializar y activar el **Servidor Web** del PLC. El Servidor Web aceptará conexiones a través de la interfaz **Ethernet** desde clientes o navegadores Web remotos (ejemplo: Firefox, Chrome, Explorer, Edge, Opera, etc).



Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- En el puerto "**PORT**" se especifica el puerto TCP donde el Servidor Web escuchará conexiones remotas de clientes. En general se utiliza el puerto 80.
- En el puerto "**TO**" se especifica el tiempo en segundos que deben transcurrir de inactividad para que un cliente sea desconectado del servidor. Valor recomendado 10 segundos. Si el cliente no realiza transacciones antes de transcurrido el tiempo **TO**, será desconectado automáticamente.

Salidas:

- La salida "**RET**" del componente es verdadera si el servidor pudo configurarse con éxito.
- La salida "**STAT**" del componente, es de uso opcional, pero permite determinar si el **servidor** pudo ser activado. Es posible especificar una variable que almacenará un código de error. Si el retorno es cero, la operación es exitosa, si es negativo se produjo un error de activación.
    - Retorno 0: Operación exitosa.
    - Retorno -1: Error, el servidor ya se encuentra inicializado.
    - Retorno -2: Error, falla en inicialización, memoria insuficiente.
    - Retorno -10: Error, el valor PORT es inválido.

Importante:

- El servidor es activado luego de llamar exitosamente a este componente.
- El servidor admite un número máximo de clientes remotos conectados simultáneamente, si se excede dicho límite, serán desconectados abortando la conexión.

7.2 Componentes Para Eventos

WEB SERVER - SET PRINT EVENT

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

El evento **OnWebServerPrint** será llamado cuando deba imprimirse una cadena en una página Web dinámica.

El evento se ejecuta cuando se encuentra el comando CGI "**PrintStr**" en la página Web dinámica.

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 **OnWebServerPrint** es un mecanismo muy útil para mostrar información dinámica en páginas Web del servidor.

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.

WEB SERVER - CLEAR PRINT EVENT

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

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**".

WEB SERVER - SET FUNCTION CALL EVENT

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

El evento **OnWebServerFuncCall** será llamado cuando deba llamarse una función personalizada desde una página Web dinámica.

El evento se ejecuta cuando se encuentra el comando CGI "**FuncCall**" en la página Web dinámica.

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 **OnWebServerFuncCall** es un mecanismo muy útil para ejecutar funciones o lógicas en páginas Web dinámicas del servidor.

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.

WEB SERVER - CLEAR FUNC CALL EVENT

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

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**".

WEB SERVER - SET POST EVENT

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

El evento **OnWebServerPost** será llamado cuando sean recibidos datos desde formularios en el servidor a través de un método POST.

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 **OnWebServerPost** es un mecanismo muy útil para obtener datos de control o información producidos por un visitante de la pagina Web (por ejemplo un valor entero, una cadena, etc). A partir de los datos recibidos, se puede realizar una lógica interna en el PLC.

Los datos recibidos están agrupados por en campos (fields) en una cadena llamada **Query String**. Puede utilizar los componentes "**GET FIELD XXX**" para obtener los valores de dichos campos.

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.

WEB SERVER - CLEAR POST EVENT

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

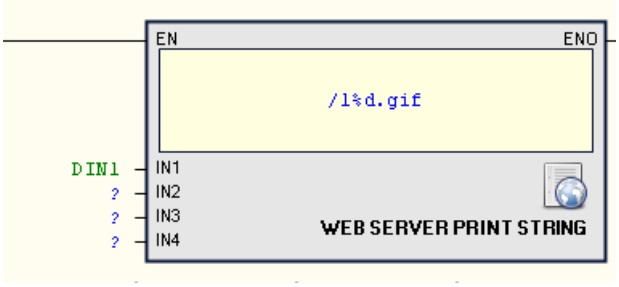
El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

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

7.3 Componentes Para Obtener Datos e Imprimir en Páginas Dinámicas

WEB SERVER - PRINT STRING

Agrega un componente que permite imprimir mensajes de texto con formato en páginas Web dinámicas. La impresión con formato permite mezclar texto con valores de variables en la impresión de mensajes.



Si busca imprimir el valor de una variable, este es el componente indicado.

La impresión del texto debe realizarse desde el evento "**OnWebServerPrint**", identificando antes el número de cadena solicitada por el servidor a imprimir.

La cadena se imprime en la posición del comando CGI "**PrintStr**" del la pagina Web dinámica.

El texto a imprimir se especifica en las propiedades del componente.

Es posible especificar en el componente que antes de imprimir la cadena, se compruebe el número de cadena a imprimir mediante la opción "***Comprobar número de cadena***". Esto habilita el ingreso del valor "**Numero de cadena**", en este caso si el numero de cadena solicitado por el servidor no coincide, no se imprime ninguna cadena y la salida **ENO** del componente es 0.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- Existen 4 entradas más, cuyo uso es opcional y permiten especificar las variables a leer e imprimir en la cadena.

  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 una copia del valor recibido en el puerto "**EN**". En el caso de estar habilitada la opción "***Comprobar número de cadena***", la salida es 0 si el numero de cadena solicitada por el servidor no coincide con la especificada por el componente.

Formato:

Para imprimir mensajes con formato, es necesario especificar "***códigos de formato***" en el campo "**Mensaje**" 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 al mensaje a imprimir.

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	)		

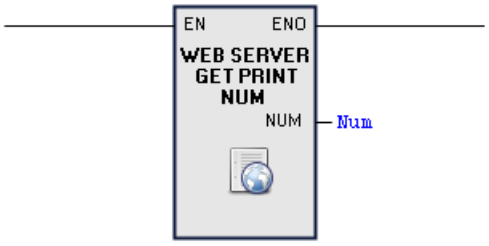
En las tablas, el término "Argumento" se refiere al valor del puerto (**IN1**, **IN2**, etc).

Notas / Consejos:

- No se pueden imprimir cadenas de texto vacías o que solo imprimen espacios. En dicho caso, se crea un error de advertencia al intentar compilar.
- Puede utilizar el componente **Get Print Num** para obtener el número de cadena a imprimir solicitada por el server.
- Si usted es un programador en lenguaje C, el uso de este componente le resultara similar a la función "**printf()"** de dicho lenguaje.

WEB SERVER - GET PRINT NUM

Agrega un componente que obtiene el número de cadena para imprimir requerida por el servidor Web al cargar una página dinámica.



El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**PrintStr x**" en la página Web dinámica. Donde "**x**" es el numero de cadena a imprimir.

En general, este componente se llama desde el evento "**OnWebServerPrint**" para obtener el número de cadena a imprimir y en función al valor obtenido, se imprime la cadena solicitada con el componente **Web Server Print Str**.

Entradas:

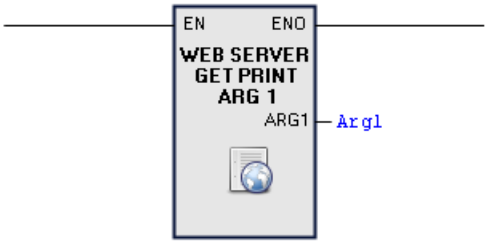
- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**NUM**" se retorna el numero de cadena a imprimir en una variable del tipo **Int32**.

WEB SERVER - GET PRINT ARG1

Agrega un componente que obtiene el argumento numero 1 (**Arg1**) asociado a una cadena requerida para imprimir por el servidor Web al cargar una página dinámica.



El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**PrintStr x Arg1**" en la página Web dinámica. Donde "**Arg1**" es el argumento (valor entero) asociado al número de la cadena "**x**" requerida para imprimir.

En general, este componente se llama desde el evento "**OnWebServerPrint**" para obtener el argumento de la cadena a imprimir y en función al valor obtenido, se realiza alguna acción o se ejecuta una lógica en particular.

Entradas:

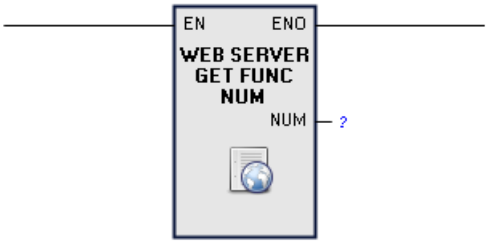
- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**ARG1**" se retorna el argumento recibido de la cadena a imprimir en una variable del tipo **Int32**.

WEB SERVER - GET FUNC CALL NUM

Agrega un componente que obtiene el número de llamada a función requerida por el servidor Web al cargar una página dinámica.



El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**FuncCall x**" en la página Web dinámica. Donde "**x**" es el numero de función a llamar.

En general, este componente se llama desde el evento "**OnWebServerFuncCall**" para obtener el número de función a llamar y en función del valor obtenido se ejecuta alguna lógica dentro del PLC.

Entradas:

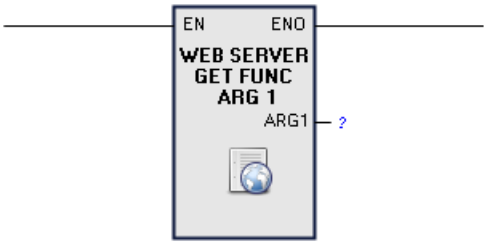
- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**NUM**" se retorna el numero de llamada a función en una variable del tipo **Int32**.

WEB SERVER - GET FUNC CALL ARG1

Agrega un componente que obtiene el argumento numero 1 (**Arg1**) asociado a una llamada a funcion requerida por el servidor Web al cargar una página dinámica.



El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**FuncCall x Arg1**" en la página Web dinámica. Donde "**Arg1**" es el argumento (valor entero) asociado al número llamada de funcion "**x**" requerida por el servidor.

En general, este componente se llama desde el evento "**OnWebServerFuncCall**" para obtener el argumento, y en función al valor obtenido, se realiza alguna acción o se ejecuta una lógica en particular.

Entradas:

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

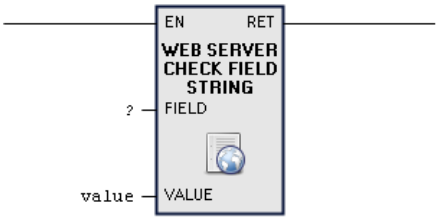
Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**ARG1**" se retorna el argumento recibido de la llamada a función en una variable del tipo **Int32**.

7.4 Componentes Para Obtener Datos de Formularios Web

WEB SERVER -CHECK FIELD VALUE STRING

Agrega un componente que comprueba si existe un valor de una cadena (**String**) determinado en un campo (**field**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener los valores de los campos (**fields**) ingresados por el usuario.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **FIELD**: Nombre del campo donde se compara el valor **VALUE**.
- **VALUE**: Constante o variable tipo **String** para comparar con el valor del campo **FIELD** recibido.

Salidas:

- El valor del puerto de salida "**RET**" es 1 si el valor de la cadena **VALUE** coincide con el valor recibido en el campo **FIELD** asociado. El valor es 0 si no coincide o si se produjo un error.

Importante:

- El nombre del campo (**field**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario.

Consejos:

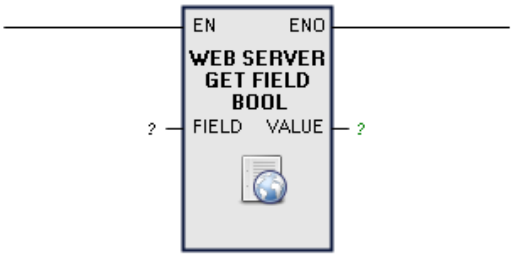
- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
- Puede utilizar este componente para identificar formularios específicos o comprobar contraseñas. Por ejemplo, si envió un formulario con el campo **Form**=**Inputs**, puede especificar el componente con las entradas **FIELD**=Form y **VALUE**=Inputs. Si al recibirse los valores del formulario mediante un método POST, este componente entregará **RET=1** si los valores coinciden. De esta manera, puede obtener los siguientes datos del formulario y diferenciarlo de otros formularios recibidos. Consulte ejemplos.

Notas:

- El valor del campo se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

WEB SERVER - GET FIELD VALUE BOOL

Agrega un componente que obtiene el valor booleano (**Bool**) de un campo (**field**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener los valores de los campos (**fields**) ingresados por el usuario.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **FIELD**: Nombre del campo a buscar.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**VALUE**" se retorna el valor del campo en una variable del tipo **Bool**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.

Importante:

- El nombre del campo (**field**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario.

Consejos:

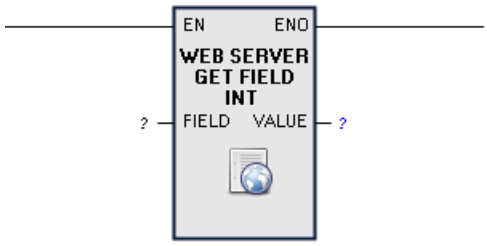
- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.

Notas:

- El valor del campo se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

WEB SERVER - GET FIELD VALUE INT

Agrega un componente que obtiene el valor entero (**Int32**) de un campo (**field**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener los valores de los campos (**fields**) ingresados por el usuario.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **FIELD**: Nombre del campo a buscar.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**VALUE**" se retorna el valor del campo en una variable del tipo **Int32**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.

Importante:

- El nombre del campo (**field**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario.

Consejos:

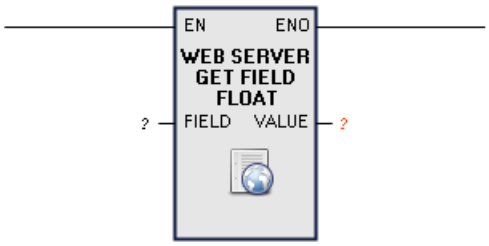
- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.

Notas:

- El valor del campo se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

WEB SERVER - GET FIELD VALUE FLOAT

Agrega un componente que obtiene el valor punto flotante (**Float**) de un campo (**field**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener los valores de los campos (**fields**) ingresados por el usuario.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **FIELD**: Nombre del campo a buscar.

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**VALUE**" se retorna el valor del campo en una variable del tipo **Float**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.

Importante:

- El nombre del campo (**field**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario.

Consejos:

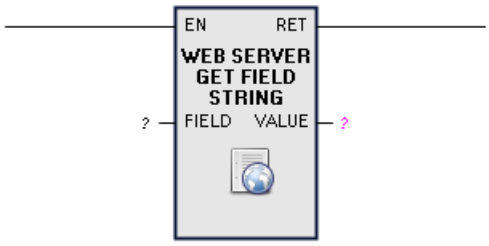
- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.

Notas:

- El valor del campo se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

WEB SERVER - GET FIELD VALUE STRING

Agrega un componente que obtiene el valor de una cadena (**String**) de un campo (**field**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener los valores de los campos (**fields**) ingresados por el usuario.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- **FIELD**: Nombre del campo a buscar.

Salidas:

- El valor del puerto de salida "**RET**" es 1 si el valor de la cadena fue obtenida exitosamente. El valor es 0 si no fue encontrada o se produjo un error.
- En el puerto "**VALUE**" se retorna el valor del campo en una variable del tipo **String**. La longitud máxima de la cadena retornada es de 60 caracteres. Si la cadena que se pasa para el retorno, es menor a la longitud de la cadena encontrada, se trunca el contenido. La cadena resultante siempre tiene el carácter NULL como terminador al final.

Importante:

- El nombre del campo (**field**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario.

Consejos:

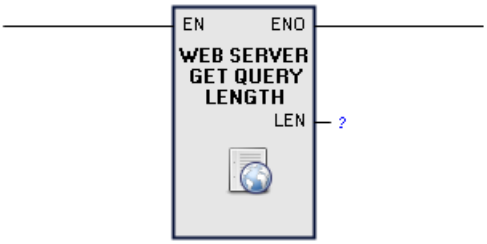
- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.

Notas:

- El valor del campo se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

WEB SERVER - GET QUERY LENGTH

Agrega un componente que obtiene la longitud en caracteres de la ultima cadena-query (**Query String**) proveniente de los datos enviados por un formulario POST en una página Web.



Si una página Web fue cargada como resultado de un método POST proveniente de una formulario Web, es posible obtener la longitud del **Query String**.

Este componente busca el campo del último valor recibido por un método POST en el servidor.

Entradas:

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

Salidas:

- El valor del puerto de salida "**ENO**" es una copia del valor recibido en el puerto "**EN**".
- En el puerto "**LEN**" se retorna la longitud en caracteres del último **Query-String** recibido en una variable del tipo **Int32**.

Consejos:

- Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor del Query-String cargado por el cliente Web.
- Este componente es útil para situaciones donde se debe realizar una prueba de recepción o depuración de páginas Web.

8 Referencia de Funciones en Lenguaje Pawn

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

Recomendamos leer primero el **Ejemplo de Uso Inicial en Lenguaje Pawn** en página 34.

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

- Funciones para configurar el servidor y la conexión.
- Funciones para activar eventos.
- Funciones para imprimir datos de páginas dinámicas y modificar su contenido.
- Funciones para obtener datos provenientes de formularios web.

En la página siguiente se describen dichas funciones en detalle.

8.1 Funciones de Configuración

WebServerInit(ServerPort, TimeoutIdle, Opt): Inicializa y configura los parámetros para que el PLC pueda funcionar como un Servidor Web.		
Argumentos	**Tipo**	**Descripción**
ServerPort	E	Puerto TCP donde el Servidor Web escuchará conexiones. Por defecto 80.
TimeoutIdle	E	Tiempo en segundos de timeout antes de desconectar un cliente inactivo que se encuentre ocupando una conexión TCP. Máximo 125 segundos. Por defecto 10 segundos.
Opt	E	Opciones de configuración. Por defecto 0.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el servidor ya se encuentra inicializado.
-2	S	Error, falla en inicialización, memoria insuficiente.
-10	S	Error, puerto TCP inválido.
Notas		**Descripción**
1		Esta función debe llamarse una vez, antes de usar cualquier otra función del Servidor Web.
2		Si selecciona otro puerto, por ejemplo el numero **88**, para acceder con un navegador deberá escribir el numero de puerto separado por ":" en la barra de direcciones, por ejemplo: http://192.168.1.81:88

Ejemplo 1:

```
// Inicializar Web Server.
// Puerto: 80

if(WebServerInit(80) < 0)
{
      // Error, pausar programa en este punto.
      while(true)
      {
            DelayMS(2000)
            LedToggle()
      }
}
```

Ejemplo 2:

```
// Inicializar Web Server.
// Puerto: 80
// Timeout Idle: 10 segundos.

WebServerInit(80, 10)
```

8.2 Funciones Para Eventos

WebServerSetPrintEvent(): Activa el evento @OnWebServerPrint(). El evento **OnWebServerPrint** será llamado cuando deba imprimirse una cadena en una página Web dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo activarse.
Notas		**Descripción**
1		El evento se ejecuta cuando se encuentra el comando CGI "**PrintStr**" en la página Web dinámica.
2		El evento **OnWebServerPrint** es un mecanismo muy útil para mostrar información dinámica en páginas Web del servidor.
3		Puede imprimir cadenas en páginas Web, usando la función **WebServerPrintStr()**.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento  @OnWebServerPrint() para imprimir
   // cadenas en paginas dinamicas.
   WebServerSetPrintEvent()

   // . . .
}

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

WebServerClrPrintEvent(): Desactiva el evento @OnWebServerPrint().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo desactivarse.
Notas		**Descripción**
-		-

Ejemplo 1:

```
PlcMain()
{
   // Desactiva evento @OnWebServerPrint
   WebServerClrPrintEvent()

   // . . .
}
```

WebServerSetFuncCallEvent(): Activa el evento @OnWebServerFuncCall(). El evento **OnWebServerFuncCall** será llamado cuando deba llamarse una función personalizada en una página Web dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo activarse.
Notas		**Descripción**
1		El evento se ejecuta cuando se encuentra el comando CGI "**FuncCall**" en la página Web dinámica.
2		El evento **OnWebServerFuncCall** es un mecanismo muy útil para ejecutar funciones o lógicas en páginas Web dinámicas del servidor.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento @OnWebServerFuncCall() para ejecutar funciones
   // desde paginas dinamicas.
   WebServerSetFuncCallEvent()

   // . . .
}

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

WebServerClrFuncCallEvent(): Desactiva el evento @OnWebServerFuncCall().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo desactivarse.
Notas		**Descripción**
-		-

Ejemplo 1:

```
PlcMain()
{
   // Desactiva evento @OnWebServerFuncCall
   WebServerClrFuncCallEvent()

   // . . .
}
```

WebServerSetPostEvent(): Activa el evento @OnWebServerPost(). El evento **OnWebServerPost** será llamado será llamado cuando sean recibidos datos desde formularios en el servidor a través de un método POST.

Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo activarse.
Notas		**Descripción**
1		El evento **OnWebServerPost** es un mecanismo muy útil para obtener datos de control o información producidos por un visitante de la pagina Web (por ejemplo un valor entero, una cadena, etc). A partir de los datos recibidos, se puede realizar una lógica interna en el PLC.
2		Los datos recibidos están agrupados por en campos (fields) en una cadena llamada **Query String**. Puede utilizar las funciones **WebServerGetFieldXXX()** para obtener los valores de dichos campos.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento  @OnWebServerPost() para procesar datos
   // recibidos de formularios..
   WebServerSetPostEvent()

   // . . .
}

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

WebServerClrPostEvent(): Desactiva el evento @OnWebServerFuncCall().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo desactivarse.
Notas		**Descripción**
-		-

Ejemplo 1:

```
PlcMain()
{
   // Desactiva evento @OnWebServerPost
   WebServerClrPostEvent()

   // . . .
}
```

8.3 Funciones Para Obtener Datos e Imprimir en Páginas Dinámicas

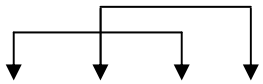
WebServerPrintStr(FormatString[], …): Imprime una cadena de texto con formato en una página web dinámica.		
Argumentos	**Tipo**	**Descripción**
FormatString	E	Cadena a imprimir, con o sin formato. [4].
…	E	Lista de argumentos variables, de acuerdo a si se especifica formato. [5].
Retorno	**Tipo**	**Descripción**
>= 0	S	Operación exitosa. El retorno es la cantidad de caracteres impresos.
-1	S	Error, array/string con dirección inválida.
-2	S	Error de conversión.
Notas		**Descripción**
1		La cadena se imprime en la posición del comando CGI "**PrintStr**" del la pagina Web dinámica.
2		La impresión del texto debe realizarse desde el evento "**OnWebServerPrint**", identificando antes el número de cadena a imprimir solicitada por el servidor.
3		La cadena es transferida a un buffer interno y luego el servidor la imprime.
4		Máximo tamaño: 100 caracteres incluyendo el carácter terminador '0'.
5		La cadena resultante total (formato + argumentos) no debe superar los 200 caracteres.

La función **WebServerPrintStr()** permite imprimir cadenas con formato de forma simple en una página web dinámica. Su utilidad es enorme y resulta muy útil para mostrar información variable en una página web dinámica. 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. Dentro de los PLC de Slicetex Electronics, hay otras funciones similares, como **nLcdPrintf()**, **LcdPrintf()**, etc. Por lo tanto, si conoce aquellas funciones, encontrará familiar el uso de **WebServerPrintStr()**.

El primer argumento de la función, es la cadena con formato que será explicada más adelante. 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 imprimir, puede o no pueden existir argumentos.

Un ejemplo básico puede ser:



```
WebServerPrintStr("Promedio %f - %d", 5.5, 9)
```

Imprimirá en la página:

```
Promedio 5.5 - 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 otro 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 (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	ŧ	ŧ						
%X	255	F	F						
%s	"HOLA"	H	O	L	A				
%c	66	B							
%c	'B'	B							
%s	0	(	n	u	l	l	)		

Ejemplos prácticos:

Remítase a los ejemplos de la nota de aplicación **AN030** para ejemplos completos.

```
// Imprimir cadena.
WebServerPrintStr("Nombre: %s", "Rodion")
```

```
Nombre: Rodion
```

```
// Imprimir numero.
WebServerPrintStr("Edad: %d", 23)
```

```
Edad: 23
```

```
// Imprimir decimal.
WebServerPrintStr("Tanque: %f %%", 60.37)
```

```
Tanque: 60.37 %
```

```
// Imprimir Variables y Cadenas
new Nombre[] = "Dunia"
new Apellido[] = "Raskolnikova"
new Edad = 25

WebServerPrintStr("Nombre: %s, Apellido: %s, Edad: %d", Nombre,Apellido,Edad)
```

```
Nombre: Dunia, Apellido: Raskolnikova, Edad: 25
```

```
// Imprimir texto encerrado por comillas "". Se antepone carácter '\'.
new Imagen[] = "/l0.gif"

WebServerPrintStr("<img src=\"%s\">", Imagen)
```

```
<img src="/l0.gif">
```

```
// Imprimir con relleno.
WebServerPrintStr("Cuenta: %05d", 112)
```

```
Cuenta: 00112
```

```
// Imprimir sin relleno.
WebServerPrintStr("Cuenta: %5d", 112)
```

```
Cuenta:   112
```

```
// Limitar precisión.
WebServerPrintStr("Tension: %.3f [V]", 11.34556)
```

```
Tension: 11.345 [V]
```

WebServerGetPrintNum(): Obtiene el número de cadena para imprimir requerida por el Servidor Web al cargar una página dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Entero	S	Numero de cadena a imprimir requerida por el Servidor Web.
Notas		**Descripción**
1		El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**PrintStr x**" en la página Web dinámica. Donde "**x**" es el numero de cadena a imprimir.
2		En general, esta función se llama desde el evento "**OnWebServerPrint**" para obtener el número de cadena a imprimir y en función al valor obtenido, se imprime la cadena solicitada con la función **WebServerPrintStr()**.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento  @OnWebServerPrint() para imprimir
   // cadenas en paginas dinamicas.
   WebServerSetPrintEvent()

   // . . .
}

@OnWebServerPrint()
{
      //
      // Obtener e imprimir el numero de cadena solicitada por el servidor.
      //

      switch(WebServerGetPrintNum())
      {
            //
            // Imprimir cadena numero 10.
            //
            case 10:
                  WebServerPrintStr("DIN1 = %d", DinValue(DIN1))

            //
            // Imprimir cadena numero 11.
            //
            case 11:
                  WebServerPrintStr("/l%d.gif", DinValue(DIN1))
      }
}
```

WebServerGetPrintArg1(): Obtiene el argumento número 1 (**Arg1**) asociado a una cadena requerida para imprimir por el servidor Web al cargar una página dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Entero	S	Numero de cadena a imprimir requerida por el Servidor Web.
Notas		**Descripción**
1		El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**PrintStr x Arg1**" en la página Web dinámica. Donde "**Arg1**" es el argumento (valor entero) asociado al número de la cadena "**x**" requerida para imprimir.
2		En general, este componente se llama desde el evento "**OnWebServerPrint**" para obtener el argumento de la cadena a imprimir y en función al valor obtenido, se realiza alguna acción o se ejecuta una lógica en particular.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento  @OnWebServerPrint() para imprimir
   // cadenas en paginas dinamicas.
   WebServerSetPrintEvent()

   // . . .
}

@OnWebServerPrint()
{
      new Num, Arg1

      // Obtener numero de cadena y argumento del comando CGI PrintStr.
      Num = WebServerGetPrintNum()
      Arg1 = WebServerGetPrintArg1()

      switch(Num)
      {
            // Imprimir cadena numero 10, que depende de Arg1.
            case 10:

                  // Procesar Arg1.
                  if(Arg1 == 1)
                     WebServerPrintStr("DIN1 = %d", DinValue(DIN1))
                  else if(Arg1 == 2)
                     WebServerPrintStr("DIN2 = %d", DinValue(DIN2))
                  else
                     WebServerPrintStr("DIN3 = %d", DinValue(DIN3))
      }
}
```

WebServerGetFuncCallNum(): Obtiene el número de llamada a función requerida por el servidor Web al cargar una página dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Entero	S	Número de llamada a función requerida por el Servidor Web.
Notas		**Descripción**
1		El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**FuncCall x**" en la página Web dinámica. Donde "**x**" es el número de función a llamar.
2		En general, este componente se llama desde el evento "**OnWebServerFuncCall**" para obtener el número de función a llamar y en función del valor obtenido se ejecuta alguna lógica dentro del PLC.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento @OnWebServerFuncCall() para ejecutar funciones
   // desde paginas dinámicas.
   WebServerSetFuncCallEvent()

   // . . .
}

@OnWebServerFuncCall()
{

      //
      // Obtener numero de función solicitada por el servidor.
      //

      switch(WebServerGetFuncCallNum())
      {
            //
            // Ejecutar función numero 20.
            //
            case 20:
                  DoutSetOn(DOUT1)

            //
            // Ejecutar función numero 21.
            //
            case 21:
                  DoutSetOn(DOUT2)
      }
}
```

WebServerGetFuncCallArg1(): Obtiene el argumento numero 1 (**Arg1**) asociado a una llamada a función requerida por el servidor Web al cargar una página dinámica.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Entero	S	Número de llamada a función requerida por el Servidor Web.
Notas		**Descripción**
1		El valor retornado es el ultimo valor especificado al encontrar el comando CGI "**FuncCall x Arg1**" en la página Web dinámica. Donde "**Arg1**" es el argumento (valor entero) asociado al número llamada de función "**x**" requerida por el servidor.
2		En general, este componente se llama desde el evento "**OnWebServerFuncCall**" para obtener el argumento, y en función al valor obtenido, se realiza alguna acción o se ejecuta una lógica en particular.

Ejemplo 1:

```
PlcMain()
{
   // Activar evento @OnWebServerFuncCall() para ejecutar funciones
   // desde paginas dinámicas.
   WebServerSetFuncCallEvent()

   // . . .
}

@OnWebServerFuncCall()
{
      new Num, Arg1

      // Obtener numero de función y argumento del comando CGI FuncCall.
      Num = WebServerGetFuncCallNum()
      Arg1 = WebServerGetFuncCallArg1()

      switch(Num)
      {
            // Ejecutar función 30, que depende de Arg1 y conmuta DOUTx.
            case 30:

                  // Procesar Arg1.
                  // Arg1 = 1 -> Conmuta DOUT1
                  // Arg1 = 2 -> Conmuta DOUT2
                  // Arg1 = 3 -> Conmuta DOUT3
                  // Arg1 = 4 -> Conmuta DOUT4... etc

                   DoutToggle(1<<(Arg1-1))
      }
}
```

8.4 Funciones Para Obtener Datos de Formularios Web

WebServerGetFieldValueStr(FieldName[ ], FieldVal[ ], FieldValMax): Obtiene el valor de una cadena (**FieldVal**) de un campo (F**ieldName**) proveniente de los datos enviados por un formulario POST en una página Web.

Argumentos	**Tipo**	**Descripción**
FieldName[]	E	Nombre del campo a buscar. Máximo 60 caracteres. Nota [1].
FieldVal[]	S	Valor de campo retornado como una cadena de texto. Máximo 60 caracteres. Si la cadena encontrada es mayor, se trunca el contenido. Siempre se incluye el carácter NULL al final.
FieldValMax	E	Número máximo de caracteres a copiar en FieldVal[]. Si la cadena encontrada es mayor, el contenido se trunca.
Retorno	**Tipo**	**Descripción**
>0	S	Numero de caracteres copiados en FieldVal[], si la cadena FieldName[] es encontrada.
0	S	FieldName[] no fue encontrada en el query-string.
-1	S	Error, dirección de array incorrecta.
-2	S	Error, falla en conversión.
Notas		**Descripción**
1		Muy importante. El nombre del campo (**FieldName**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario. Tenga en cuenta que solo el diseñador Web tiene control sobre este campo.
2		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
3		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

Ejemplo 1:

```
@OnWebServerPost()
{
      new FieldVal[20]

      // Obtener valor de campo "Name" de un formulario Web.
      if(WebServerGetFieldValueStr("Name", FieldVal, 20) > 0)
      {
            // Imprimir nombre en display de Virtual-HMI.
            nLcdPrintf(0, 1, LCD_NONE, "Su Nombre es: %s", FieldVal)
      }
}
```

WebServerGetFieldValueInt(FieldName[ ]): Obtiene el valor entero de un campo (F**ieldName**) proveniente de los datos enviados por un formulario POST en una página Web.		
Argumentos	**Tipo**	**Descripción**
FieldName[]	E	Nombre del campo a buscar. Máximo 60 caracteres. Nota [1].
Retorno	**Tipo**	**Descripción**
Entero	S	Retorna el valor del campo en una variable del tipo **Int32 (32 bits con signo)**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.
Notas		**Descripción**
1		Muy importante. El nombre del campo (**FieldName**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario. Tenga en cuenta que solo el diseñador Web tiene control sobre este campo.
2		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
3		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

Ejemplo 1:

El siguiente ejemplo guarda en una variable entera el valor recibido desde un formulario web, luego de que es enviado mediante el Navegador Web. El formulario tiene un elemento “input” con el atributo “name=**ANALOG**”. Si el usuario del navegador Web, introduce un numero positivo o negativo, el mismo será recibido correctamente. Si introduce una palabra, por ejemplo “hola 565”, el valor convertido será 0.

```
@OnWebServerPost()
{
      new AnalogOut

      // Obtener Valor de  salida  nalógica y guardar en variable.

      AnalogOut = WebServerGetFieldValueInt(“ANALOG”)

      // Escribir valor en salida Analogica VOUT1.

      VoutWrite(VOUT1, AnalogOut)
}
```

WebServerGetFieldValueBool(FieldName[ ]): Obtiene el valor booleano de un campo (**FieldName**) proveniente de los datos enviados por un formulario POST en una página Web.		
Argumentos	**Tipo**	**Descripción**
FieldName[]	E	Nombre del campo a buscar. Máximo 60 caracteres. Nota [1].
Retorno	**Tipo**	**Descripción**
Entero	S	Retorna el valor del campo como una variable del tipo Bool **(1 o 0)**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.
Notas		**Descripción**
1		Muy importante. El nombre del campo (**FieldName**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario. Tenga en cuenta que solo el diseñador Web tiene control sobre este campo.
2		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
3		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

Ejemplo 1:

El siguiente conmuta la salida digital DOUT1 según el valor recibido desde un formulario web, luego de que es enviado mediante el Navegador Web. El formulario tiene un elemento “input” con el atributo “name=**DO1**”. Si el usuario del navegador Web, introduce un número 0 o 1, el mismo será recibido correctamente. Si introduce una palabra, por ejemplo “hola 565”, el valor convertido será 0.

```
@OnWebServerPost()
{
	//
	// Activar/Desactivar salida DOUT1 según valor del campo DO1 recibido.
	//

	// Obtener el valor Bool del campo DO1.
	If(WebServerGetFieldValueBool(“DO1”))
	{
		DoutSetOn(DOUT1)
	}
	else
	{
		DoutSetOff(DOUT1)
	}
}
```

Este código se trato en profundidad en el ejemplo de la pagina 52.

WebServerGetFieldValueFloat(FieldName[ ]): Obtiene el valor punto flotante (**float**) de un campo (F**ieldName**) proveniente de los datos enviados por un formulario POST en una página Web.		
Argumentos	**Tipo**	**Descripción**
FieldName[]	E	Nombre del campo a buscar. Máximo 60 caracteres. Nota [1].
Retorno	**Tipo**	**Descripción**
Float	S	Retorna el valor del campo en una variable del tipo **Float (punto flotante)**. En caso de error o campo no encontrado, el valor retornado es 0 siempre.
Notas		**Descripción**
1		Muy importante. El nombre del campo (**FieldName**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario. Tenga en cuenta que solo el diseñador Web tiene control sobre este campo.
2		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
3		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.

Ejemplo 1:

El siguiente ejemplo guarda en una variable entera el valor recibido desde un formulario web, luego de que es enviado mediante el Navegador Web. El formulario tiene un elemento “input” con el atributo “name=**TEMP**”. Si el usuario del navegador Web, introduce un numero positivo o negativo, el mismo será recibido correctamente. Si introduce una palabra, por ejemplo “hola 565”, el valor convertido será 0.

```
@OnWebServerPost()
{
      new Float: TemperatureSetPoint

      // Obtener Valor de  “SetPoint” para Temperatura y guardar
      // en variable.

      TemperatureSetPoint = Float: WebServerGetFieldValueFloat(“TEMP”)
}
```

WebServerCheckFieldValueStr(FieldName[ ], FieldVal[ ]): Comprueba si existe un valor de una cadena (**FieldVal)** determinado en un campo (**FieldName**) proveniente de los datos enviados por un formulario POST en una página Web..		
Argumentos	**Tipo**	**Descripción**
FieldName[]	E	Nombre del campo a buscar. Máximo 60 caracteres. Nota [1].
FieldVal[]	E	Valor del campo a comparar con el valor recibido. Máximo 60 caracteres.
Retorno	**Tipo**	**Descripción**
1	S	Valor de FieldVal[] existe y coincide con el recibido.
0	S	FieldName[] o FieldVal[] no fue encontrada en el query-string.
-1	S	Error, dirección de array incorrecta.
-2	S	Error, falla en conversión.
Notas		**Descripción**
1		Muy importante. El nombre del campo (**FieldName**) en el formulario (pagina Web) debe contener caracteres ASCII, sin caracteres espaciales ni espacios en blanco. Se recomienda nombres cortos y únicos para cada formulario. Tenga en cuenta que solo el diseñador Web tiene control sobre este campo.
2		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
3		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.
4		Esta función puede ser útil para realizar un sistema de Passwords.

Ejemplo 1:

El siguiente ejemplo identifica el nombre del formulario web recibido, luego de que es enviado mediante el Navegador Web. Si el formulario tiene un elemento “input” con el atributo “name=**Form**” y el valor “value=**Outputs**”, se llama a la función **ProcessFormOutputs()** donde el usuario procesa otros campos recibidos del formulario.

```
@OnWebServerPost()
{
	//
	// Determinar el formulario de donde recibimos los datos.
	//

	if(WebServerCheckFieldValueStr("Form", "Outputs") == 1)
	{
		// Procesar formulario "Outputs" en outputs.shtml.
		ProcessFormOutputs()
	}
}
```

Este código se trato en profundidad en el ejemplo de la pagina 52.

WebServerGetQueryLength(): Obtiene la longitud en caracteres de la ultima cadena-query (**Query String**) proveniente de los datos enviados por un formulario POST en una página Web.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
Entero	S	Longitud en caracteres del último **Query-String** recibido.
Notas		**Descripción**
1		Este componente puede llamarlo desde un evento **OnWebServerPost**, o desde eventos **OnWebServerPrint / OnWebServerFuncCall**, si la página fue cargada como resultado de la carga de un formulario POST. De esta manera se asegura obtener el último valor de campo cargado por el cliente Web.
2		El nombre del campo **FieldName** se busca dentro de un **Query-String** enviado por el formulario a través del método POST.
3		Este componente es útil para situaciones donde se debe realizar una prueba de recepción o depuración de páginas Web.

Ejemplo 1:

```
@OnWebServerPost()
{
      new QueryLen

      // Obtener valor longitud del Query-String recibido.
      QueryLen = WebServerGetQueryLength()

      // Imprimir longitud eb display de Virtual-HMI.
      nLcdPrintf(0, 1, LCD_NONE, "Longitud: %d", QueryLen)
}
```

9 Comandos CGI Soportados por el Servidor

Los siguientes comandos CGI (Common Gateway Interface) pueden ser utilizados en páginas Web dinámicas (extensión **.shtml**) del Servidor Web del PLC de Slicetex Electronics. Estos comandos permiten ejecutar acciones y enviar datos desde el servidor al programa del PLC.

Los comandos CGI deben estar encerrados por los caracteres **$!** y **$** para que el servidor los interprete como comandos.

Por ejemplo:

```
$! Comando $
```

Es posible incluir varios comandos en una misma línea:

```
$! Comando1 $ $! Comando2 $
```

Se aconseja respetar los espacios en blancos entre comandos y escribirlos como figuran los ejemplos.

Hay que aclarar que el texto del comando CGI se reemplaza por el resultado de la acción del comando.

Comando: + /file Descripción: Incluye el contenido de un archivo del Servidor Web en la pagina dinámica.		
Argumentos	**Tipo**	**Descripción**
file	E	Nombre de archivo. Debe incluir extensión.
Retorno	**Tipo**	**Descripción**
-	-	-
Notas		**Descripción**
1		Si el archivo no existe, no se incluye ningún contenido.
2		Si el archivo a incluir es otra página dinámica, la pagina dinámica incluida no es procesada como pagina dinámica, sino como estatica.

Ejemplo 1 – Pagina Web Dinámica:

```
$!+ /header.html $

<h2>Principal</h2>

<p>¡Bienvenido al servidor Web del PLC!.</p>

$!+ /footer.html $
```

El ejemplo anterior, incluye el contenido de los archivos **header.html** y **footer.html** en la página web. Los mismos incluyen código HTML de encabezado (como títulos, menúes, etc) y pie de página. Esto nos permite ahorrar espacio en el servidor, ya que cada página dinámica puede reutilizarlos, y no se duplican.

Comando: PrintStr Num Arg1
Descripción: Solicita la impresión de una cadena de texto en la página web dinámica, para ello genera un evento **OnWebServerPrint** en el Servidor Web que el programa del PLC puede utilizar para devolver la cadena a imprimir.

Argumentos	**Tipo**	**Descripción**
Num	E	Numero de cadena de texto requerida. Numero entero. Obligatorio.
Arg1	E	Argumento numero 1. Numero entero. Opcional.
Retorno	**Tipo**	**Descripción**
-	-	-
Notas		**Descripción**
1		La página dinámica detiene su carga hasta que el programa del PLC devuelve la cadena de texto solicitada. En caso de que el programa del PLC tarde un tiempo excesivo en retornar la cadena, de produce un timeout, y el servidor continua la carga de la pagina.
2		El programa del PLC desde el evento **OnWebServerPrint** puede obtener el numero cadena "**Num**" y valor de "**Arg1**" (opcional) solicitados por el servidor. Ver ejemplos en páginas 21 y 47.
3		No es obligatorio retornar o imprimir una cadena. Si no retorna o imprime cadena, el comando será reemplazado por un carácter en blanco.

Ejemplo 1 – Pagina Web Dinámica:

```
$!+ /header.html $

<h2>Entradas Digital</h2>

<center>

<table>
  <tr>
    <th>ENTRADA</th>
    <th>VALOR</th>
  </tr>
  <tr>
    <td>DIN1</td>
    <td>$! PrintStr 10 1 $</td>
  </tr>
</table>

<br>
</center>

$!+ /footer.html $
```

El servidor al encontrar el comando **$! PrintStr 10 1 $**, genera el evento **OnWebServerPrint**, donde el programa del PLC puede obtener el valor Num=10 y Arg=1. Luego retorna por ejemplo la cadena "**ON**" si la entrada DIN está en alto, o la cadena "**OFF**" si la cadena esta en nivel bajo.

Comando: FuncCall Num Arg1
Descripción: Solicita la ejecución de una función personalizada en la página web dinámica, para ello genera un evento **OnWebServerFuncCall** en el Servidor Web que el programa del PLC puede utilizar para ejecutar la funcion.

Argumentos	**Tipo**	**Descripción**
Num	E	Numero de funcion requerida. Numero entero. Obligatorio.
Arg1	E	Argumento numero 1. Numero entero. Opcional.
Retorno	**Tipo**	**Descripción**
-	-	-
Notas		**Descripción**
1		La página dinámica detiene su carga hasta que el programa del PLC ejecute la función solicitada. En caso de que el programa del PLC tarde un tiempo excesivo en retornar de la ejecución, de produce un timeout, y el servidor continua la carga de la pagina.
2		El programa del PLC desde el evento **OnWebServerFuncCall** puede obtener el número de función "**Num**" y valor de "**Arg1**" (opcional) solicitados por el servidor.
3		Este comando es recomendado para ejecutar funciones que no retornar texto la página dinámica.

Ejemplo 1 – Pagina Web Dinámica:

```
$!+ /header.html $

<h2>Alarma</h2>

$! FuncCall 15 1 $

<center>

<p>¡Se activó una alarma!</p>

</center>

$!+ /footer.html $
```

El servidor al encontrar el comando **$! FuncCall 15 1 $**, genera el evento **OnWebServerFuncCall**, donde el programa del PLC puede obtener el valor Num=15 y Arg=1, y en base a dicho valor activar una salida DOUT conectada a una alarma. Esta página con el simple hecho de accederla, activa la alarma, sin imprimir ningún contenido dinámico.

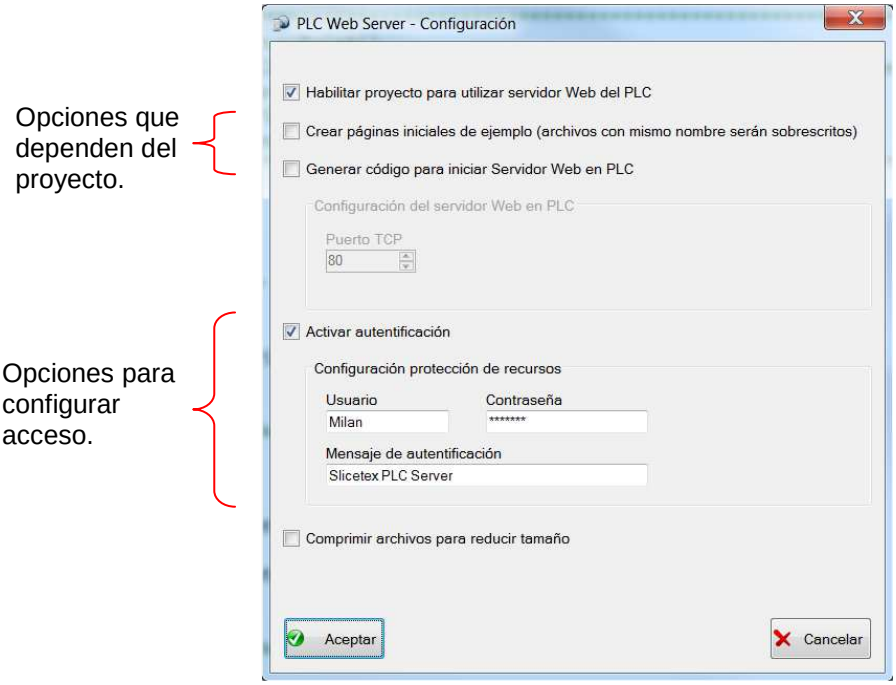
También puede ser útil usar esta función CGI en páginas que fueron cargadas como resultado de un formulario web usando el método POST.

Se puede usar en combinación con el comando CGI del tipo **PrintStr** para otras lógicas.

10 Opciones de Configuración del Servidor

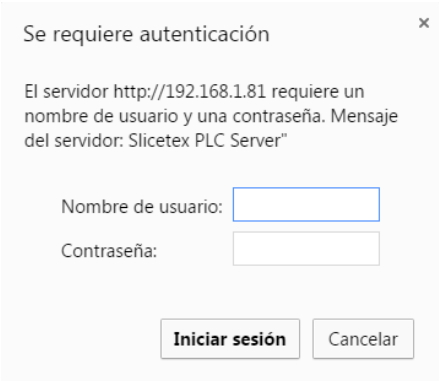
10.1 Acceso con Contraseña al Servidor Web

Es posible configurar una contraseña de acceso para restringir el ingreso al servidor Web. Para activarlo, debemos ir al menú "**Herramientas > PLC Web Server**" del entorno **StxLadder** y elegir el tilde "**Activar Autentificación**" como se muestra a continuación:



Seleccione un **Usuario** y **Contraseña** para entrar al servidor. El **Mensaje de autentificación** es para fines informativos, y se recomienda un nombre único entre diferentes Servidores Web configurados con contraseña.

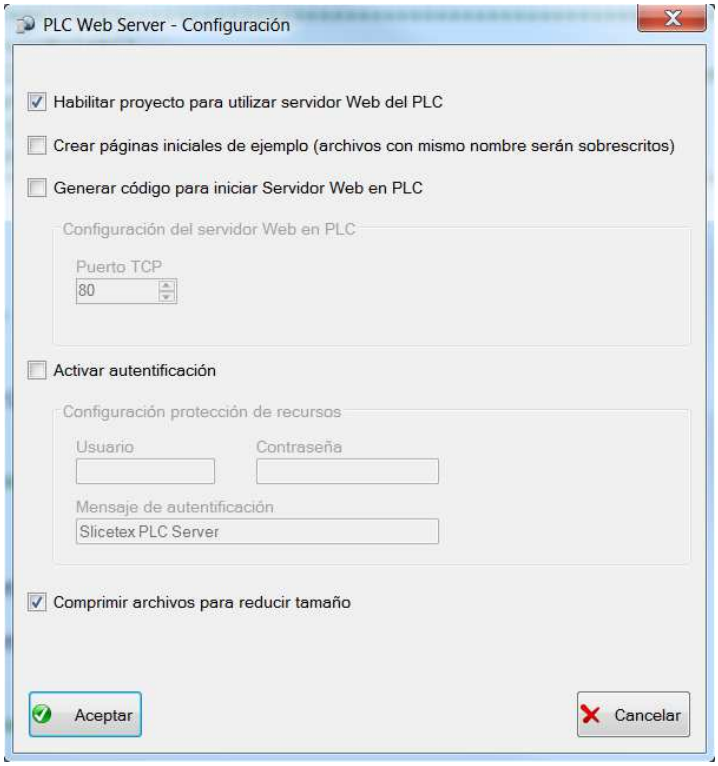
Una vez configurada la contraseña, al acceder desde el navegador al Servidor Web se solicitará acceso con un cartel similar al siguiente:



Si se introduce mal el nombre de usuario o contraseña, el servidor carga la pagina "**401.html**" que informa "Acceso Denegado". El contenido de página es configurable, pero es obligatorio que exista.

10.2 Comprimir Archivos

Si los archivos del servidor web ocupan mucho espacio y comienza a ser crítico en su proyecto, considere utilizar la opción "**Comprimir archivos para reducir tamaño**" como se muestra en la siguiente imagen:



Compresión de datos

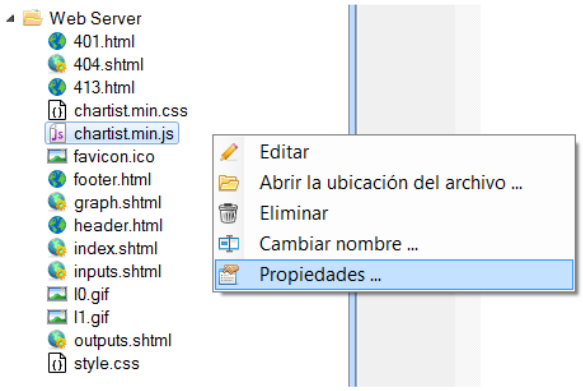
Una vez activada dicha opción, ciertos archivos de texto del servidor serán comprimidos. Por ejemplo las extensiones de archivos que se comprimen (al momento de escribir esta nota) son: .css, .js, .txt, .xml y .ico.

Esto puede reducir notablemente el tamaño ocupado por dichos archivos en el servidor web, especialmente si usamos librerías JavaScript (.js) o hojas de estilos complejas (.css).

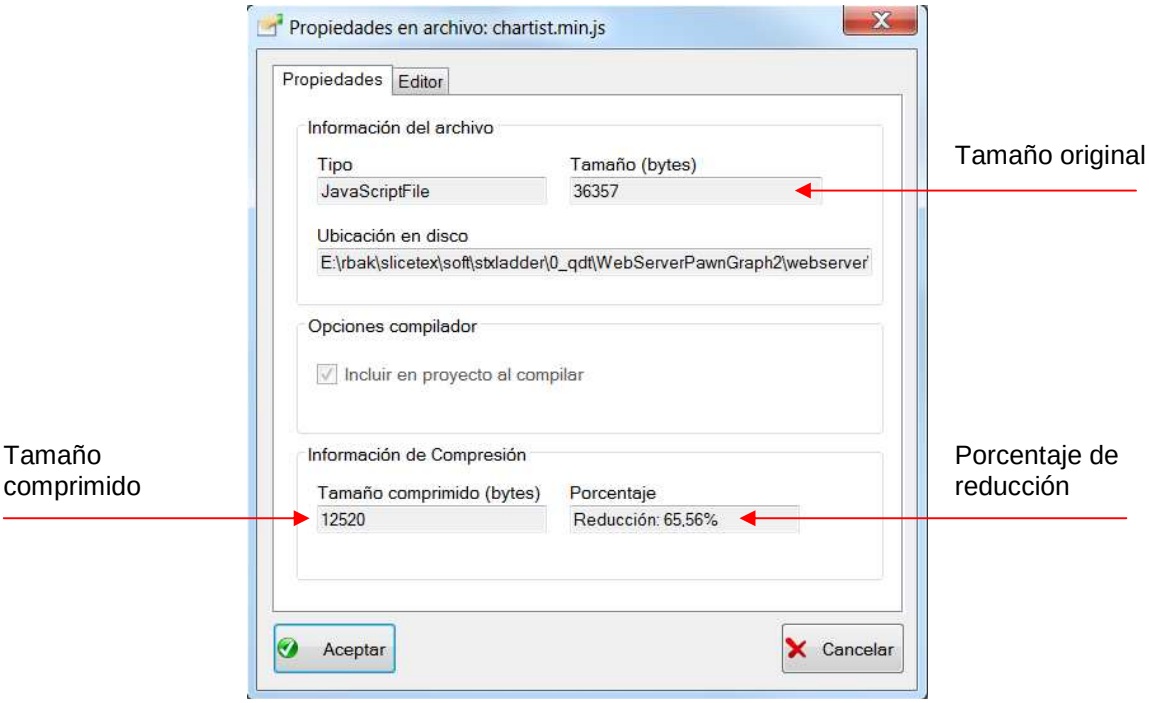
Los archivos de imágenes .png, .jpg y .gif ya están comprimidas por definición, por lo tanto no se comprimen.

En cuanto a los archivos .html y .shtml no son comprimidos por consideraciones de diseño internas.

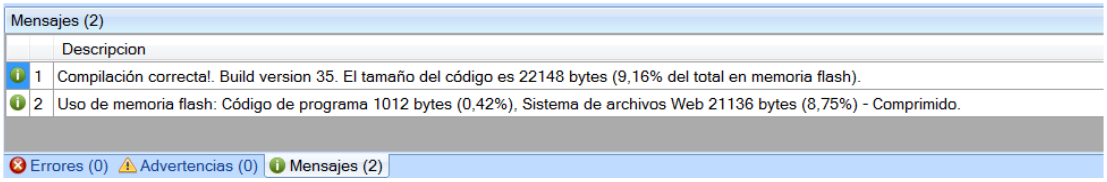
Si seleccionamos las **Propiedades** de un archivo (con extensión apta para compresión) desde el **Explorador de Proyecto** como se muestra en la imagen a continuación:



Obtenemos la siguiente ventana con información de compresión, donde observamos una reducción del **65.56%** del archivo comprimido respecto al tamaño original:



Al compilar el proyecto, en la sección de Mensajes, **StxLadder** nos indicará que el sistema de archivos web esta comprimido:



Finalmente, debemos mencionar que la compresión de archivos puede traer problemas de compatibilidad con navegadores antiguos o básicos, sin embargo en la mayoría de navegadores modernos y actualizados como Chrome, Firefox, Edge, Explorer, Opera, Safari, etc no tendría ningún inconveniente.

11 Recomendaciones de Diseño Web

A continuación presentamos algunos consejos útiles a la hora de encarar un diseño de página web.

Recomendamos altamente utilizar Google para buscar ejemplos de código HTML y CSS. Se recomienda por empezar con ejemplos básicos y no utilizar programas de diseño a menos que entienda lo que esta haciendo. Puede ver algunos links informativos en la sección de **Requerimientos** de la página 2.

11.1 Encabezados o Headers

Si siguió los ejemplos de esta nota de aplicación, habrá notado que siempre se incluyen los archivos **header.html** y **footer.html** en páginas dinámicas, esto no solo disminuye el espacio en memoria utilizado por el Servidor Web al reutilizar código, sino que permite incluir el siguiente código HTML muy útil:

Contenido de **header.html**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="es">

<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <meta name="format-detection" content="telephone=no">
      <meta name = "viewport" content = "width = 320, initial-scale = 1.0,
user-scalable = no">

      <title>Slicetex PLC Web Server!</title>
      <link rel="stylesheet" type="text/css" href="/style.css">
</head>

<body bgcolor="#fffeec" text="black">

<div class="menu">
      <div class="menubox"><a href="/">Inicio</a></div>
      <div class="menubox"><a href="/inputs.shtml">Entradas</a></div>
      <div class="menubox"><a href="/outputs.shtml">Salidas</a></div>
      <div class="menubox"><a href="/info.shtml">Info</a></div>
      <br>
</div>

<div class="contentblock">
```

La línea:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Le dice al Navegador que se trata de un archivo HTML del tipo “**transitional”** y define para ello una serie de atributos y elementos estándar para la página. No hay que saber más desde un punto de vista práctico.

Las líneas:

```
<html lang="es">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Son necesarias para decirle que el contenido de la pagina web esta en español y que se use un juego de caracteres "**utf-8**". De otra manera, es posible que los acentos y caracteres específicos del idioma en español no se muestren adecuadamente.

Las líneas:

```
      <meta name="format-detection" content="telephone=no">
      <meta name = "viewport" content = "width = 320, initial-scale = 1.0,
user-scalable = no">
```

Son "**metas**" muy útiles para los tiempos modernos. Nos permiten hacer que el contenido web de la página se represente adecuadamente en los dispositivos móviles o tablets.

La primer línea especifica al navegador que los números de la pagina web no deben considerarse "números telefónicos". Esto es muy útil para que no se presenten como "links de acceso para llamada" cualquier número presentado.

Mientras que la segunda línea regula como la pagina es visualizada en el dispositivo.

Observar la siguiente captura desde un Smartphone de una página de ejemplo disponible, la página Web se muestra perfectamente dimensionado para la pantalla del dispositivo, gracias a los "**metas**" que indican al navegador como mostrarla.

La línea:

```
<title>Slicetex PLC Web Server!</title>
```

Nos permite especificar el titulo que se mostrará en el la pestaña del navegador o barra de la ventana, al abrir la pagina web.

La línea:

```
<link rel="stylesheet" type="text/css" href="/style.css">
```

Nos permite especificar un archivo .CSS de estilos, que modifica la presentación del código HTML en la pagina. En este caso se especifica el archivo "**style.css**", el cual está disponible en los ejemplos del Servidor Web. Más adelante hablaremos de CSS.

La línea:

```
<body bgcolor="#fffeec" text="black">
```

Nos da el color del fondo y texto por defecto HTML de la página.

Las líneas:

```
<div class="menu">
      <div class="menubox"><a href="/">Inicio</a></div>
      <div class="menubox"><a href="/inputs.shtml">Entradas</a></div>
      <div class="menubox"><a href="/outputs.shtml">Salidas</a></div>
      <div class="menubox"><a href="/info.shtml">Info</a></div>
      <br>
</div>
```

Permiten crear el menú siguiente:

El cual usa configuración proveniente del archivo CSS en **styles.css**.

Finalmente, la línea:

```
<div class="contentblock">
```

Crea una clase "**contenblock**" que hereda atributos comunes, la cual es configurada desde el archivo CSS.

11.2 Hoja de Estilos CSS

11.2.1 Introducción

Los archivos CSS definen estilos para las páginas web, que incluyen diseños, layout (presentación) y variaciones en la visualización para diferentes dispositivos y tamaños de pantalla.

El lenguaje HTML **nunca tuvo la intención de definir tags que permitan formatear visualmente** una página web, ya que el mismo **describe el contenido** de una página web, por ejemplo:

```
<h1>Esto es un encabezado.</h1>
<p>Esto es un parrafo.</p>
```

Cuando se intentó introducir **tags** para tal fin, como **<font>** y atributos de color en la especificación de HTML 3.2, comenzó una pesadilla para los desarrolladores web, ya que el desarrollo de sitios web con muchas páginas debía contener información de tipos de letras (fonts) y colores en cada simple página, lo que generó un excesivo desorden e ineficiente sistema para ser procesado por los navegadores web.

Para solucionar este problema, el World Wide Web Consortium (W3C) creó CSS.

¡La hoja de estilo CSS eliminó la formato de estilos de las paginas HTML!

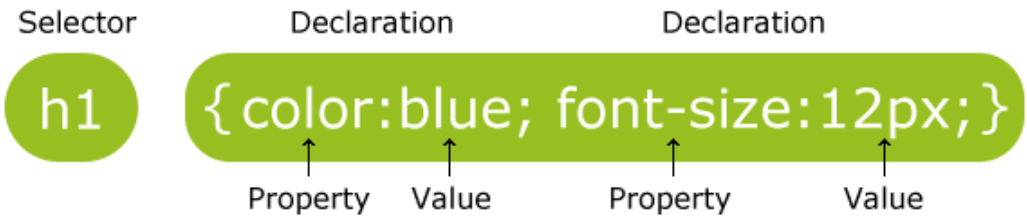
11.2.2 CSS Ahorra Mucho Trabajo

La hoja de estilos normalmente se almacena en un archivo externo con extensión **.css**

¡Con una hoja de estilos externa, se puede cambiar la apariencia del sitio web completo con solo editar un solo archivo!

11.2.3 Sintaxis de una Regla CSS

Una regla CSS consiste en un “selector” y un bloque de declaraciones entre corchetes { } .



El “selector” apunta al elemento o tag HTML que se desea aplicar el estilo.

En el bloque de declaraciones, una o más declaraciones se separan con “;”, que debe ser el ultimo carácter. Puede colocarse una declaración por línea o muchas por línea.

Cada declaración incluye un nombre propiedad CSS y un valor asignado, ambos separados por “:”.

11.2.4 Usando Selectores CSS

Los selectores (**selector**) CSS, se utilizan para buscar o seleccionar elementos HTML basados en el nombre del elemento, un id (identificador), una clase, atributo o más.

Ejemplo:

CSS:

```
p {
    color: red;
    text-align: center;
}
```

La anterior regla CSS selecciona todos los elementos **<p>** de una página web, y les da el atributo "alinear al centro" y "hacer de color rojo".

Entonces si en la página web escribimos el código HTML:

```
<p>Hola, soy un párrafo.</p>
```

En el navegador web se mostrará:

Hola, soy un párrafo.

Esto se aplica para todos los elementos o tags **<p>** del sitio web que usen la hoja de estilo CSS.

11.2.5 El Selector Identificador (id)

Si necesitamos seleccionar un elemento en particular para darle formato, podemos usar un "**id**". El cual debe ser un nombre **único** en la pagina, comenzar con el carácter #, y nunca empezar por un numero.

Ejemplo CSS: Especifica una regla aplicada solo para los elementos con **id = "para1"**

```
#para1 {
    text-align: center;
    color: red;
}
```

Entonces si en la página web escribimos el código HTML:

```
<p>Hola, soy un párrafo normal.</p>
<p id="para1">Hola, soy un párrafo con estilo personalizado.</p>
```

En el navegador web se mostrará:

Hola, soy un párrafo normal.

Hola, soy un párrafo con estilo personalizado.

11.2.6 El Selector de Clase (class)

El selector de clase selecciona elementos con un **atributo** de clase específico.

Para seleccionar los elementos de clase, usar el carácter ".", seguido por el nombre de la clase.

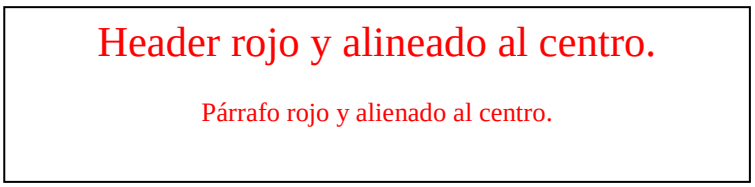
El siguiente código CSS selecciona todos los elementos con clase "class=center", definiéndoles "como rojos y alineados al centro":

```
.center {
     text-align: center;
     color: red;
}
```

Si tenemos el siguiente código HTML:

```
<h1 class="center">Header rojo y alineado al centro.</h1>
<p class="center">Párrafo rojo y alineado al centro.</p>
```

Se visualiza en el navegador:



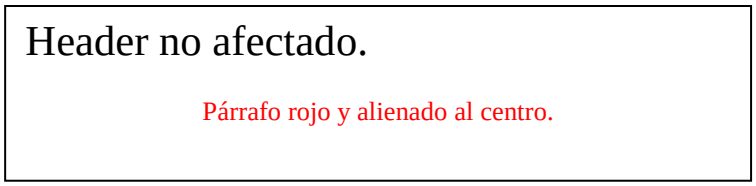
Si solo queremos que un elemento HTML especifico sea afectado por una clase, seleccionamos elemento y luego con "." la clase, por ejemplo en CSS:

```
p.center {
   text-align: center;
   color: red;
}
```

Si tenemos el siguiente código HTML:

```
<h1 class="center">Header no afectado.</h1>
<p class="center">Párrafo rojo y alineado al centro.</p>
```

Se visualiza en el navegador:



11.2.7 Agrupando Varios Selectores

Para minimizar código (y ahorrar especio en Servidor Web) podemos agrupar varios selectores en un archivo CSS separándolos por "," de la siguiente manera:

```
h1, h2, p {
     text-align: center;
     color: red;
}
```

Es equivalente al siguiente código:

```
h1 {
     text-align: center;
     color: red;
}

h2 {
     text-align: center;
     color: red;
}

p {
     text-align: center;
     color: red;
}
```

11.2.8 Archivo "styles.css" del Ejemplo

Ahora que conoce los aspectos básicos de CSS está en condiciones de interpretar el archivo "**styles.css**" incluido en los ejemplos presentados en esta nota de aplicación. El cual se provee como base, usted puede modificarlo y mejorarlo para sus proyectos.

Si abrimos "**styles.css**" con StxLadder desde el "**Explorador de Proyecto**", se obtiene el siguiente contenido:

```
/* Header elements */

h1
{
  text-align: center;
  font-size:14pt;
  font-family:arial,helvetica;
  font-weight:bold;
  padding:10px;
}

h2
{
  text-align: center;
  font-size:20pt;
  font-family:arial,helvetica;
  font-weight:bold;
  padding:10px;
}
```

```
/* Body */

body
{

  background-color: white;
  color:black;

  font-size:14pt;
  font-family:arial,helvetica;
}

/* Paragraph elements */

p.intro
{
  margin-left:20px;
  margin-right:20px;

  font-size:10pt;
  font-family:arial,helvetica;
}

p.clink
{
  font-size:12pt;
  font-family:courier,monospace;
  text-align:center;
}

p.clink9
{
  font-size:9pt;
  font-family:courier,monospace;
  text-align:center;
}

p
{
  padding-left:10px;
}

p.right
{
  text-align:right;
}

/* Classes */

.menu
{
  margin: 4px;
  width:95%;

  padding:2px;

  border: solid 1px;
  background-color: #005CE6;
  text-align:left;

  font-size:16pt;
  font-size:3vw;
  font-family:arial,helvetica;
}
```

```
div.menubox
{
  width: 25%;
  border: 0;
  float: left;
  text-align: center;
}

.contentblock
{
  margin: 4px;
  width:95%;

  padding:2px;

  border: solid 1px;
  background-color: #ebf5fb;

  font-size:14pt;
  font-family:arial,helvetica;
}

/* Unvisited link */
.menu a:link {
    color: #E6E6E6;
    text-decoration: none;
}

/* Visited link */
.menu a:visited
{
    color: #E6E6E6;
    text-decoration: none;
}

/* Mouse over link */
.menu a:hover
{
    color: #99DAEB;
    text-decoration: none;
}

/* Tables Config */

table
{
    width:90%;
}

table, th, td
{
    border: 1px solid black;
    border-collapse: collapse;
}

th, td
{
    padding: 5px;
    text-align: left;
}
```

```
/* Table Input Design */

table#tInputs tr:nth-child(even){
    background-color: #e0e0eb;
}

table#tInputs tr:nth-child(odd){
   background-color:#fff;
}

table#tInputs th{
    background-color: #00cc00;
    color: white;
}

/* Table Output Design */

table#tOutputs tr:nth-child(even) {
    background-color: #e0e0eb;
}

table#tOutputs tr:nth-child(odd) {
   background-color:#fff;
}

table#tOutputs th     {
    background-color: #ff3333;
    color: white;
}

/* Input "update" Button Design */

input.update {
       font-size:16pt;
}
```

Si lo analizamos con lo que ya sabemos, no hay nada raro.

Analicemos los fragmentos notables:

```
h1
{
  text-align: center;
  font-size:14pt;
  font-family:arial,helvetica;
  font-weight:bold;
  padding:10px;
}

h2
{
  text-align: center;
  font-size:20pt;
  font-family:arial,helvetica;
  font-weight:bold;
  padding:10px;
}
```

Muy simple, define atributos comunes para los tags o elementos HTML **<h1>** y **<h2>,** como tamaños, fuente de letra, alineación, etc.

El siguiente código CSS especifica “para la clase menu”, hacer que el estilo de los links (selector “a”) sea diferente para cuando “es un link normal (estado link), sea visitado (estado visited) y para cuando el mouse pase por encima del mismo (estado hover):

```
/* Unvisited link */
.menu a:link {
    color: #E6E6E6;
    text-decoration: none;
}

/* Visited link */
.menu a:visited
{
    color: #E6E6E6;
    text-decoration: none;
}

/* Mouse over link */
.menu a:hover
{
    color: #99DAEB;
    text-decoration: none;
}
```

Esto hace que el menú en el navegador tenga el siguiente aspecto:

Sobre “Entradas” se paso el mouse, haciendo que cambie de color. Además, notar que al hacer click sobre un link del menú no cambia de color. Esto es porque “**a:link**” y “**a:visited**” tienen el mismo valor del atributo “**color: #E6E6E6**”.

Buscar en la pagina header.html, el siguiente código HTML que define al menú en cuestión:

```
<div class="menu">
      <div class="menubox"><a href="/">Inicio</a></div>
      <div class="menubox"><a href="/inputs.shtml">Entradas</a></div>
      <div class="menubox"><a href="/outputs.shtml">Salidas</a></div>
      <div class="menubox"><a href="/info.shtml">Info</a></div>
      <br>
</div>
```

Notar cómo se utiliza “class=menu” para obtener las reglas del archivo CSS.

Volviendo al código CSS del archivo styles.css:

```
.contentblock
{
  margin: 4px;
  width:95%;

  padding:2px;
  border: solid 1px;
  background-color: #ebf5fb;
  font-size:14pt;
  font-family:arial,helvetica;
}
```

Nos dice que todas clases “**contentblock**” del código HTML tendrán dichas propiedades. Notar “width” que define un 95%. Esto influye en el porcentaje de la página web ocupada en la ventana del navegador.

Un código importante es:

```
/* Tables Config */

table
{
    width:90%;
}

table, th, td
{
    border: 1px solid black;
    border-collapse: collapse;
}

th, td
{
    padding: 5px;
    text-align: left;
}
```

Que define atributos comunes para representar tablas.

Sin embargo, para tablas personalizadas usamos:

```
/* Table Input Design */

table#tInputs tr:nth-child(even){
    background-color: #e0e0eb;
}

table#tInputs tr:nth-child(odd){
   background-color:#fff;
}

table#tInputs th{
    background-color: #00cc00;
    color: white;
}

/* Table Output Design */

table#tOutputs tr:nth-child(even) {
    background-color: #e0e0eb;
}

table#tOutputs tr:nth-child(odd) {
   background-color:#fff;
}

table#tOutputs th     {
    background-color: #ff3333;
    color: white;
}
```

Esto nos da para tablas con “**id=tInputs**” y “**id=tOutputs**” (ver archivos **inputs.shtml y outputs.shtml**) visualizaciones personalizadas. Por ejemplo, definimos colores especiales para encabezados (th), y colores diferentes para filas impares (odd) y filas pares (even).

En el ejemplo numero 2 (ver sitio web de nota de aplicación AN030) podemos ver la visualización de la página **inputs.shtml** que tiene una tabla con 4 filas:

Su código HTML es:

```
<table id="tInputs">
  <tr>
    <th>ENTRADA</th>
    <th>VALOR</th>
    <th>LED</th>
  </tr>
  <tr>
    <td>DIN1</td>
    <td>$! PrintStr 1 $</td>
    <td><img src="$! PrintStr 2 $"></td>
  </tr>
  <tr>
    <td>DIN2</td>
    <td>$! PrintStr 3 $</td>
    <td><img src="$! PrintStr 4 $"></td>
  </tr>
  <tr>
    <td>DIN3</td>
    <td>$! PrintStr 5 $</td>
    <td><img src="$! PrintStr 6 $"></td>  </tr>
  <tr>
    <td>DIN4</td>
    <td>$! PrintStr 7 $</td>
    <td><img src="$! PrintStr 8 $"></td>
  </tr>
</table>
```

Notar como usamos el “id=tInputs” para aplicar las reglas del archivo CSS.

Finalmente en el archivo "**styles.css**" se define:

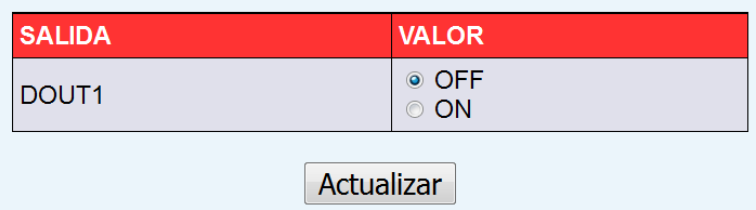
```
/* Input "update" Button Design */

input.update {
	font-size:16pt;
}
```

Selecciona el botón del elemento "**input**" con clase "**update**" y le da un tamaño de letra de "16pt".

Si vemos el código HTML de outputs.shtml el mismo se define como:

```
<input type="submit" value="Actualizar" class="update">
```

Mostrándose en pantalla como:



Si no definiéramos estos atributos en CSS, el botón "Actualizar" por defecto sería muy pequeño.

11.2.9 Colores HTML

Habrá notado como los colores son especificados en CSS o código HTML, con un carácter # seguido por un numero en hexadecimal.

Ejemplo CSS:

```
background-color: #ff3333;
```

Puede usar otros colores seleccionándolo de forma visual, para ello recomendamos el siguiente link donde es posible obtener el número del color a través de una paleta:

http://www.w3schools.com/tags/ref_colorpicker.asp

11.2.10 Más Información CSS

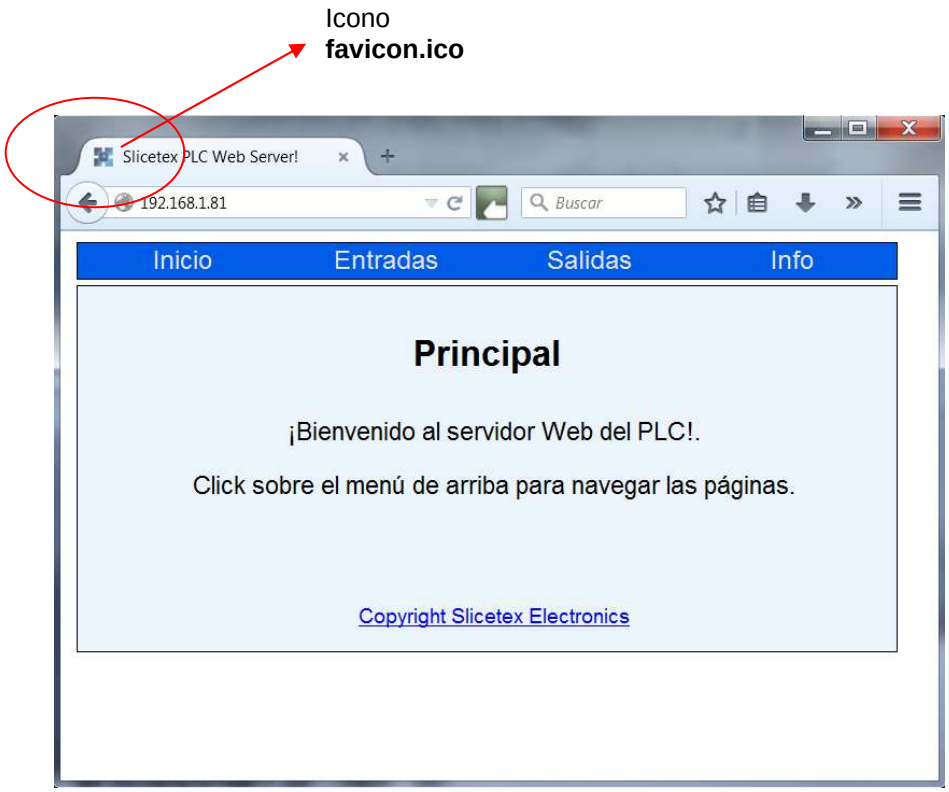
El siguiente link es muy útil para continuar aprendiendo sobre CSS y HTML:

http://www.w3schools.com/html/default.asp

http://www.w3schools.com/css/default.asp

11.3 Icono de Página

En los ejemplos de la nota de aplicación habrá notado al cargar la página web, que un icono se visualiza en la pestaña del navegador:



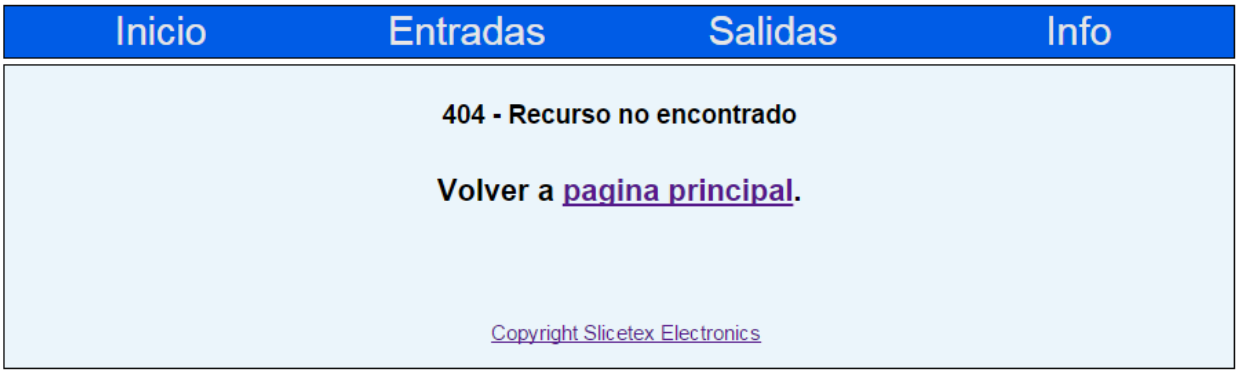
El icono es el archivo "**favicon.ico**" y es de uso opcional. En este caso se muestra el logo de Slicetex Electronics.

El tamaño del icono puede ser 16x16 (preferido) o 32x32. El formato es "ICO", con 8-bit o 24-bit de profundidad de colores.

En general los navegadores automáticamente intentan obtener el archivo **favicon.ico** y no es necesario especificarlo en la página web.

11.4 Pagina de Error 404

Cuando tipea mal el nombrede un recurso o archivo la barra de direcciones del navegador web, el Servidor Web muestra la página **404.html** o **404.shtml** (dependiendo de cual este definida) avisando al usuario del sitio web que no se pudo encontrar el recurso solicitado:



Por lo tanto es muy útil que siempre diseñe adecuadamente la pagina 404, así sus clientes web podrán saber qué hacer cuando un recurso no se encuentre disponible.

11.5 Próximos Pasos

¡Felicitaciones, ya es un principiante avanzado del diseño de sitios web! ☺

Recomendamos entrar al sitio web de la nota de aplicación **AN030** para descargar otros ejemplos más avanzados para el servidor Web:

www.slicetex.com/docs/an/an030

Puede efectuar consultas en nuestro foro de soporte técnico:

www.slicetex.com/foro/

```
<!DOCTYPE html>

<html>

<body>

<h1>Au revoir !!!</h1>

</body>

</html>
```

12 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.
- **CSS:** Código para realizar formato de páginas web.
- **CGI:** Interfaz de comandos para comunicarse desde el servidor al programa del PLC.
- **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.

13 Historial de Revisiones

Tabla 4: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
02 14/Feb/2016	2	1. Agrega extensión de archivos .js (JavaScript). Ver pág. 5. 2. Agrega descripción de opción para comprimir archivos, ver pág. 104.	Preliminar
01 29/Dic/2015	1	1. Versión preliminar liberada.	Preliminar

14 Referencias

Ninguna.

15 Información Legal

15.1 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.

16 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

© Slicetex Electronics, todos los derechos reservados.

17 Contenido

1 DESCRIPCIÓN GENERAL 1

2 LECTURAS RECOMENDADAS 2

2.1 ¿CÓMO LEER ESTA NOTA DE APLICACIÓN? 2

3 REQUERIMIENTOS 2

4 TEORÍA DE FUNCIONAMIENTO 3

4.1 CARACTERÍSTICAS DEL SERVIDOR WEB 4
4.2 DETALLES DE EXTENSIONES SOPORTADAS 5
4.3 ARCHIVOS WEB OBLIGATORIOS 6
4.4 CONSIDERACIONES PARA OPTIMIZAR EL FUNCIONAMIENTO 6

5 EJEMPLO DE USO INICIAL EN LENGUAJE LADDER 8

5.1 CREAR EL PROYECTO 8
5.2 ACCEDA A LA PÁGINA WEB DEL PLC 10
5.3 CONFIGURACIÓN DEL SERVIDOR WEB 13
5.4 CREANDO ARCHIVOS Y PÁGINAS WEB 14
5.5 MEMORIA UTILIZADA 16
5.6 PROCESO DE CARGA DE UNA PÁGINA DINÁMICA 17
5.7 MODIFICAR CONTENIDO DE PÁGINA DESDE PROGRAMA EN PLC 21
5.8 PROCESAR FORMULARIOS WEB Y RECIBIR DATOS DEL NAVEGADOR 26
5.9 FIN DEL EJEMPLO 33

6 EJEMPLO DE USO INICIAL EN LENGUAJE PAWN 34

6.1 CREAR EL PROYECTO 34
6.2 ACCEDA A LA PÁGINA WEB DEL PLC 36
6.3 CONFIGURACIÓN DEL SERVIDOR WEB 39
6.4 CREANDO ARCHIVOS Y PÁGINAS WEB 40
6.5 MEMORIA UTILIZADA 42
6.6 PROCESO DE CARGA DE UNA PÁGINA DINÁMICA 43
6.7 MODIFICAR CONTENIDO DE PÁGINA DESDE PROGRAMA EN PLC 47
6.8 PROCESAR FORMULARIOS WEB Y RECIBIR DATOS DEL NAVEGADOR 52
6.9 FIN DEL EJEMPLO 60

7 REFERENCIA DE COMPONENTES EN LENGUAJE LADDER 61

7.1 COMPONENTES DE CONFIGURACIÓN 62
7.2 COMPONENTES PARA EVENTOS 63
7.3 COMPONENTES PARA OBTENER DATOS E IMPRIMIR EN PÁGINAS DINÁMICAS 66
7.4 COMPONENTES PARA OBTENER DATOS DE FORMULARIOS WEB 72

8 REFERENCIA DE FUNCIONES EN LENGUAJE PAWN 78

8.1 FUNCIONES DE CONFIGURACIÓN 79
8.2 FUNCIONES PARA EVENTOS 80
8.3 FUNCIONES PARA OBTENER DATOS E IMPRIMIR EN PÁGINAS DINÁMICAS 86
8.4 FUNCIONES PARA OBTENER DATOS DE FORMULARIOS WEB 94

9 COMANDOS CGI SOPORTADOS POR EL SERVIDOR 100

10 OPCIONES DE CONFIGURACIÓN DEL SERVIDOR 103

10.1 ACCESO CON CONTRASEÑA AL SERVIDOR WEB 103
10.2 COMPRIMIR ARCHIVOS 104

11 RECOMENDACIONES DE DISEÑO WEB 107

11.1 ENCABEZADOS O HEADERS 107
11.2 HOJA DE ESTILOS CSS 110
11.2.1 INTRODUCCIÓN 110
11.2.2 CSS AHORRA MUCHO TRABAJO 110
11.2.3 SINTAXIS DE UNA REGLA CSS 110
11.2.4 USANDO SELECTORES CSS 111
11.2.5 EL SELECTOR IDENTIFICADOR (ID) 111
11.2.6 EL SELECTOR DE CLASE (CLASS) 112
11.2.7 AGRUPANDO VARIOS SELECTORES 113
11.2.8 ARCHIVO “STYLES.CSS” DEL EJEMPLO 113
11.2.9 COLORES HTML 120
11.2.10 MÁS INFORMACIÓN CSS 120
11.3 ICONO DE PÁGINA 121
11.4 PAGINA DE ERROR 404 122
11.5 PRÓXIMOS PASOS 122

12 ABREVIACIONES Y TÉRMINOS EMPLEADOS 123

13 HISTORIAL DE REVISIONES 123

14 REFERENCIAS 124

15 INFORMACIÓN LEGAL 124

15.1 AVISO DE EXENCIÓN DE RESPONSABILIDAD 124

16 INFORMACIÓN DE CONTACTO 125

17 CONTENIDO 126

17.1 ÍNDICE DE TABLAS 128

17.1 Índice de Tablas

Tabla 1: Características del Servidor Web 4
Tabla 2: Extensiones de archivos soportadas 5
Tabla 3: Archivos obligatorios 6
Tabla 4: Historia de Revisiones del Documento 123

Copyright Slicetex Electronics

www.slicetex.com