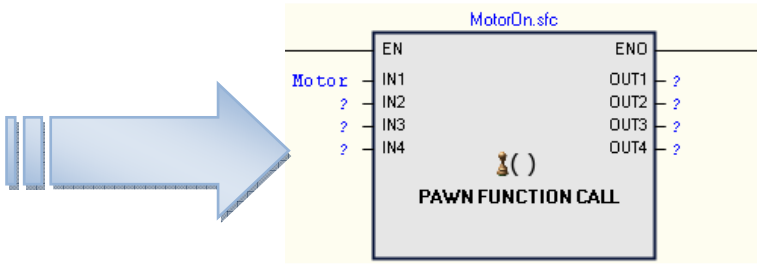
Slicetex Ladder Designer Studio

(StxLadder)

Mezclando Lenguaje Pawn con Ladder

```
switch(IN1)
{
    // Activar Motor 1.
    case 1:
        pBoolSet(gb_RELAY1)

    // Activar Motor 2.
    case 2:
        pBoolSet(gb_RELAY2)

    // Activar Motor 3.
    case 3:
        pBoolSet(gb_RELAY3)

    // Activar Motor 4.
    case 4:
        pBoolSet(gb_RELAY4)

    // Activar Alarma.
    default:
        pBoolSet(gb_RELAY5)
}
```

Autor: Ing. Boris Estudiez



1 Descripción General

*Este documento explica cómo combinar el lenguaje **Pawn** con el Lenguaje **Ladder** utilizando el entorno de programación **StxLadder**.*

El lenguaje Pawn es muy útil para realizar algoritmos complejos y exprimir al máximo la capacidad del PLC. También puede ser necesario para implementar lógicas de control que no son posibles de realizar con el lenguaje grafico Ladder.

Otra ventaja del lenguaje Pawn, es que puede utilizar todas las funciones de dicho lenguaje en el entorno Ladder. Tradicionalmente nuestros PLC implementan primero las mejoras en lenguaje Pawn y luego en lenguaje Ladder, por lo tanto puede utilizar características del PLC que no se encuentran disponibles en lenguaje Ladder.

*Finalmente, la verdadera potencia de nuestro entorno **StxLadder** se logra mediante la combinación de ambos lenguajes: Grafico (Ladder) y Textual (Pawn), obteniendo lo mejor de ambos mundos, simplicidad, rapidez y potencia a la hora de programar.*

2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el PLC y el entorno **StxLadder**. Para ello recomendamos leer los siguientes documentos, en el orden detallado a continuación:

1. **STXLADDER-UM**: Manual de Usuario de **StxLadder**.
2. **STXLADDER-PI**: Introducción al lenguaje **Pawn**.

3. Es altamente recomendado complementar esta guía con el **Manual de Programación Pawn del PLC** que incluye una referencia completa de las funciones nativas soportadas por el PLC.

3 Alcance del Documento

Este documento está orientado a comprender como utilizar el lenguaje Pawn en combinación con el lenguaje Ladder. En este documento no se explican las bases del lenguaje Pawn, para ello puede leer el documento **STLADDER-PI**.

4 Requerimientos

Para programar el PLC, es necesario tener instalado, el siguiente software:

1. **StxLadder**: Slicetex Ladder Designer Studio. Disponible en nuestro sitio Web.

Recuerde visitar nuestra página Web en www.slicetex.com para obtener una copia actualizada de este documento, ejemplos y software.

5 Introducción

5.1 Finalidad de la Mezcla de Lenguajes

Los PLC de Slicetex Electronics pueden ser programados en dos lenguajes: Ladder y Pawn.

El lenguaje Ladder es grafico y muy simple de aprender, ya que consiste en la interconexión eléctrica de componentes para implementar la lógica de funcionamiento. Ampliamente adoptado por electricistas, técnicos en la automatización industrial y usuarios de áreas no relacionadas a la informática, como mecánicos y aficionados.

El leguaje Pawn es un lenguaje textual, de 32-bits y con una sintaxis similar al lenguaje de programación C, pero de alto nivel. Este tipo de lenguaje, es el preferido por usuarios relacionados al área de la informática, electrónica, sistemas embebidos y entornos universitarios o de laboratorios.

El usuario puede optar por programar el PLC íntegramente en lenguaje Ladder, en lenguaje Pawn o combinando ambos lenguajes en el mismo proyecto.

La elección del lenguaje es cuestión de preferencias y afinidad.

Sin embargo, al combinar ambos lenguajes se obtiene lo mejor de ambos mundos, el de la programación grafica y el de la programación textual.

En la gran mayoría de los proyectos, el lenguaje Ladder permite implementar una solución en muy poco tiempo, ahorrando el tiempo de escribir a mano y recordar sentencias en lenguaje Pawn.

Pero:

1. ¿ Qué pasa si utilizamos Ladder y necesitamos implementar un algoritmo complejo, que difícilmente puede resolverse en lenguaje grafico ?.
2. ¿ Qué pasa si necesitamos ejecutar funciones no soportadas en Ladder ?.
3. ¿ Y si necesitamos optimizar la velocidad del PLC para operaciones criticas ?.

La respuesta es muy simple, insertar código Pawn en Ladder y así combinar ambos lenguajes.

Con Pawn resolvemos las preguntas anteriores, pero seguimos en el cómodo entorno de programación Ladder.

Pawn aprovecha al máximo las capacidades del PLC y es muy útil disponer de este conocimiento a la hora de programar en Ladder.

Para aquellos que están acostumbrados a los lenguajes textuales tipo Pawn, la posibilidad de combinarlo con el lenguaje grafico Ladder, puede ser una estupenda oportunidad para programar gráficamente sin alejarse de Pawn, pero aprovechando las ventajas que da Ladder en cuanto a simplicidad de aprendizaje.

5.2 Como utilizar Pawn en Ladder

Importante: Es recomendable que lea completamente el capítulo 5 actual, aunque no entienda del todo los conceptos. Más adelante se darán ejemplos completos en el capítulo 6, pag. 11.

Para combinar el lenguaje Pawn con Ladder, es necesario comenzar un proyecto en **StxLadder** utilizando lenguaje Ladder.

Esto significa que no es posible utilizar Ladder en un proyecto exclusivo del lenguaje Pawn.

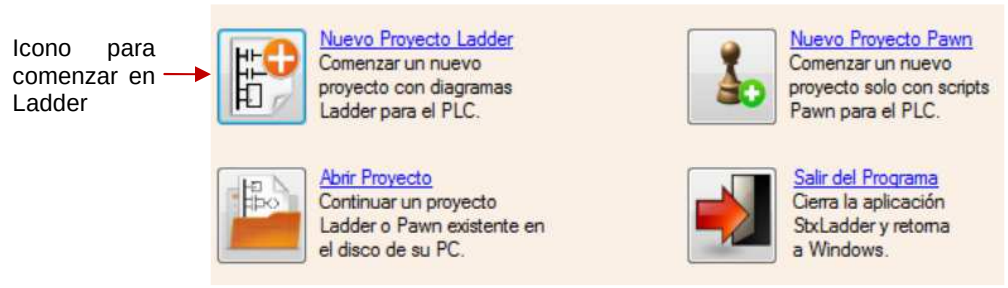


Fig 1: Selección de tipo de proyecto en StxLadder

Hay tres métodos para utilizar lenguaje Pawn en un proyecto Ladder (ver pag. 11):

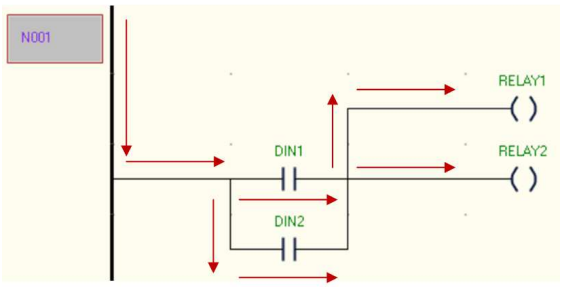
1. Insertar código Pawn en un diagrama Ladder.
2. Llamar a una función Pawn contenida en un archivo SFC (Script Function Code).
3. Agregar un archivo Pawn de extensión ".p" y crear funciones Pawn normalmente.

En las secciones posteriores, entraremos en detalle sobre cada método. El próximo paso que debemos aprender, es como el lenguaje Ladder es "visto" desde el lenguaje Pawn.

5.3 Interfaz entre Ladder y Pawn

Cuando programamos en lenguaje Ladder, todos los componentes insertados en un diagrama están conectados mediante un cable. Por dicho cable circula el flujo de corriente, llamado "**CurrentFlow**", que normalmente si vale "1" ejecuta el componente conectado y si es "0" el componente no se ejecuta.

El siguiente circuito Ladder, nos muestra con flechas en rojo el "Flujo de Corriente". Dependiendo de su valor, se ejecutan o no los componentes conectados (más información en documento **STXLADDER-UM**).



Entonces, si nosotros insertamos un código Pawn conectado a un cable eléctrico, podemos leer el valor del flujo de corriente con la variable "**CurrentFlow**". Dicha variable es "1" si el flujo de corriente esta activo o es "0" si fue detenido por alguna causa (ejemplo, por un componente previamente conectado).

Por ejemplo, el siguiente código Pawn le asigna un valor a la variable "**gi_Temp**" si el flujo de corriente es "1":

```
if(CurrentFlow == 1)
{
   gi_Temp = (gi_Temp * 25 + 87) / 5
}
```

Note que también es posible modificar el valor de "**CurrentFlow**" asignándole un valor "0" luego de la operación, de tal forma que el próximo componente Ladder no sea ejecutado.

Otro aspecto importante a la hora de utilizar código Pawn en Ladder es la modificación de variables.

En Ladder las variables se nombran de la forma ordinaria, con un nombre cualquiera. Pero en Pawn, una variable definida en Ladder se accede con otro nombre diferente.

Por ejemplo, si la variable "**Temp**" del tipo "**Int32**" es definida en Ladder como global. En Pawn debemos accederla anteponiendo un prefijo, seguido de un guion bajo: "**gi**_".

El prefijo "**gi**" significa "global integer 32 bits".

La misma consideración debe hacerse para variables del tipo float, bool, array, etc.

En el caso de las variables tipo "**Bool**" el método de acceso es a través de funciones específicas, ya que Pawn no soporta variables booleanas.

Las variables serán tratadas en la próxima sección.

Las funciones Ladder definidas en diagramas Ladder, pueden llamarse desde Pawn, anteponiendo también un prefijo que se describe más adelante.

Para código Pawn independiente, definido en otras áreas del proyecto, es posible crear variables o funciones sin los prefijos mencionados.

5.4 Aprendiendo de StxLadder

Siempre es bueno tener un experto al lado, por ello, mucho ejemplos para utilizar Pawn en Ladder puede obtenerlos desde el entorno **StxLadder**. Puede ver el código Pawn generado de cada componente haciendo click derecho sobre el componente y seleccionando "**Mostrar código del componente …".**

Nota: Para ver el código Pawn generado, es necesario primero compilar exitosamente el proyecto.

5.5 Acceso a Variables Ladder

Esta sección explica como acceder desde Pawn a variables que fueron definidas en el lenguaje Ladder.

La siguiente tabla muestra las variables soportadas por el lenguaje Ladder, la tabla puede encontrarse más actualizada en el documento **STXLADDER-UM**, allí encontrará más información sobre el tipo de dato permitido.

Tabla: Tipos de datos admitidos por las variables

Tipo	Tamaño	Identificador Pawn	Descripción	Ejemplo
Bool	1-bit	b	Variable booleana.	1
Bool_Array	N x 1-bit	ab	Array con N variables o elementos booleanos.	1,0,1,0,0,1
Int32	32-bits (4-bytes)	i	Variable entera de 32-bits con signo.	5486
Int32_Array	N x 32-bits	ai	Array con N variables o elementos enteros de 32-bits con signo.	5486,78,0,-453
Float	32-bits (4-bytes)	f	Variable de punto-flotante, admite valores decimales.	123.789
String	4-bytes por carácter	s	Permite almacenar una cadena o conjuntos de caracteres que forman un mensaje.	Hola Mundo!

El nombre de una variable Ladder es visible con la siguiente sintaxis en Pawn:

Alcance + Identificador Pawn + _ + Nombre del diagrama + _ + Nombre de variable en Ladder

Donde:

- **Alcance**: Puede ser "**g**" si la variable es global y es "**l**" si la variable es local a un diagrama.
- **Identificador Pawn**: Ver tabla de tipo de datos.
- **Nombre del diagrama**: Es el nombre del diagrama donde fue definida la variable si es local.
- **Nombre de Variable**: El nombre propiamente dicho de la variable en Ladder.

Ejemplos:

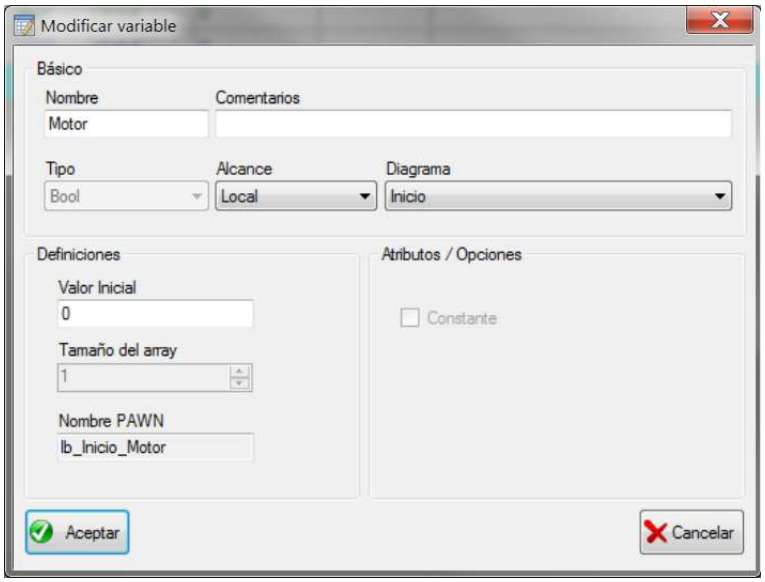
Variable: Nombre "**Status**", tipo **Bool**, definida localmente en diagrama "Inicio.sld".
Nombre en Pawn: **lb_Inicio_Status**

Variable: Nombre "**Temp**", tipo **Int32**, definida globalmente.
Nombre en Pawn: **gi_Status**

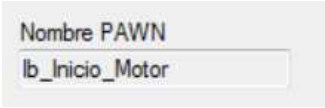
Variable: Nombre "**Samples**", tipo **Int32_Array**, definida globalmente.
Nombre en Pawn: **gai_Samples**

Variable: Nombre "**Volt**", tipo **Float**, definida localmente en diagrama "Principal.sld".
Nombre en Pawn: **lf_Principal_Volt**

Si le es difícil recordar cómo construir el nombre de la variable Ladder para accederla desde Pawn, puede ir a la tabla de variable en el entorno **StxLadder** desde el menú "**Proyecto > Tabla de variables**". Luego seleccione la variable en cuestión y haga click en el botón "**Modificar**". Aparecerá la siguiente ventana con información de la variable:

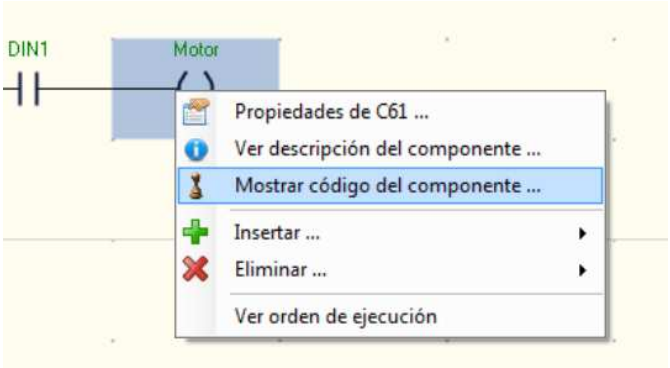


Ubique el texto "**Nombre PAWN**" y debajo del mismo aparece el nombre de la variable en Pawn:



En este caso la variable "**Motor**" del tipo Bool, fue definida en el diagrama "Inicio.sld". Su nombre en Pawn según muestra la pantalla es "**lb_Inicio_Motor**".

Otra forma de ver como el entorno Ladder define variables o crea código Pawn, es utilizando la opción "**Mostrar código del componente…"** al hacer click derecho sobre el mismo en el diagrama:



En este caso se aplica el componente "**Bobina normal abierta**" sobre la variable "**Motor**" del tipo **Bool** y el código Pawn generado es:

Código PAWN del componente C61 (CoilNo)

```
1  // -----------------------------------------------
2  // RECUERDE COMPILAR EL PROYECTO PARA VER EL
3  // CODIGO ACTUALIZADO DEL COMPONENTE
4  // -----------------------------------------------
5
6  //
7  // COMPONENT CoilNo (InicioC61) CODE: BEGIN
8  // PRIORITY  2
9  //
10
11 if(CurrentFlow)
12 {
13    lBoolSet(lb_Inicio_Motor)
14 }
15 else
16 {
17    lBoolClr(lb_Inicio_Motor)
18 }
19
20 //
21 // COMPONENT CoilNo (InicioC61) CODE: END
22 //
23
```

Notar como el componente utiliza la variable del flujo de corriente "**CurrentFlow**" para luego escribir un "1" o un "0" a la variable "**Motor"**.

La variable "**Motor"** del tipo Bool se accede mediante funciones en Pawn, ya que en Pawn no existe el tipo Bool.

5.5.1 Acceso a Variables tipo Bool desde Pawn

Para modificar o leer el valor de una variable tipo "**Bool**" o "**Bool_Array**" desde Pawn, es necesario utilizar funciones, ya que no es posible tratarlas como "variables normales".

Esto quiere decir, que para leer el valor de una variable debe utilizarse una función que haga de intermediario.

Por ejemplo, si queremos asignar el valor "1" a la variable "**lb_Inicio_Motor**", no podemos hacer:

```
lb_Inicio_Motor = 1                    // MAL !!!
```

Debemos utilizar una función que establezca un "1" se la siguiente forma:

```
lBoolSet(lb_Inicio_Motor)              // CORRECTO
```

Si la variable es global, debe utilizarse la función gBoolSet().

A continuación listamos las funciones Pawn para acceder a variables tipo "**Bool**":

Funciones para variables del tipo global:
Var = Nombre de la variable en Pawn.

- **gBoolSet(Var)** : Escribe un "1" en variable.
- **gBoolClr(Var)** : Escribe un "0" en variable.
- **gBoolToggle(Var)** : Invierte el valor de la variable.
- **gBoolRead(Var)** : Lee el valor de la variable y devuelve un "0" o un "1".

```
Ejemplo: if(gBoolRead(gb_Status) == 1) CurrentFlow = 1
```

Funciones para variables del tipo local:
Var = Nombre de la variable en Pawn.

- **lBoolSet(Var)** : Escribe un "1" en variable.
- **lBoolClr(Var)** : Escribe un "0" en variable.
- **lBoolToggle(Var)** : Invierte el valor de la variable.
- **lBoolRead(Var)** : Lee el valor de la variable y devuelve un "0" o un "1".

```
Ejemplo: lBoolSet(lb_Inicio_Motor)
```

A continuación listamos las funciones Pawn para acceder a variables tipo "**Bool_Array**":

Funciones para variables Array (mismas funciones para array global o local):
Array = Nombre del Array en Pawn
Item = Numero de elemento del Array.

- **aBoolSet(Array, Item)** : Escribe un "1" en el elemento "Item" del array.
- **aBoolClr(Array, Item)** : Escribe un "0" en el elemento "Item" del array.
- **aBoolToggle(Array, Item)** : Invierte el valor del elemento "Item" del array.
- **aBoolRead(Array, Item)** Lee el valor del elemento "Item" del array y devuelve un "0" o "1".

```
Ejemplo: CurrentFlow = aBoolRead(lab_Inicio_Flags, 1)
```

Finalmente, listamos las funciones Pawn para acceder a variables tipo "**Bool**" que acceden a periféricos. Por ejemplo las variables de entradas discretas DIN1, DIN2, DIN3, etc, o de las salidas discretas RELAY1, RELAY2, RELAY3 …., DOUT1, DOUT2, etc.

Funciones para variables Bool en memoria de Periféricos:
Var = Nombre de la variable en Pawn.

- **pBoolSet(Var)** : Escribe un "1" en variable.
- **pBoolClr(Var)** : Escribe un "0" en variable.
- **pBoolToggle(Var)** : Invierte el valor de la variable.
- **pBoolRead(Var)** : Lee el valor de la variable y devuelve un "0" o un "1".

```
Ejemplo: pBoolClr(gb_RELAY4)
```

Nota: Las variables de periféricos son globales.

5.6 Acceso a Funciones Ladder

Si tenemos definido en el proyecto un diagrama-función Ladder, podemos llamarla desde Pawn con la siguiente sintaxis:

fl + _ + Nombre Diagrama de Función + (InVal)

Donde:

- **fl** = Prefijo, significa función Ladder.
- **Nombre de Diagrama** = El nombre del diagrama donde se define la función Ladder.
- **InVal** = Valor del tipo **Int32** a pasar como argumento en la función.

Por ejemplo, si la función Ladder se llama "**Motor.sld**", en el lenguaje Pawn podemos llamarla de la siguiente forma:

```
fl_Motor(100)
```

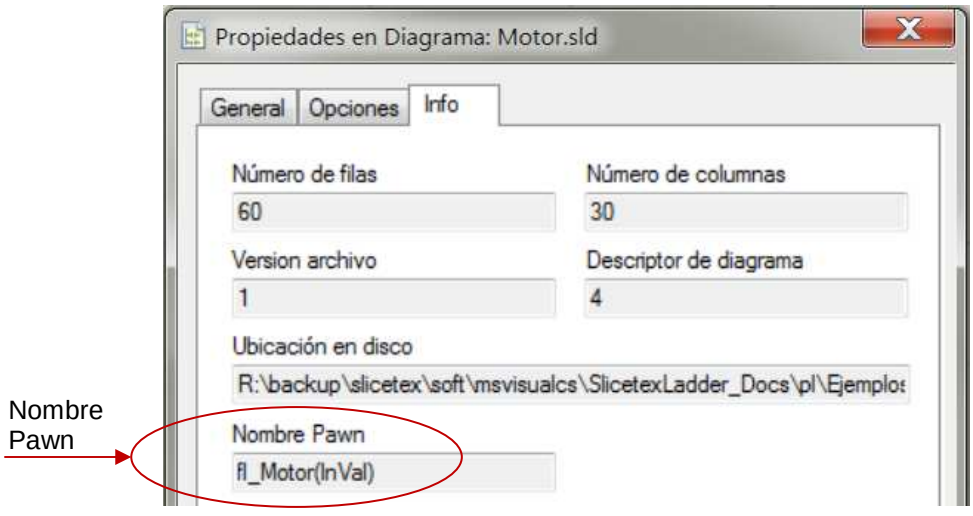
En el caso anterior, le pasamos a la función el valor "100" como argumento. La función puede o no utilizar dicho valor, esto depende del usuario.

Dentro del diagrama de la función Ladder "**Motor.sld**" el parámetro pasado como argumento es visible en la variable **local** llamada "**InVal**", del tipo **Int32**. Esta variable está disponible para todo componente que necesite una variable Int32 para lectura (ya que es de solo lectura).

En Pawn, la variable **InVal**, pasa a llamarse: **li_Motor_InVal**. Ver página 6.

Es posible ver el nombre Pawn de una diagrama-función Ladder, haciendo click derecho sobre el archivo del diagrama en el "**Explorador de Proyecto**" y seleccionando "**Propiedades …**" en el menú desplegable.

Luego vamos a la pestaña "**Info**" de la ventana abierta y buscamos el casillero "**Nombre Pawn**":



Con la instrucción "**return**" en Pawn podemos retornar de un diagrama-función Ladder. Podemos devolver un valor del tipo Int32 también, por ejemplo "**return 123**" o "**return gi_Temp**".

6 Métodos para Utilizar Pawn en Ladder

Hay tres métodos para utilizar lenguaje Pawn en un proyecto Ladder:

1. Insertar código Pawn en un diagrama Ladder. Ver pag. 11.
2. Llamar a una función Pawn contenida en un archivo SFC (Script Function Code). Ver pag. 22.
3. Agregar un archivo Pawn de extensión ".p" y crear funciones Pawn normalmente. Ver pag. 36.

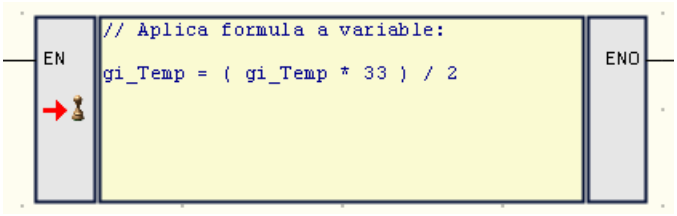
6.1 Insertar Código Pawn en un Diagrama Ladder

Este es el método mas practico para operaciones sencillas, donde necesitamos mediante una o más líneas de programación implementar una lógica con Pawn dentro de un diagrama Ladder.

El código se inserta mediante un componente Ladder especial, que se cablea naturalmente como cualquier otro componente Ladder. Pero al ejecutarse este componente, se ejecuta el código Pawn introducido por el usuario.

Actualmente hay dos componentes para insertar código Pawn, que se encuentran en el selector de componentes, dentro del grupo "**Pawn**":

Componente **PawnQuickInsert**:



Componente **PawnInsert**:

El componente **PawnQuickInsert**, está pensado para insertar código Pawn de unas cuantas líneas y al mismo tiempo visualizarlo en el diagrama. El componente se ejecuta si el puerto a la entrada "EN" tiene un flujo de corriente iguala a "**1**".

El componente **PawnInsert**, está pensado para insertar un código extenso y no muestra su contenido en el diagrama. Este componente se ejecuta **independientemente** del valor del flujo de corriente.

Debemos aclarar que para ambos componentes no hay límite en la cantidad de código introducido, pero para grandes cantidades de código, conviene utilizar otros métodos o funciones Pawn.

La inserción de código es útil cuando debemos implementar operaciones que no se repiten, por ejemplo modificar una variable o ejecutar instrucciones Pawn específicas. Pero no es práctico cuando es necesario aplicar el mismo algoritmo en distintas partes del diagrama.

Cuando requiere realizar la misma acción varias veces, conviene agrupar el código en una función Pawn en un script .p, y luego llamar a esa función a través de la inserción de código tantas veces como sea necesario.

6.1.1 Ejemplo Práctico

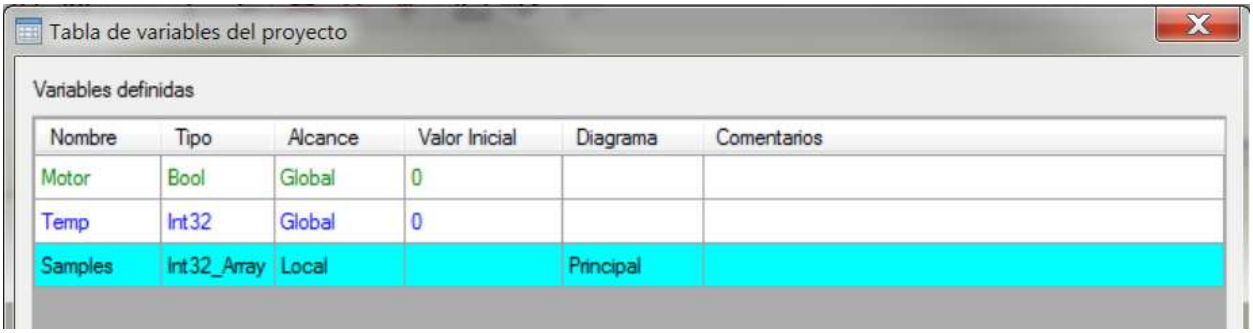
A continuación crearemos un proyecto en Ladder llamado "**InsertPawn**". Puede descargarlo desde nuestro sitio Web para utilizar como referencia. El ejemplo ilustra cómo utilizar la inserción de código Pawn.

El ejemplo tiene por objetivo:

- Activar varios relays al mismo tiempo cuando la entrada DIN1 = 1.
- Llamar a un diagrama-función Ladder cuando la entrada DIN2 = 1.
- Adquirir una muestra de cada entrada analógica, guardarlas en un array y aplicarles una operación matemática cuando la entrada discreta DIN3 = 1.
- Leer la entrada DIN4 y activar RELAY6 utilizando código Pawn.

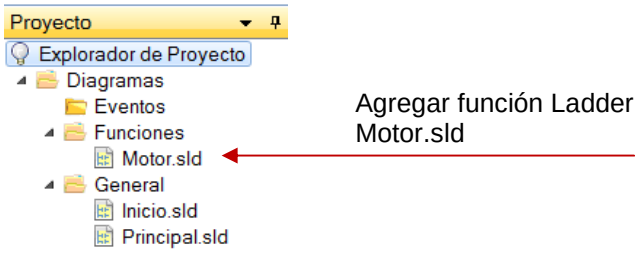
Nota: Si su PLC no contiene entradas analógicas, procure entender el concepto, ya que puede aplicarlo para otras operaciones.

Para comenzar, primero comience un proyecto nuevo en Ladder y nómbrelo como "**InsertPawn**". Una vez creado el proyecto, defina las siguientes variables desde "**Proyecto > Tabla de variables**":



Donde **Motor**, es una variable Bool global, **Temp** es una variable Int32 global, y **Samples** es un array Int32_Array local del diagrama Principal.

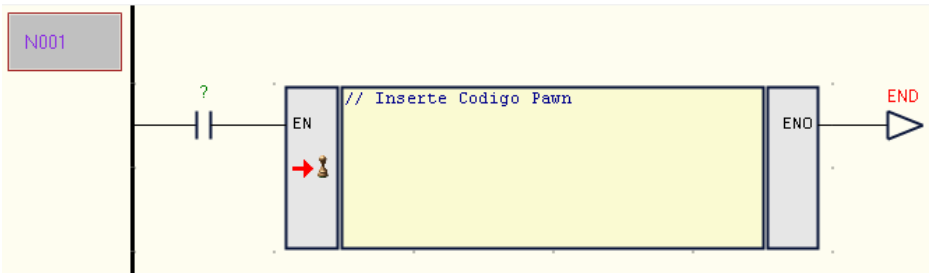
Desde el "**Explorador de Proyecto**", agregue en la carpeta "**Funciones**" un diagrama llamado "**Motor**":



El diagrama **Inicio.sld** del proyecto no será utilizado, por lo tanto puede dejarlo vacio.

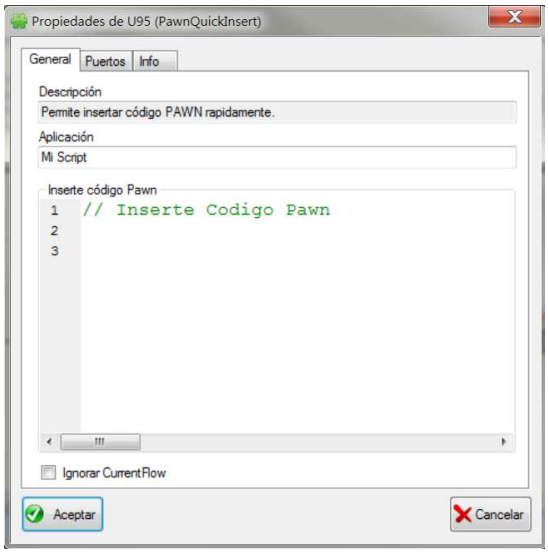
Entrada DIN1:

Abra el diagrama **Principal.sld**, y agregue los siguientes componentes en la Network N001:



El primer componente de la izauiqerda, es un componente contacto normal abierto (**SwitchNO**). El mismo leerá la entrada DIN1 del PLC. Si DIN1=1, se procede a ejecutar el componente de la derecha "**PawnQuickInsert**" que permite insertar código Pawn.

Seleccione en el componente contacto normal abierto la variable DIN1. Luego entre a las propiedades del componente **PawnQuickInsert** haciendo doble-click sobre el mismo. De esta forma podrá introducir código Pawn en la siguiente ventana:



El código Pawn que insertemos debe activar 4 salidas o relays del PLC cuando se ejecute. Por lo tanto, como los RELAYS son variables **Bools** globales dentro de la memoria de periféricos (ver pag. 8 en 5.5.1) debemos utilizar el siguiente código para activar los RELAY1 al RELAY4:

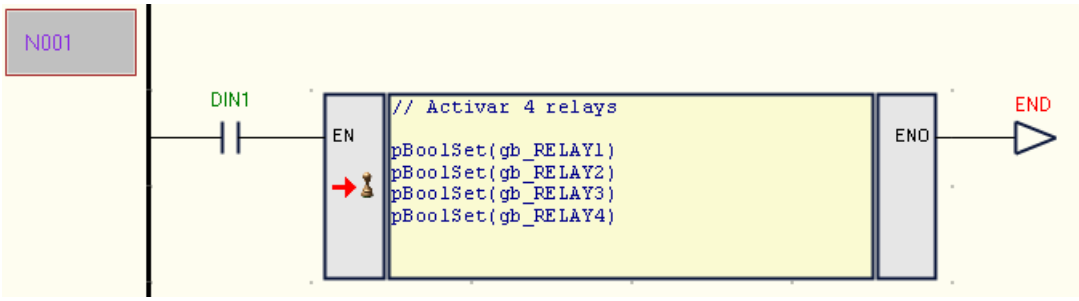
```
pBoolSet(gb_RELAY1)
pBoolSet(gb_RELAY2)
pBoolSet(gb_RELAY3)
pBoolSet(gb_RELAY4)
```

Es decir, dentro del componente agregamos:

Inserte código Pawn

```
1  // Activar 4 relays
2
3  pBoolSet(gb_RELAY1)
4  pBoolSet(gb_RELAY2)
5  pBoolSet(gb_RELAY3)
6  pBoolSet(gb_RELAY4)
7
```

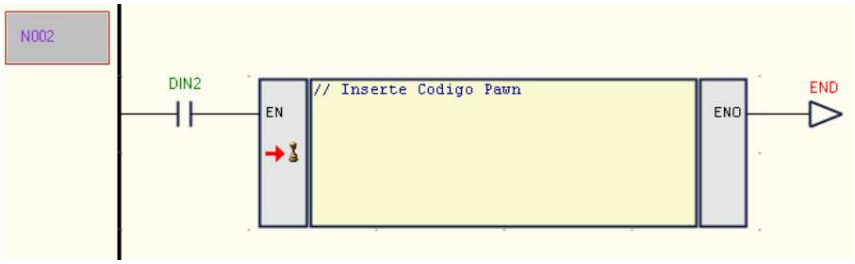
Luego presionamos el botón **Aceptar** y el circuito resultante es:



Como se observa en la figura superior, cuando DIN1=1, el puerto "**EN**" del componente "**PawnQuickInsert**" será "1" y se ejecutara el código mostrado. Notar como el nombre de los relays tiene el prefijo "**gb**_" en Pawn. Esto se debe a que son variable globales (ver pag. 6).

Entrada DIN2:

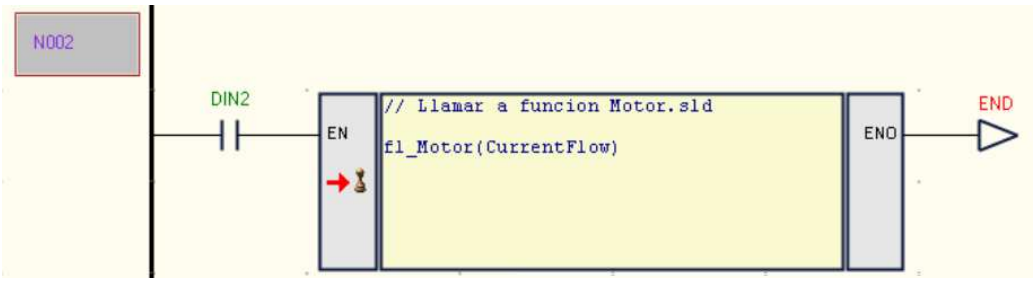
El próximo paso consistirá en llamar al diagrama-función **Motor.sld** cuando la entrada discreta DIN2=1. Pasaremos como argumento un valor "**1**" o "**0**" según el estado de la entrada discreta. En la función Motor.sld modificáremos el valor de la variable global tipo bool "**Motor**" de acuerdo al argumento pasado a la función. Agregue los siguientes componentes a la Network N002 del diagrama **Principal.sld**:



Agregue el siguiente código al componente **PawnQuickInsert**:

```
// Llamar a funcion Motor.sld

fl_Motor(CurrentFlow)
```

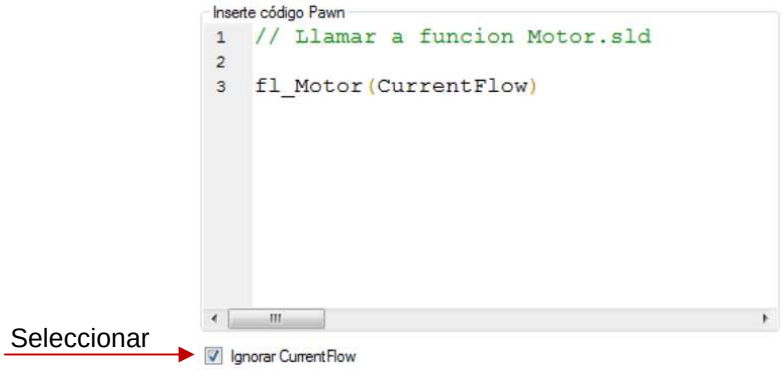
Resultando:



El componente se ejecuta cuando DIN2=1, ya que **CurrentFlow**=1 y luego llama a la funcion-ladder **fl_Motor()** pasando como argumento la variable **CurrentFlow**. Ver página 10 en sección 5.6.

¿ Pero qué pasa si la variable CurrentFlow es 0 ?. El componente no se ejecuta y no se llama a la función **fl_Motor()**. Este comportamiento no sería el deseado para nuestra aplicación, ya que debemos pasar un 0 o 1 para alterar el valor de la variable **Motor**.

Entonces, para ignorar el flujo de corriente y ejecutar el código de del componente independientemente del valor de CurrentFlow, debemos ir a las propiedades del componente y seleccionar el check-box "**Ignorar CurrentFlow**":



Seleccionado el check-box, independientemente de si DIN2=1 o DIN2=0, el componente **PawnQuickInsert** se ejecutará siempre ya que ignoramos el flujo de corriente almacenado en **CurrentFlow**. Pero note, que igualmente podemos leer y/o alterar el valor de **CurrentFlow**.

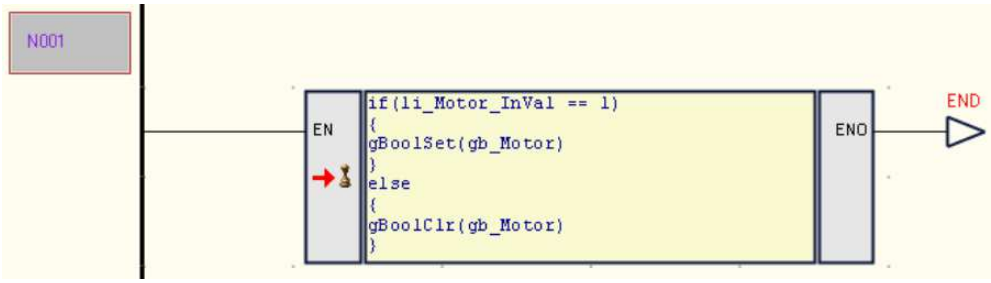
Como practica, puede ver el código generado por **StxLadder**, cuando "**Ignorar CurrentFlow"** esta activado y cuando esta desactivado. Esto es posible seleccionando "**Mostrar código del componente ...**".

El próximo paso es abrir el diagrama-función "**Motor.sld**". Aquí debemos escribir un "0" en la variable global **Motor** si el argumento de entrada de la función es 0, y escribir un 1 si el argumento es 1.

Agregue un componente PawnQuickInsert en la primera network del diagrama, con el código siguiente:

```
if(li_Motor_InVal == 1)
{
      gBoolSet(gb_Motor)
}
else
{
      gBoolClr(gb_Motor)
}
```

Resultando el siguiente circuito:



La línea:

```
if(li_Motor_InVal == 1)
```

Comprueba si el argumento de entrada de la función, almacenado en la variable "InVal" del tipo Int32, de alcance local y perteneciente al diagrama Motor.sld es igual a "1".

Si es asi, escribe un 1 a la variable global Motor del tipo Bool con:

```
      gBoolSet(gb_Motor)
```

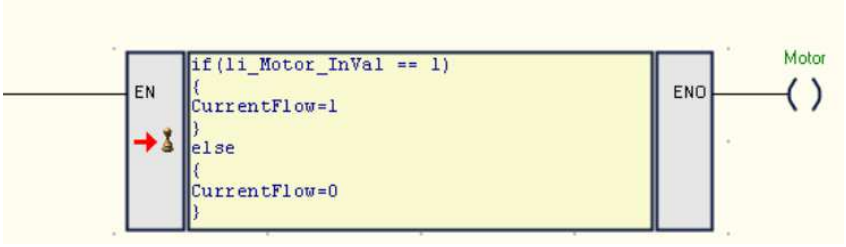
Caso contrario, escribe un 0 con el siguiente código:

```
      gBoolClr(gb_Motor)
```

Ver página 8, sección 5.5.1.

Importante: Como **PawnQuickInsert** está conectado directo a la barra de alimentación, siempre se ejecuta ya que CurrentFlow es 1 en ese punto.

Otro circuito equivalente al presentado, podría ser el siguiente:



Donde el código Pawn insertado altera el valor del flujo de corriente en **CurrentFlow**, dejando el trabajo al componente Bobina para escribir un 0 o 1 a la variable **Motor**:

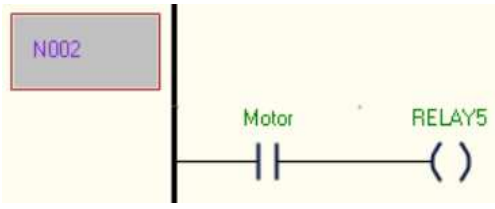
```
// Código componente PawnQuickInsert
if(li_Motor_InVal == 1)
{
      CurrentFlow=1
}
else
{
      CurrentFlow=0
}
```

Si compilamos el código del último circuito, podremos observar el código del componente Bobina generado por **StxLadder**, el cual es el siguiente:

```
// Código componente CoilNo
if(CurrentFlow)
{
   gBoolSet(gb_Motor)
}
else
{
   gBoolClr(gb_Motor)
}
```

Notar como de acuerdo al valor del flujo de corriente (que previamente modificamos desde el componente **PawnQuickInsert**) se escribe un 0 o un 1 a la variable **Motor** desde el componente Bobina.

Finalmente, en la Network N002 agregue los siguientes componentes que nos permitirán comprobar si la variable Motor fue escrita con un 0 o un 1:



En el circuito anterior, si la variable **Motor** = 1, se activa el **RELAY5**, caso contrario, el **RELAY5** es desactivado.

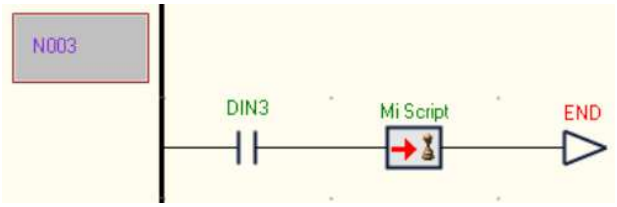
Para resumir, en el proyecto, cuando DIN2=1 o DIN2=0, se llama al diagrama-función **Motor.sld**, se pasa como argumento un 0 o 1, el cual escribe la variable Motor. Luego, el RELAY5 copia el estado de la variable **Motor**, activándose o desactivándose.

Entrada DIN3:

Ahora vamos a leer las entradas analógicas utilizando código Pawn insertado.

El objetivo es: Adquirir una muestra de cada entrada analógica del PLC, guardarlas en un array y aplicarles una operación matemática cuando la entrada discreta DIN3 = 1.

Para ello debemos crear el siguiente circuito en el diagrama **Principal.sld**, network N003:



En el **Manual de Programacion Pawn del PLC**, puede encontrar funciones nativas para leer entradas analógicas de su PLC. En el caso del PLC modelo STX8081, las funciones VinRead() leen las entradas analógicas. Si su PLC no posee entradas analógicas, no importa, lo importante es que entienda el concepto.

Agregue el siguiente código al componente **PawnInsert**:

```
if(CurrentFlow)
{
	// Leer entradas analógicas.
	lai_Principal_Samples[0] = Vin1Read()
	lai_Principal_Samples[1] = Vin2Read()
	lai_Principal_Samples[2] = Vin3Read()
	lai_Principal_Samples[3] = Vin4Read()

	// Aplicar factor.
	lai_Principal_Samples[0] *= 5
	lai_Principal_Samples[1] *= 15
	lai_Principal_Samples[2] *= 25
	lai_Principal_Samples[3] *= 35
}
```

Se eligió el componente **PawnInsert** por una cuestión de preferencia estética simplemente. Note que en este componente, comprobamos el valor de **CurrentFlow** antes de leer las entradas, ya que por defecto el componente ejecuta el código del usuario sin comprobar el flujo de corriente. Esto es necesario para leer las entradas analógicas solo si DIN3=1, momento en el cual **CurrentFlow** será “1” para el componente **PawnInsert** debido a su conexión.

Según el **Manual de Programación Pawn del PLC** modelo STX8081, Vin1Read() entrega la lectura digital del voltaje presente en la entrada analógica 1. Lo mismo para Vin2Read() con la entrada analógica 2, y así sucesivamente.

La lectura de cada canal se guarda en un elemento tipo **Int32** del Array nombrado "**Samples**" en Ladder. Observar que el array es local y pertenece al diagrama **Principal.sld**, de allí su prefijo. Ver pag. 6, sección 5.5.

La línea de código:

```
lai_Principal_Samples[0] *= 5
```

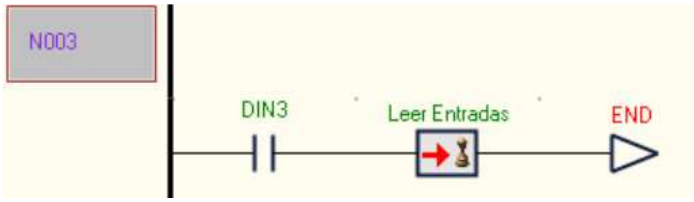
Es equivalente a:

```
lai_Principal_Samples[0] = lai_Principal_Samples[0] * 5
```

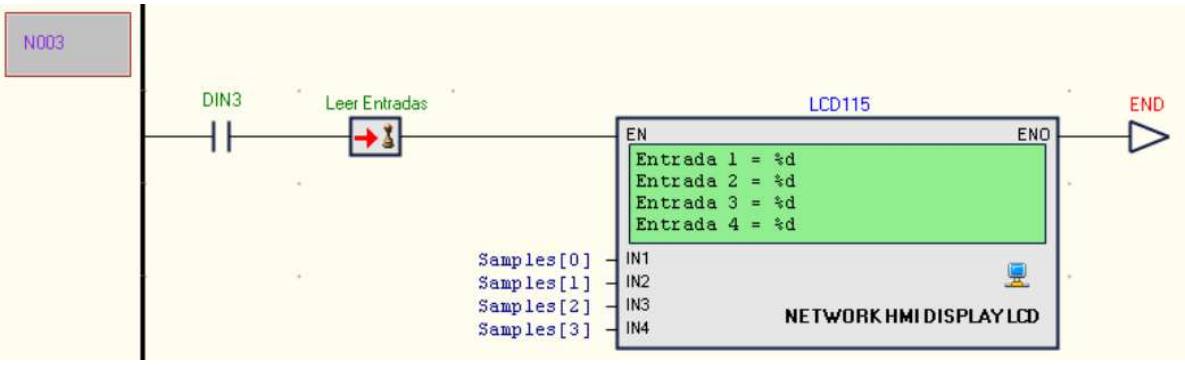
Que significa, al valor del elemento 0 en el array Samples, multiplicarlo por 5 y luego guardar el resultado en el mismo elemento. En otras palabras, multiplicamos por 5 (un factor) el valor de la muestra analógica almacenada y la guardamos nuevamente.

Como el array es local, sus valores se conservan solo dentro del diagrama **Principal.Sld**, y se destruyen en cada SCAN CYCLE del PLC.

En el campo "**Aplicación**" del componente **PawnInsert**, dentro de sus propiedades, podemos escribir "**Leer Entradas**" y el circuito resultante será:



En PLC con display LCD o a través del software Virtual-HMI (www.slicetex.com/virtualhmi) puede ver el valor de los 4 elementos del array colocando un componente como se muestra a continuación:



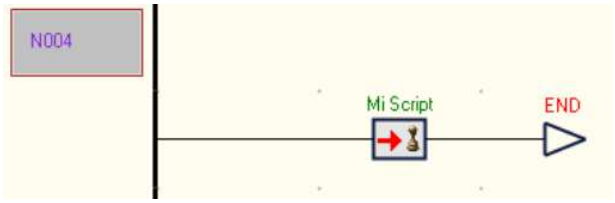
Al mostrar los elementos del array, está observando indirectamente el valor de las entradas analógica.

Note como integrar código Pawn y componentes Ladder, es una herramienta simple y muy útil.

Entrada DIN4:

Para completar el ejemplo, es necesario cumplir con el siguiente objetivo: Leer la entrada DIN4 y activar RELAY6 utilizando código Pawn.

Agregar el componente **PawnInsert** como se muestra a continuación en el diagrama Principal.sld, Network N004:



Dentro del componente, agregar el siguiente código:

```
// Comprobar flujo de corriente (opcional).
if(CurrentFlow)
{
	// Leer entrada DIN4.
	if(pBoolRead(gb_DIN4))
	{
		// Activar RELAY6.
		pBoolSet(gb_RELAY6)
	}
	else
	{
		// Desactivar RELAY6.
		pBoolClr(gb_RELAY6)
	}
}
```

Se comprueba el flujo de corriente al comenzar, solo por formalismo. Ya que como dijimos, cuando un componente se conecta directo a la barra de alimentación, **CurrentFlow** siempre es “1”. Por lo tanto, en este caso siempre se ejecuta este componente debido a la posición donde está colocado. Pero si está conectado luego de otro componente, el flujo de corriente puede cambiar y es decisión nuestra comprobarlo o no.

La línea:

```
	if(pBoolRead(gb_DIN4))
```

Comprueba si la variable global DIN4 del tipo bool en memoria de periféricos es “1”, en caso afirmativo, activa el RELAY6, que también es una variable de periféricos, global y del tipo Bool mediante el sig. Código:

```
pBoolSet(gb_RELAY6)
```

Para desactivar el relé, utiliza la función pBoolClr().

Ver funciones en página 8, sección 5.5.1.

6.1.2 Acceso Inmediato

Cuando leemos entradas o salidas discretas mediante las funciones para acceder a la memoria de periféricos, no actuamos directamente sobre el hardware, sino que leemos un área de memoria que se actualiza en cada SCAN CYCLE.

Si necesitamos conocer el valor inmediato de una entrada discreta DIN, o actuar de manera inmediata sobre una salida discreta RELAY o DOUT, el punto mismo de la inserción de código, podemos ver el código que utiliza **StxLadder** en componentes tipo contactos o bobinas, cuando el check-box "**Acceso inmediato**" esta activado. Luego podemos copiarlo a nuestro código Pawn

6.1.3 Conclusiones

Vimos que la inserción de código Pawn en Ladder es una herramienta muy útil para exprimir al máximo las capacidades del PLC y afinar la programación del proyecto utilizando código específico para determinadas tareas.

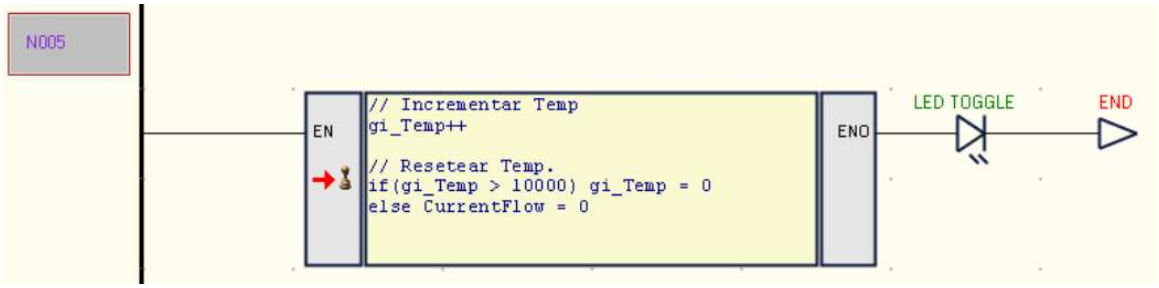
Explore **StxLadder**, lea manuales y practique los ejemplos. Es la mejor manera de aprender.

6.1.4 Código Extra

Si agrega el siguiente código en la Network N005, de **Principal.sld**, la variable Ladder "**Temp**" se incrementará en cada SCAN CYCLE. Cuando sea mayor a 10000, se conmuta el LED debug del PLC:

```
// Incrementar Temp
gi_Temp++

// Resetear Temp.
if(gi_Temp > 10000) gi_Temp = 0
else CurrentFlow = 0
```

El circuito es el siguiente:



El componente "LED TOGGLE" solo se ejecutara si **CurrentFlow** es "1". Eso ocurre solamente cuando "**Temp**" es mayor a 10000. Pero cuando es menor, se altera **CurrentFlow** para que sea 0.

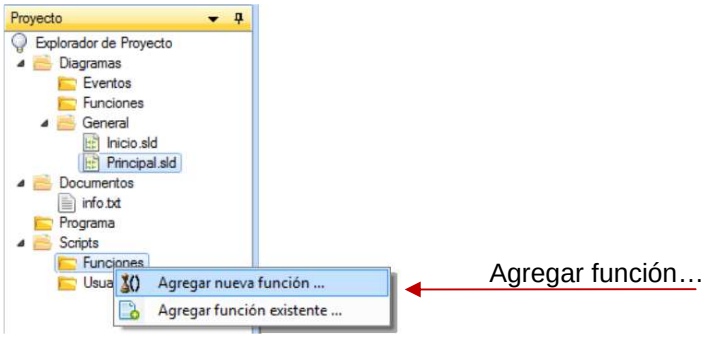
6.2 Utilizar Función Pawn SFC (Script Función Code)

El segundo método que presentaremos, nos permite de una forma simple y organizada crear una función Pawn, a la cual también es posible pasarles variables desde Ladder sin complicaciones.

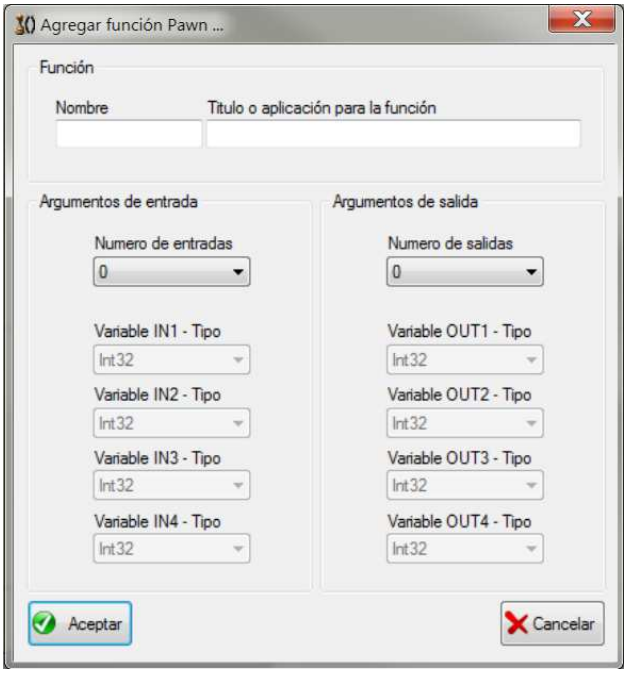
A diferencia de la inserción directa de código Pawn (vista en la sección 6.1, página 11) una función Pawn nos permite reutilizar el código escrito, ya que se escribe una sola vez y puede ser llamado desde cualquier diagrama o archivo.

Por lo tanto, para código que se llamará desde diferentes lugares del proyecto, las funciones SFC son una excelente alternativa.

Un archivo SFC (Script Function Code) se define en la carpeta "**Scripts > Funciones**" desde el "**Explorador de Proyecto**", como se muestra en la siguiente figura:



Al agregar la función, aparece la siguiente ventana:



Hay dos áreas importantes en la ventana, el área relacionada al nombre de la función y el área de los argumentos de entrada y salida de la función.

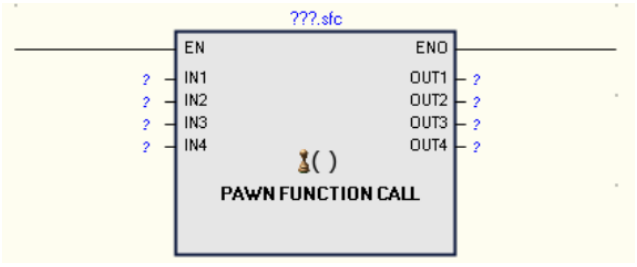
En "**Nombre**" colocaremos el nombre visible de la función en Ladder. Adicionalmente es posible colocarle un titulo o descripción a la función.

En argumento de entrada y salida, debemos seleccionar el número de los mismos. Utilice 0, si no utiliza entradas y salidas para la función.

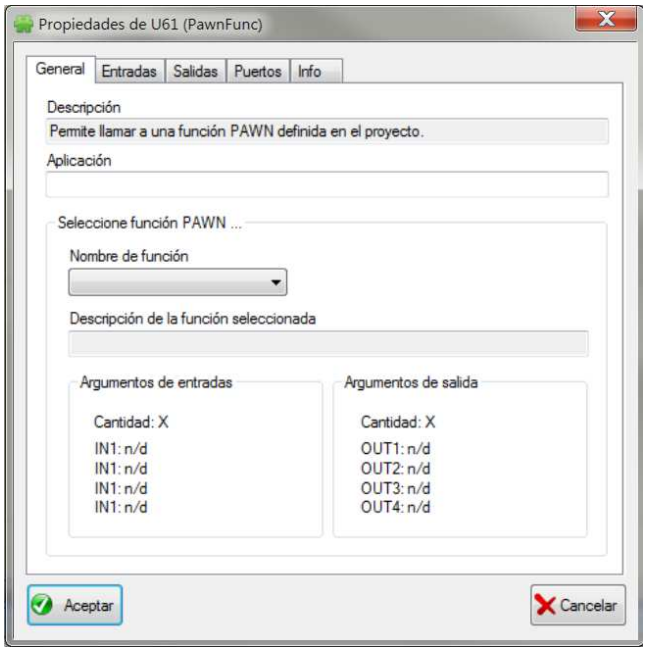
Cada función puede recibir hasta 4 parámetros de entrada y/o salida. En caso de seleccionar argumentos, debe especificar el tipo de variable a ingresar (Int32, Int32_Array, etc).

Dentro del código de la función, podrá utilizar los argumentos de entrada o salida, a través de los nombres IN1, IN2…IN4, OUT1, OUT2…OUT4.

Finalmente, para llamar a la función debe utilizar el componente "**PawnFunc**" que se muestra a continuación.



Desde las propiedades del componente, puede seleccionar el archivo SFC con la función Pawn creada. A través de los puertos de entrada y salida puede pasar los parámetros mediante variables definidas en el proyecto Ladder:



6.2.1 Ejemplo Práctico

Para ilustrar el uso de funciones Pawn SFC, realizaremos el siguiente ejemplo:

- Cuando DIN1=1, muestrearemos 4 entradas analógicas del PLC y depositaremos sus valores en variables de salida.
- Cuando DIN2=1, sumar dos variables enteras mediante una función SFC y luego mostrar en display LCD.
- Cuando DIN3=1, activaremos una salida del PLC que depende del valor pasado como argumento a la función SFC.

Nota:

- Si su PLC no contiene entradas analógicas, procure entender el concepto, ya que puede aplicarlo para otras operaciones.
- El presente ejemplo puede descargarse desde nuestro sitio Web.

Para comenzar, primero comience un proyecto nuevo en Ladder y nómbrelo como "**PawnFuncSFC**". Una vez creado el proyecto, defina las siguientes variables desde "**Proyecto > Tabla de variables**":

Nombre	Tipo	Alcance	Valor Inicial	Diagrama	Comentarios
Sample1	Int32	Global	0		
Sample2	Int32	Global	0		
Sample3	Int32	Global	0		
Sample4	Int32	Global	0		
Motor	Int32	Global	0		
Add	Int32	Global	0		

Donde **Sample1**, **Sample2**, **Sample3**, **Sample4**, **Motor** y **Add** son variables globales del tipo **Int32**.

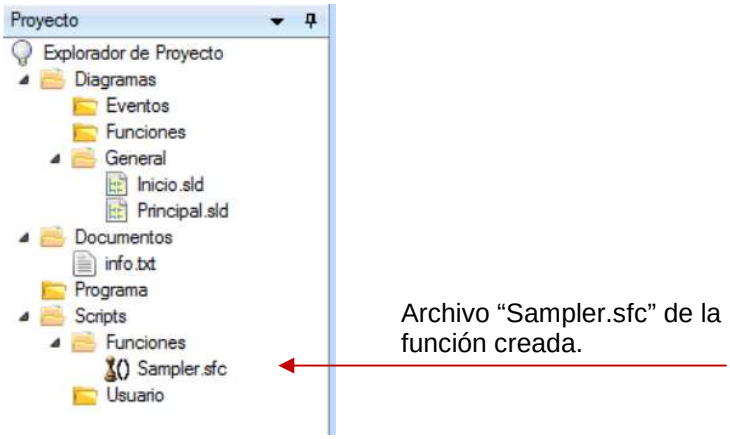
El proyecto utilizará solo el diagrama **Principal.sld**.

Entrada DIN1:

La primera función que implementaremos tendrá como objetivo muestrear 4 entradas analógicas y depositar sus valores en las variables Sample1, Sample2, Sample3 y Sample4.

En el **Explorador de Proyecto**, agregue la una función Pawn en la carpeta "**Script \ Funciones**". Nómbrela como "**Sampler**" y seleccione 4 argumentos de salida tipo Int32 como se muestra en la siguiente figura:

Luego presione el botón **Aceptar**.

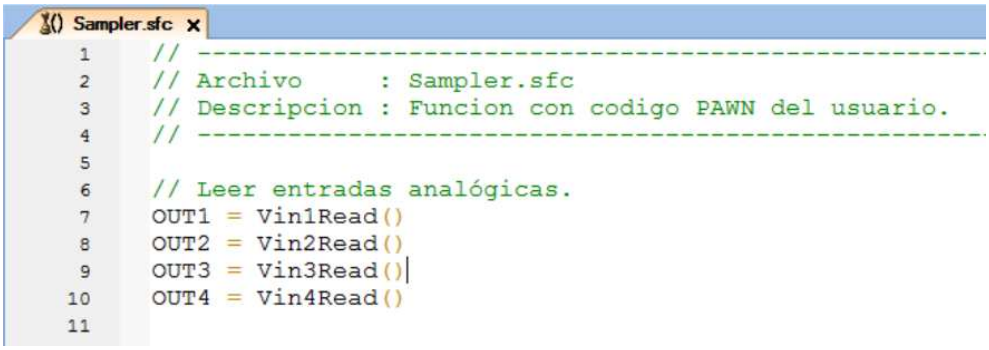


Hacer doble-click sobre el archivo **Sampler.sfc** para abrir el editor y poder ingresar código Pawn.

Luego en el editor, ingresar el siguiente código:

```
// Leer entradas analógicas.
OUT1 = Vin1Read()
OUT2 = Vin2Read()
OUT3 = Vin3Read()
OUT4 = Vin4Read()
```

El código en el archivo **Sampler.sfc** se muestra en **StxLadder** en la sig. Figura:



Dentro del código de un archivo SFC, los argumentos de entrada se nombran como **INx** y los argumentos de salida se nombran como **OUTx**, donde "**x**" es el número de argumento.

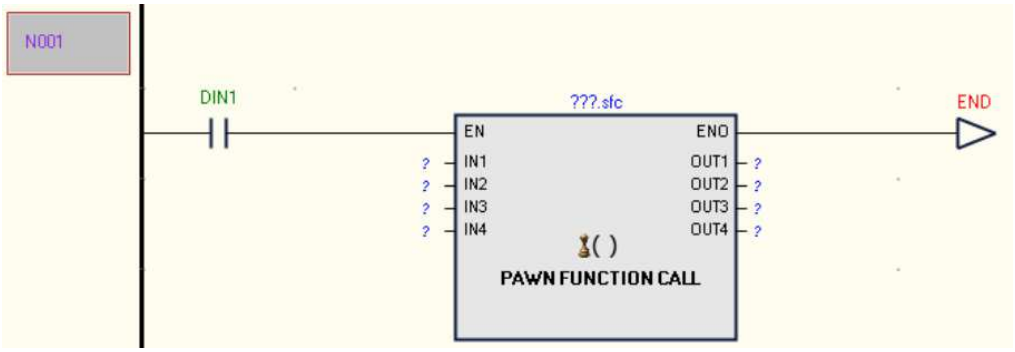
El siguiente código, indica que en el argumento de salida **OUT1** debemos colocar el valor de la entrada analógica VIN1 del PLC, que es retornada por la función **Vin1Read().**

```
OUT1 = Vin1Read()
```

Puede encontrar la descripción de la función **Vin1Read()** en el Manual de **Programación en Lenguaje Pawn** del PLC. Si el PLC no posee entradas analógica, solo debe saber que **Vin1Read()** devuelve un entero **Int32** con el valor digital de la entrada analógica.

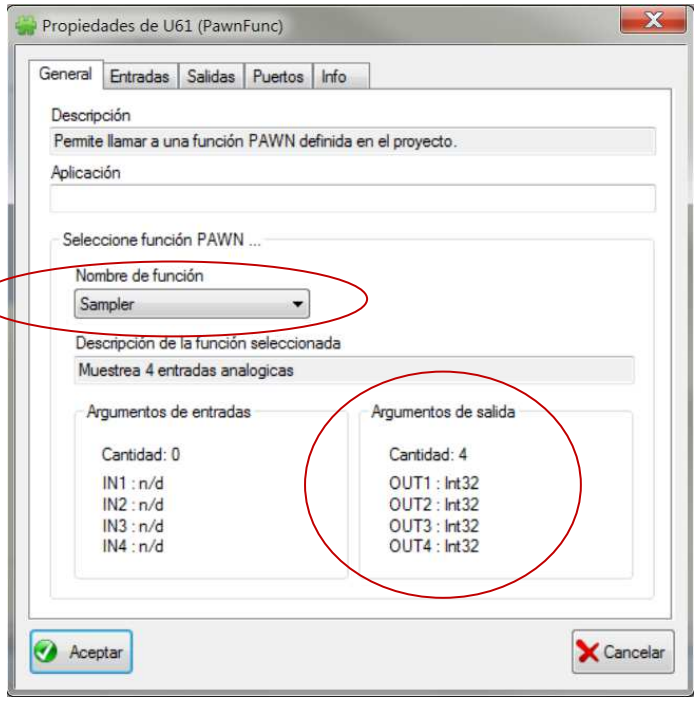
La misma explicación para las salidas OUT2, OUT3 y OUT4 respectivamente.

Ahora vamos a llamar a la función SFC creada. Para ello inserte en el diagrama **Principal.sld**, en la Network **N001**, el siguiente circuito:



Luego del contacto que lee la entrada **DIN1** del PLC, se encuentra el componente "**PawnFunc**" que llama a una función Pawn SFC. Entre a las propiedades del componente y seleccione la función Pawn para muestrear las entradas analógicas, llamada **Sampler**.

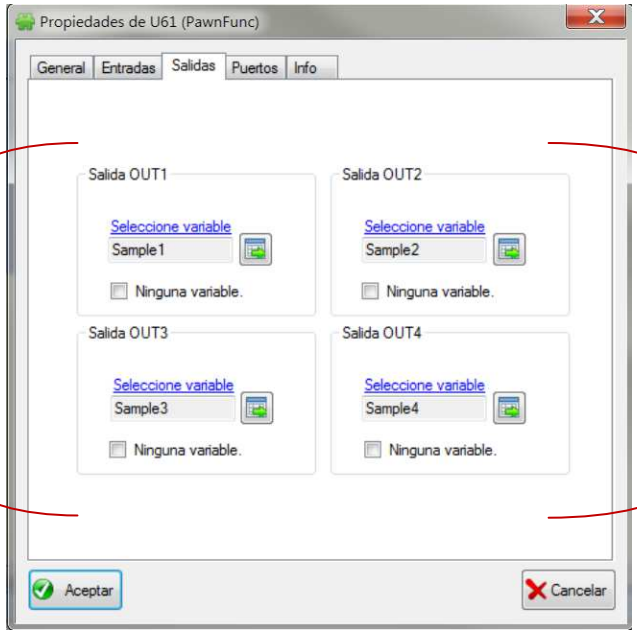
En la siguiente figura, se muestra la selección de la función Pawn en las propiedades del componente:



Note que al seleccionar la función, aparece la descripción de la misma y la cantidad y tipo de argumentos de entrada y salida que posee.

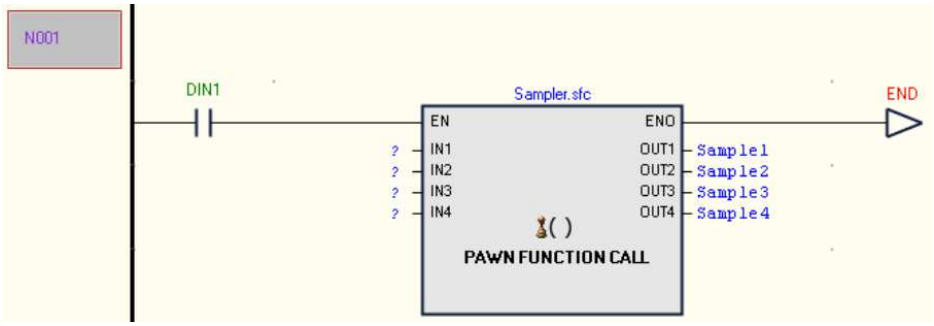
Diríjase a la pestaña "**Salidas**" y seleccione las variables **Sample1**, **Sample2**, **Sample3** y **Sample4** que previamente definimos en el proyecto:

Seleccione variables.



Seleccione variables.

Luego click en el botón "**Aceptar**" y debería ver el siguiente circuito:



Entonces, cuando la entrada **DIN1** sea 1, el componente **PawnFunc** llama a la función **Sampler.sfc** que a través de código Pawn muestrea las 4 entradas analógicas, depositando sus valores en los argumentos de salida **OUT1**, **OUT2**, **OUT3** y **OUT4**. Se conectan a los puertos **OUT1** a **OUT4** (argumentos de salida) del componente las variables **Sample1** a **Sample4**. Luego de ejecutarse el componente, las variables contendrán los valores de las entradas analógicas.

Puede llamar a la función Sampler.sfc tantas veces como sea necesario a través de del componente **PawnFunc**.

Si desea llamar a la función Sampler.sfc desde un script Pawn la sintaxis es la siguiente:

fp_ + Nombre + (Argumentos de entrada + Argumentos de Salida)

Donde:

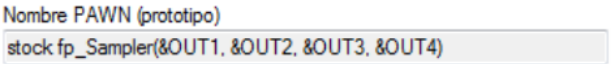
- **fp_** : Prefijo que significa "Funcion Pawn".
- **Nombre**: Nombre del archivo SFC de la función.
- **Argumentos de Entrada :** Valores o variables con los argumentos de entrada.
- **Argumentos de Salida:** Variables para retornar los valores de salida.

En el caso de la función **Sampler.sfc** creada, puede llamar desde Pawn a la función de la siguiente forma:

```
fp_Sampler(gi_Sample1, gi_Sample2, gi_Sample3, gi_Sample4)
```

Notar que 4 variables se pasan para almacenar los valores de los argumentos de salida devueltos por la función. Recuerde que el nombre de las variables en Pawn es distinto que en Ladder, para ello lea sección 5.5 en pag. 6.

Puede obtener un prototipo de la función Pawn de un archivo SFC, haciendo click en sus **Propiedades** y luego yendo a la pestaña **Info**:



La keyword "**stock**" le dice a Pawn que solo genere código por la función si es utilizada en el proyecto y los caracteres **"&"** indican que los argumentos de la función son variables que se pasan por referencia y se retorna un valor.

Finalmente, dentro de una función SFC, es posible utilizar la instrucción "**return Valor**", donde "**Valor**" es un numero del 0 al 1 (que también puede ser una variable) y altera el CurrentFlow de salida del componente **PawnFunc** cuando es llamado.

Ejemplo:

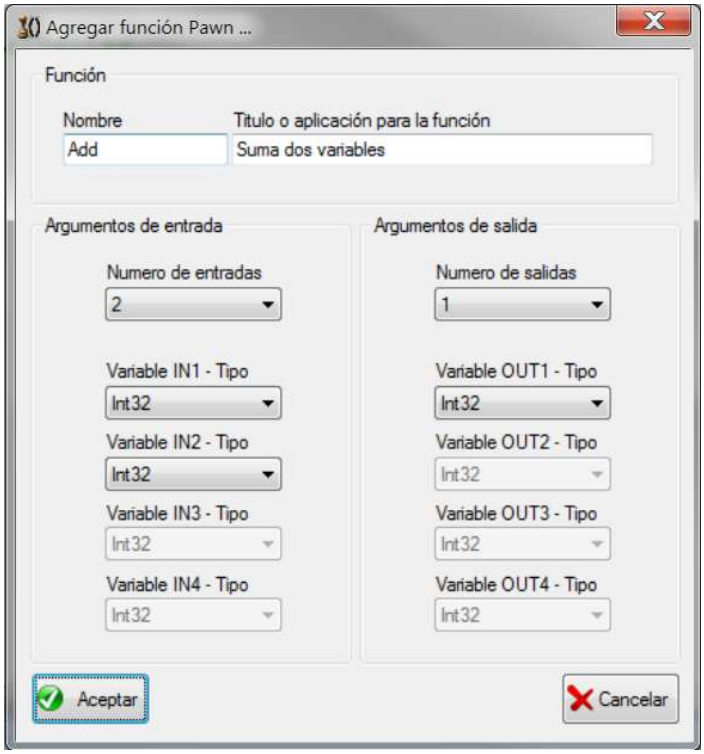
```
return 0
```

La función retorna el valor 0 y el **CurrentFlow** a la salida del componente es 0. Esto permite determinar el estado del flujo de corriente que le llega al próximo componente conectado al componente **PawnFunc** que llama a la función SFC,

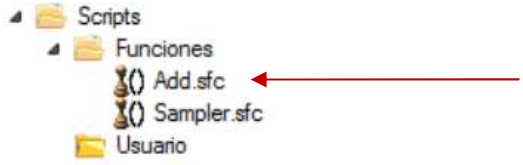
Entrada DIN2:

El próximo objetivo es determinar si DIN2=1, en ese caso, sumar dos variables enteras mediante una función SFC y luego mostrar en display LCD.

Para ello cree una función SFC con el nombre **Add** y dos argumentos de entrada tipo **Int32** y un argumento de salida tipo **Int32**:



El archivo **Add.sfc** debería crearse en el Explorador de Proyecto:

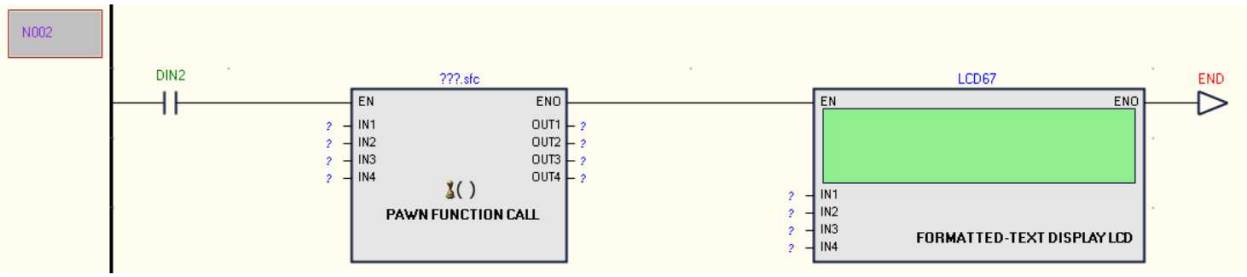


En el archivo **Add.sfc** agregue el siguiente código:

```
// Sumar variables.
OUT1 = IN1 + IN2
```

El código Pawn anterior, suma dos argumentos de entrada IN1 e IN2 y los deposita en el argumento de salida OUT1.

Luego en el diagrama **Principal.sld**, en la Network **N002**, dibuje el siguiente circuito:



Hay tres componentes, de izquierda a derecha: El primero es un contacto, el segundo llama a la función Pawn SFC con el componente **PawnFunc** y el tercero imprime texto en un display LCD del PLC con el componente **LcdTextPrintf**.

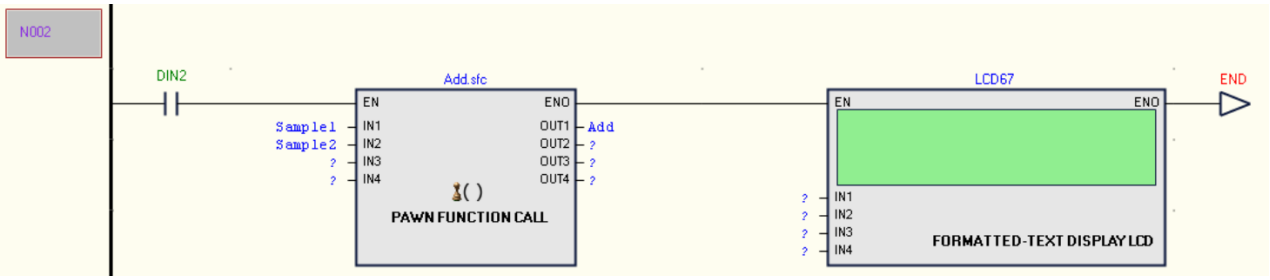
Si su PLC no tiene display LCD, puede utilizar el componente **NetHmiLcdPrintf** para reemplazar al componente **LcdTextPrinf**. En cuyo caso debe imprimir utilizando el software Virtual-HMI (www.slicetex.com/virtualhmi).

Entre a las propiedades del componente **PawnFunc**, seleccione la función Pawn "**Add**", luego desde las pestañas Entradas y Salidas, seleccione las siguientes variables:

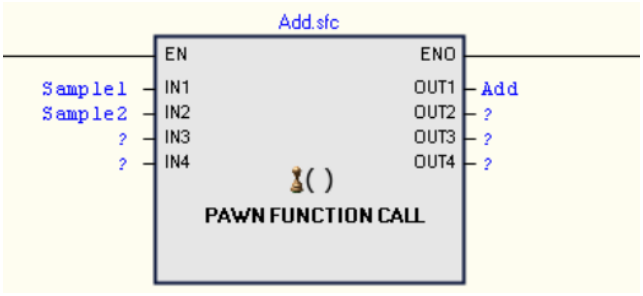
Pestaña Entradas: Seleccione variables **Sample1** y **Sample2**

Pestaña Salidas: Seleccione variable **Add**

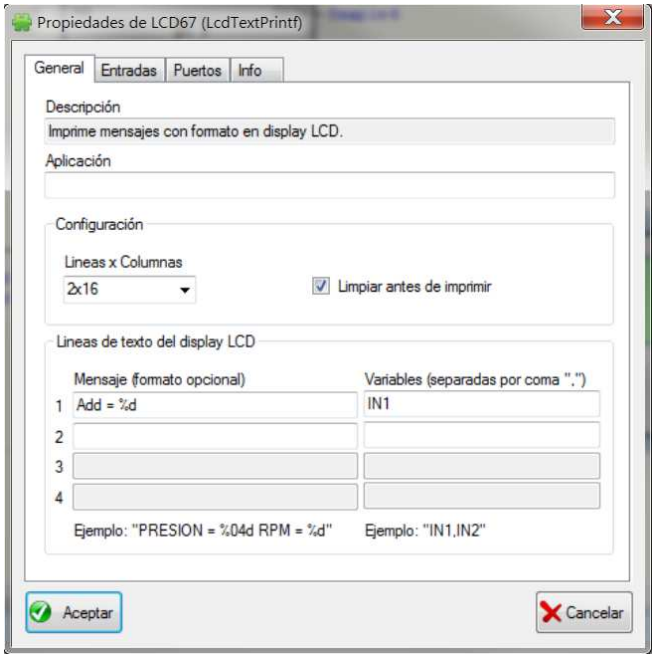
Luego de click en el botón **Aceptar**, resultando el siguiente circuito:



Entonces, cuando la entrada DIN2 es 1, se ejecuta la función Pawn **Add.sfc**, que suma las variables **Sample1** y **Sample2** conectadas a sus puertos IN1 e IN2. Luego deposita el resultado en la variable **Add** conectada al puerto OUT1.

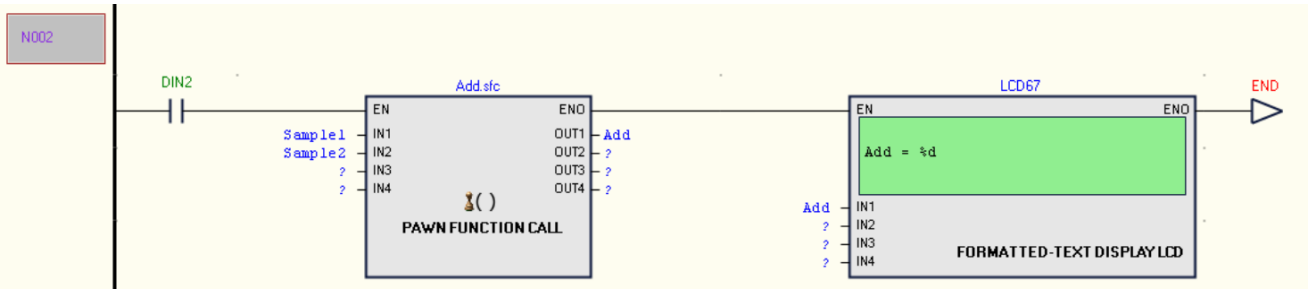


En el componente display **LcdTextPrinf**, es posible mostrar el valor de la variable "**Add**", especificando lo siguiente en sus propiedades:



Recuerde seleccionar en el puerto **IN1** del display la variable **Add** desde la pestaña Entradas.

El circuito resultante es:



Al ejecutarse la función **Add.sfc**, la suma se deposita en la variable Add, que luego se utiliza como valor de entrada para el puerto IN1 del display LCD, y así mostrarlo en la pantalla.

La función **Add.sfc**, se puede llamar desde el lenguaje Pawn de la siguiente manera:

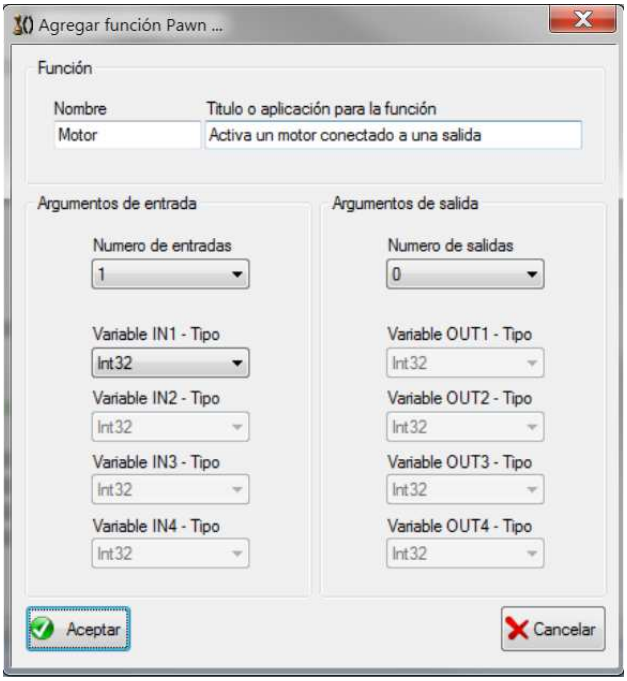
```
fp_Add(gi_Sample1, gi_Sample2, gi_Add)
```

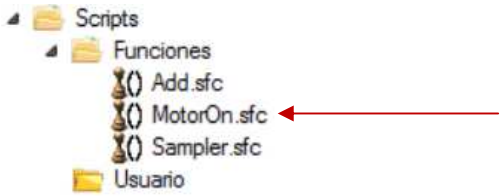
Entrada DIN3:

Para la entrada DIN3 tenemos el siguiente objetivo:

- Cuando DIN3=1, activaremos una salida del PLC que depende del valor pasado como argumento a la función SFC.

Para lograr el objetivo, primero debemos crear la función Pawn con un archivo SFC, llamado **MotorOn**, que acepte 1 argumento de entrada del tipo **Int32**:





Agregar en **MotorOn.sfc** el siguiente código Pawn:

```
// Procesar argumento de entrada.
switch(IN1)
{
	// Activar Motor 1.
	case 1:
		pBoolSet(gb_RELAY1)

	// Activar Motor 2.
	case 2:
		pBoolSet(gb_RELAY2)

	// Activar Motor 3.
	case 3:
		pBoolSet(gb_RELAY3)

	// Activar Motor 4.
	case 4:
		pBoolSet(gb_RELAY4)

	// Activar Alarma.
	default:
		pBoolSet(gb_RELAY5)
}
```

La instrucción:

```
switch(IN1)
```

Procesa el valor de IN1. Por cada posible valor hay un “case” o caso que se ejecutará. Si IN1=1, se ejecuta el case 1:

```
	// Activar Motor 1.
	case 1:
		pBoolSet(gb_RELAY1)
```

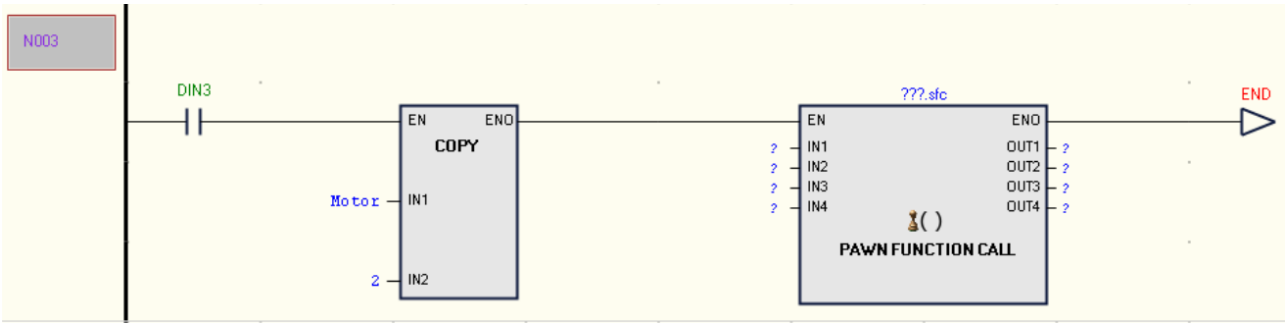
Pero si IN1 no tiene ningún valor valido, se ejecuta el “default” que activa una alarma conectada:

```
	// Activar Alarma.
	default:
		pBoolSet(gb_RELAY5)
```

Recordar que la función **pBoolSet()** sirve para acceder a variables localizadas en los periféricos del PLC, como los relés o salidas discretas, ver pag. 8 en sección 5.5.1.

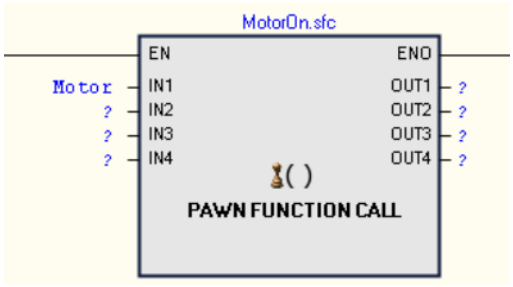
Nota: Solo se permite una instrucción por línea en “**case**” de un “**switch**”. Puede utilizar los “**if**”, “**else if**” para grandes cantidades de código.

Luego desde el diagrama **Principal.sld**, agregue el siguiente circuito en la network N003:

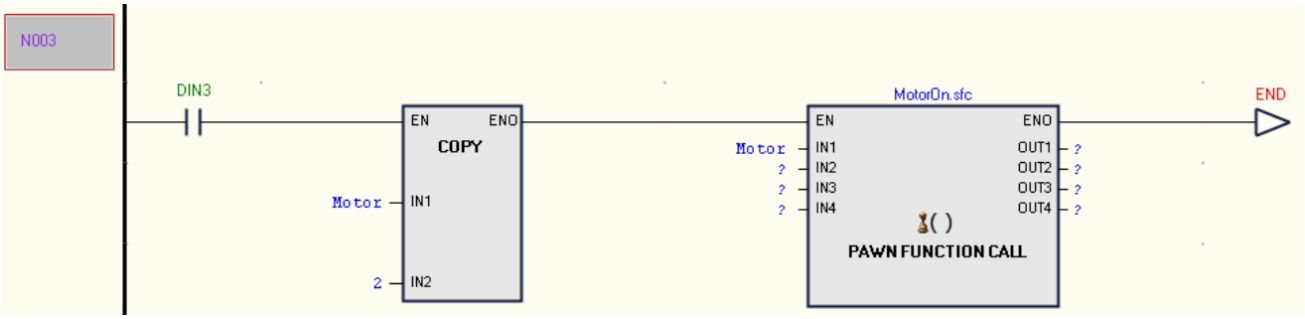


Cuando **DIN3** sea 1, se ejecuta el componente **Int32Copy**, que copia el valor "**2**" a la variable "**Motor**". Desde las propiedades del componente **Int32Copy** seleccione la variable **Motor** y especifique el valor 2 en el puerto **IN2** como constante.

Luego configure el componente **PawnFunc**, para que llame a la función Pawn **MotorOn.sfc** y desde la pestaña **Entrada** configure el puerto **IN1** para que lea la variable **Motor**:



Resultando el siguiente circuito:



Cuando el componente **PawnFunc** se ejecute, se pasa a la función **MotorOn.sfc** el valor de la variable **Motor**, como argumento de entrada **IN1**. En este caso Motor es igual a 2, por lo que según el código de la función Pawn implementada, el **RELAY2** del PLC debería activarse.

Ejercicio: Asigne diferentes valores a la variable **Motor** y observe el resultado.

6.2.2 Conclusiones

Utilizar funciones Pawn a través de archivos SFC y luego llamarlas desde Ladder a través del componente **PawnFunc**, es muy simple y cómodo. La ventaja que presenta es la posibilidad de reutilizar código en caso de que debamos llamarlo desde diferentes lugares del proyecto.

6.3 Agregar Código Pawn Mediante Archivos Script .p

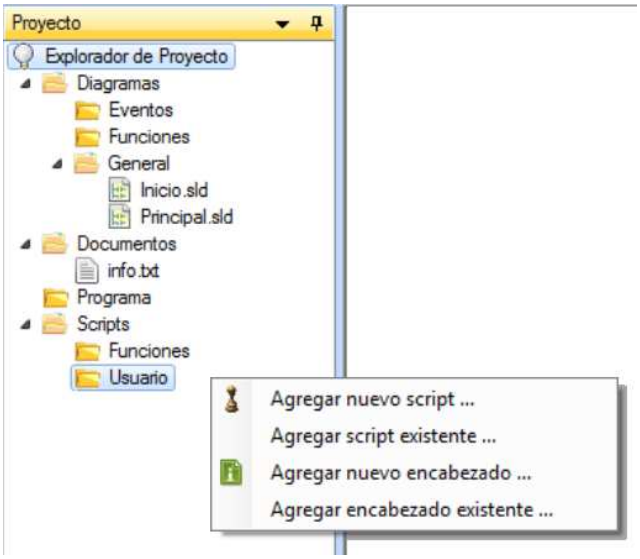
Este es el método clásico para agregar código Pawn. Para aquellos que programaban el PLC en Pawn únicamente, puede resultarles muy familiar.

Este método tiene la ventaja de que "**no hay límites**" para el código introducido, pero tiene como contra, que implica mayor conocimiento del lenguaje Pawn a la hora de programar.

Es posible agregar dos tipos de archivos con código Pawn al proyecto Ladder:

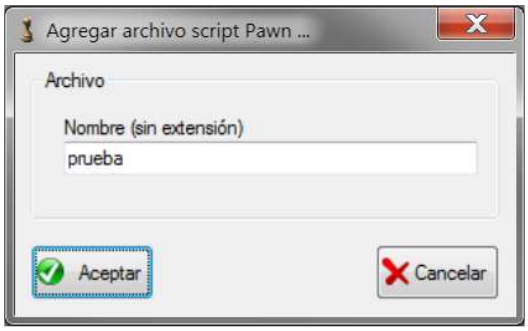
- Archivos con extensión **.p**: Contienen código Pawn como funciones, variables, eventos, etc.
- Archivos con extensión **.inc**: Encabezados, contienen declaraciones, constantes, macros, etc.

Los archivos se agregan haciendo click-derecho en la carpeta "**Scripts \ Usuario**" del "**Explorador de Proyecto**", como se muestra a continuación:



Para agregar un nuevo archivo "**.p**" seleccione en el menú contextual el ítem "**Agregar nuevo script ...**" y para agregar un nuevo archivo "**.inc**" seleccione "**Agregar nuevo encabezado …**".

Si elije crear un archivo **.p** para agregar código Pawn, se presenta la siguiente pantalla:



Ingrese el nombre del archivo y luego presione el botón "**Aceptar**".

6.3.1 Ejemplo Práctico

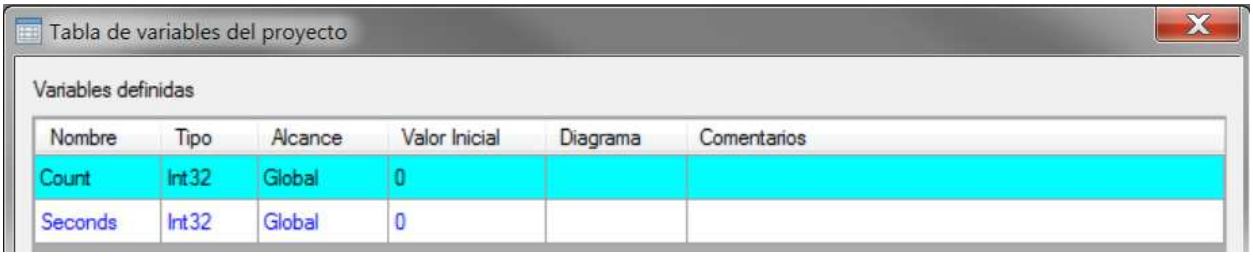
En el siguiente ejemplo mostraremos como utilizar funciones Pawn y Eventos desde archivos .p en el proyecto Ladder.

El objetivo del proyecto es el siguiente:

- Activar un evento **OnTimer** (temporizador) desde un archivo **.p** Pawn, que cada 1 segundo incrementará una variable entera definida en lenguaje Ladder. Luego desde el diagrama **Principal.sld** se comprueba el valor de dicha variable y si es mayor a 10, se conmuta el RELAY1.
- Se definirá una variable en Pawn que se incrementara cuando la entrada discreta DIN1 sea 1. Cuando la entrada discreta DIN2 sea 1, se mostrara dicha variable en un display LCD.

Importante: Este documento no es una guía de introducción para lenguaje Pawn, para ello recomendamos leer el documento **STXLADDER-PI** (Introducción al lenguaje Pawn) y el **Manual de Programación en Lenguaje Pawn** especifico para el modelo de PLC que está utilizando. Ambos se encuentran disponibles en nuestro sitio Web.

Para comenzar, primero comience un proyecto nuevo en Ladder y nómbrelo como "**PawnCodeFile**". Una vez creado el proyecto, defina las siguientes variables desde "**Proyecto > Tabla de variables**":

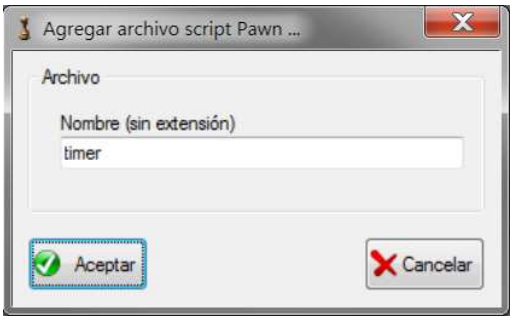


Donde **Count** y **Seconds** son variables globales del tipo **Int32**.

El ejemplo completo se encuentra disponible en nuestro sitio Web.

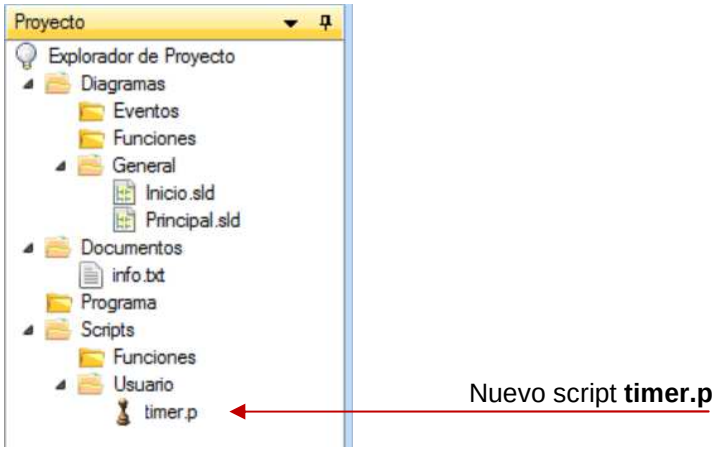
Activación del Evento OnTimer1:

Agregue un archivo .p al proyecto desde "**Scripts \ Usuario**" del "**Explorador de Proyecto**", seleccione "**Agregar nuevo script ...**" y nómbrelo "**timer**":

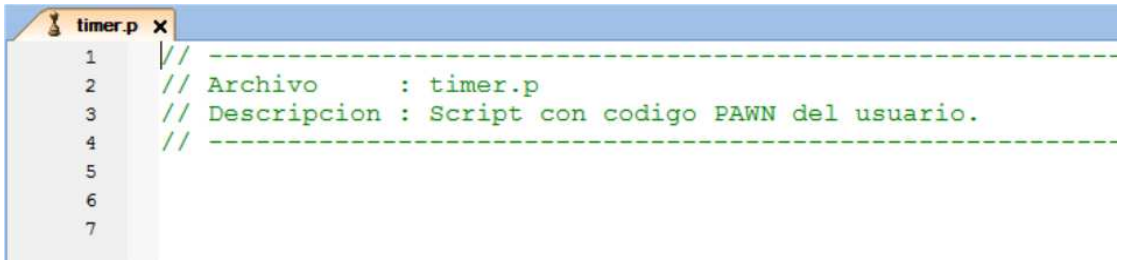


Luego de click en "**Aceptar**".

En el **Explorador de Proyecto** se debería agregar el archivo **timer.p** como se muestra:

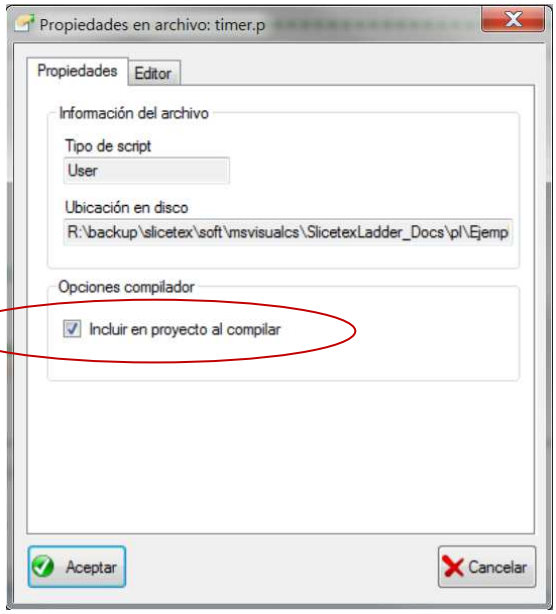


Si hace doble-click sobre el archivo **timer.p**, se abre el archivo y puede comenzar con la edición del mismo:



Es muy importante que incluya explícitamente a la lista de compilación del proyecto el archivo creado, para ello acceda a sus propiedades (click-derecho sobre el archivo) y tilde el siguiente check-box:

Importante: incluya el archivo en la lista de compilación, seleccionado "**Incluir en proyecto al compilar**".



Agregue el siguiente código Pawn en el archivo:

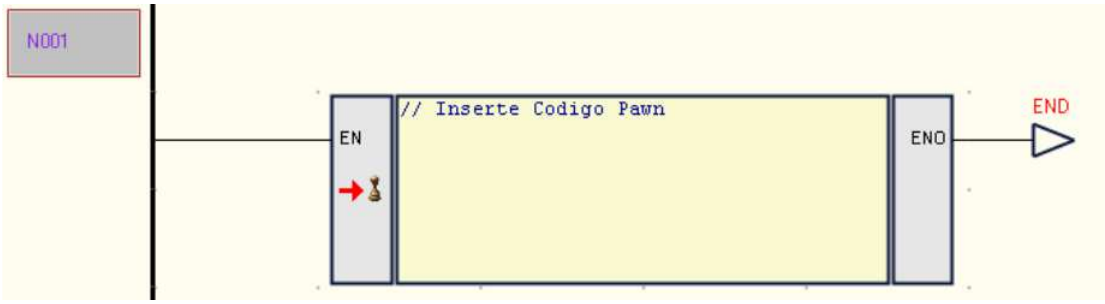
```
Timer1Config()
{
      // Resetear variable global a 0.
      gi_Seconds = 0

      // Configurar timer1 para generar un evento cada 1000 mS
      Timer1SetEvent(1000, true)
}

@OnTimer1()
{
      // Incrementar variable.
      gi_Seconds++
}
```

Se crea una función Pawn llamada **Timer1Config()**, que asigna el valor 0 a la variable **gi_Seconds** (definida en Ladder con el nombre **Seconds**) y luego mediante la función nativa **Timer1SetEvent()** se activa el evento **@OnTimer1()**.

Finalmente, se define el evento **@OnTimer1()** que es llamado cada 1 segundo por el Timer1. En cada llamada incrementa la variable **Seconds** (cuyo nombre en Pawn es **gi_Seconds**, ver pag. 6, sección 5.5).

El próximo paso, consiste en llamar desde el diagrama Inicio.sld en Ladder la función Pawn **Timer1Config()** creada en el archivo **timer.p**.

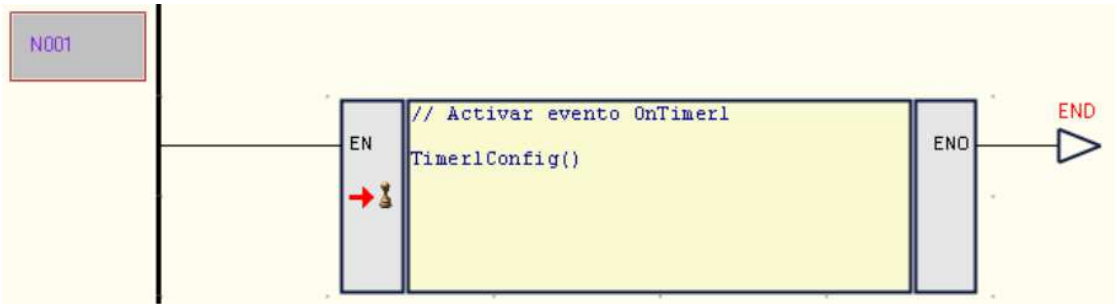
En la Network **N001**, del diagrama **Inicio.sld**, inserte el componente **PawnQuickInsert** como se muestra a continuación:



Desde las propiedades del componente, inserte el siguiente código:

```
// Activar evento OnTimer1
Timer1Config()
```

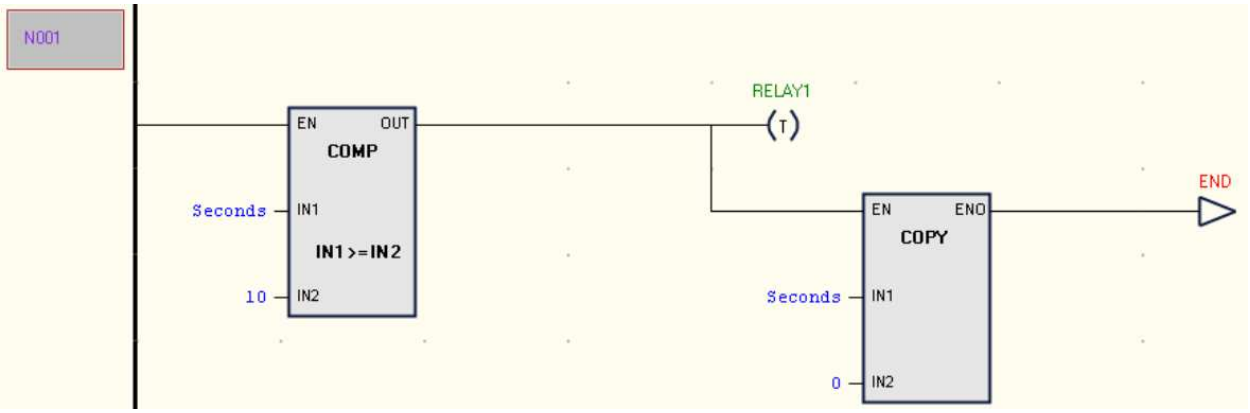
Resultando el siguiente circuito:



Cuando el PLC se inicialice, ejecuta en el diagrama **Inicio.sld** el componente **PawnQuickInsert**, que llama a la función **Timer1Config()** creada en el archivo Pawn **timer.p**. La función **Timer1Config()** activa el evento **@OnTimer1(),** que es llamado cada 1 segundo e incrementa la variable **Seconds** definida en Ladder.

El paso siguiente consiste en comprobar el valor de la variable **Seconds** desde el diagrama **Principal.sld**, si es mayor a 10, conmutamos o invertimos el estado del RELAY1 en el PLC.

En la Network N001 del diagrama **Principal.sld** dibuje el siguiente circuito:



El circuito mostrado, primero compara a la variable **Seconds** con el numero 10, si es mayor llama a la bobina toggle para invertir el estado del **RELAY1**. Al mismo tiempo asigna el valor 0 a la variable **Seconds** para resetearla.

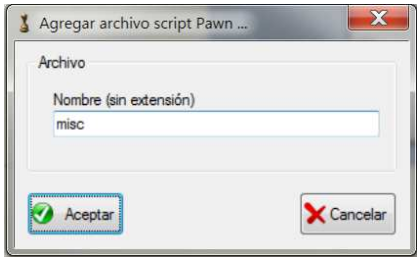
Como la variable **Seconds** es incrementada cada 1 segundo desde el evento Pawn **@OnTimer1()** en **timer.p**, el efecto neto de este circuito es que provoca cada 10 segundos la inversión del estado del **RELAY1**.

Este ejemplo, presupone que usted sabe cómo seleccionar variables desde los componentes mostrados (**Int32Comp** y **Int32Copy**), de lo contrario lea el Manual de Usuario de **StxLadder** (**STXLADDER-UM**).

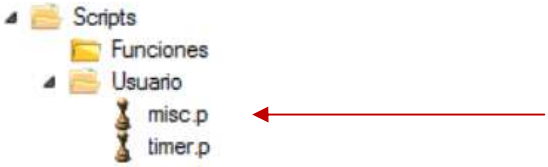
Variable creada en Pawn:

En el próximo paso, se definirá una variable en Pawn que se incrementara cuando la entrada discreta DIN1 del PLC sea 1. Cuando la entrada discreta DIN2 sea 1, se mostrara dicha variable en un display LCD.

Agregue un nuevo script **.p** en el Explorador de Proyecto y nómbrelo "**misc**":



Como resultado, el archivo **misc.p** debe agregarse al explorador:



Nota: Recuerde entrar a las propiedades y tildar la check-box "**Incluir en proyecto al compilar**".

Haga doble-click sobre el archivo para agregar el siguiente código Pawn:

```
new MyCount = 0

Increment()
{
      MyCount++
}

Copy(&Var)
{
      Var = MyCount
}
```

El código es muy simple y trivial, pero nos permite ilustrar los siguientes conceptos:

La siguiente línea crea una variable en Pawn de alcance global del tipo Int32:

```
new MyCount = 0
```

La función **Increment()** incrementa la variable **MyCount** cuando es llamada.

Y la función **Copy(),** permite copiar en "**Var**" la variable pasada como argumento el valor de la variable global "**MyCount**".

Note como en la definición de la función se utiliza el símbolo "**&**" para indicar que la variable que se pasa como argumento, se pasa por "referencia", por lo tanto cuando modificamos "**Var**", estamos modificando el valor de la variable que pasamos como argumento en realidad.

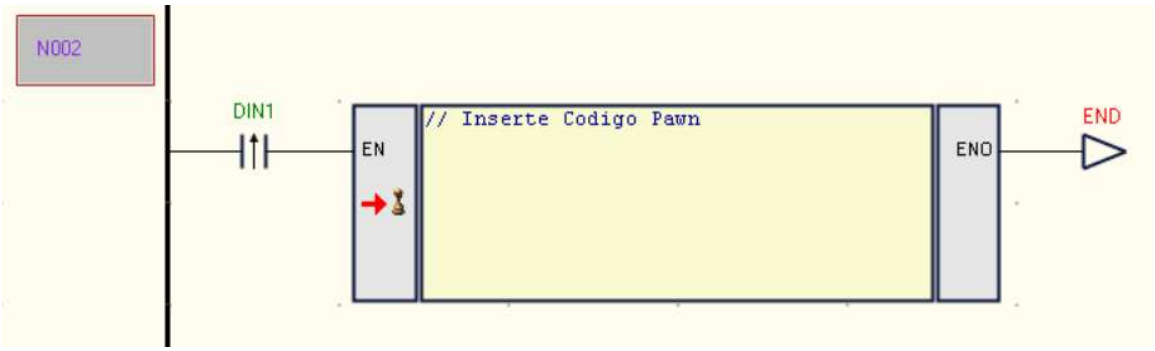
```
Copy(&Var)
```

La línea:

```
      Var = MyCount
```

Deposita en **Var** el valor de **MyCount**.

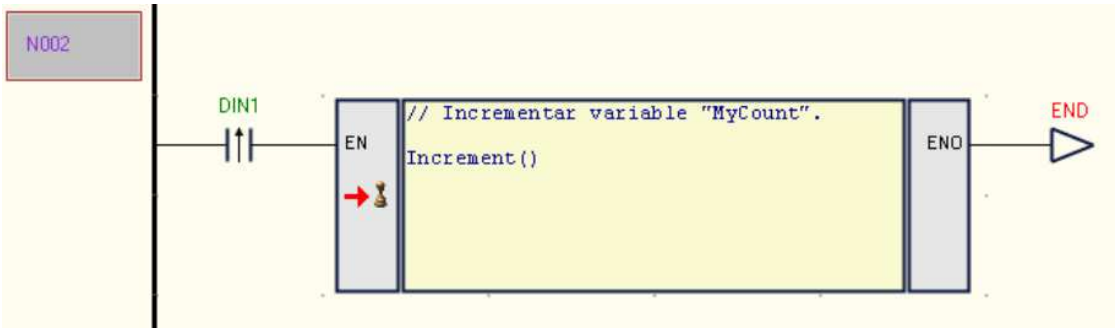
El próximo paso consiste en agregar en la Network N002 del diagrama **Principal.sld** el siguiente diagrama:



Inserte el siguiente código en el componente **PawnQuickInsert**:

```
Increment()
```

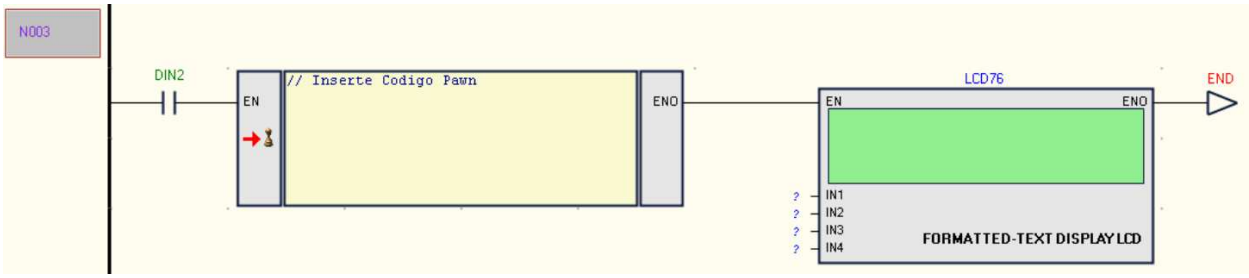
Resultando:



Entonces, el funcionamiento es el siguiente: Cuando DIN1 = 1, se llama al componente **PawnQuickInsert** que mediante código Pawn llama a la función **Increment()**, que incrementa la variable **MyCount** definida en el archivo **misc.p**.

Note que se utilizo un contacto por detección de flanco ascendente para leer la entrada DIN1, de esta forma solo incrementamos la variable **MyCount** cuando la entrada pasa de 0 a 1. Se lo contrario, mientras la entrada sea "1" estaríamos incrementando la variable indefinidamente, y ese no es el comportamiento que deseamos.

A continuación, vamos a agregar el siguiente circuito a la Network **N003** del diagrama **Principal.sld**:



La idea consiste en copiar la variable **MyCount** definida en Pawn a la variable **Count** definida en Ladder. Luego mostrar la variable **Count** en el display LCD. De esta forma, podemos leer indirectamente una variable definida en Pawn a través de componentes Ladder.

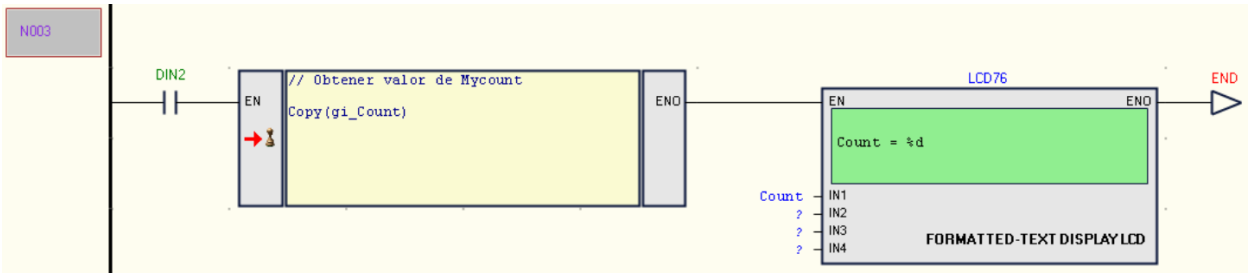
Agregue el siguiente código al componente **PawnQuickInsert**:

```
Copy(gi_Count)
```

Esto llama a la función **Copy()** en **misc.p**, que copia el valor de la variable **MyCount** en Pawn. En este caso le pasamos como argumento la variable **gi_Count** definida en Ladder. Cuando la función retorne, tendremos en **Count** el valor de **MyCount**.

Luego en el display LCD, agregue **Count** como variable de entrada al puerto **IN1**. Muestre con la leyenda "Count = %d" el valor del puerto **IN1**. Si su PLC no tiene display LCD, puede utilizar **VirtualHMI** (www.slicetex.com/virtualhmi).

El circuito resultante es el siguiente:



Cuando **DIN2 = 1**, se llama a la función **Copy()** definida en **misc.p** mediante el componente **PawnQuickInsert**, que copia **MyCount** en la variable **Count** de Ladder. Luego el display LCD muestra la variable **Count** en el display LCD del PLC.

6.3.2 Conclusiones

Agregar código Pawn mediante archivos Pawn, es una herramienta poderosa porque le da libertad absoluta para crear funciones, eventos, variables y algoritmos extensos en Pawn, pero manteniendo a su vez la integración con Ladder. Para hacerlo debe conocer el lenguaje Pawn y a su vez las funciones nativas del PLC, pero el esfuerzo luego se traduce en mayores posibilidades de creatividad.

7 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **Script**: Secuencia de sentencias y funciones que se escriben en un archivo.

8 Historial de Revisiones

Tabla: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
01 05/Jul/2013	1	1. Versión preliminar liberada.	Desarrollo

9 Información Legal

9.1 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.

10 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

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

Para ventas, envíe un mail a: ventas@slicetex.com

Para soporte, visite: www.slicetex.com/foro

Slicetex Electronics
Córdoba, Argentina

© Slicetex Electronics, todos los derechos reservados.

11 Contenido

1 DESCRIPCIÓN GENERAL 1

2 LECTURAS RECOMENDADAS 2

3 ALCANCE DEL DOCUMENTO 2

4 REQUERIMIENTOS 2

5 INTRODUCCIÓN 3

5.1 FINALIDAD DE LA MEZCLA DE LENGUAJES 3
5.2 COMO UTILIZAR PAWN EN LADDER 4
5.3 INTERFAZ ENTRE LADDER Y PAWN 4
5.4 APRENDIENDO DE STXLADDER 5
5.5 ACCESO A VARIABLES LADDER 6
5.5.1 ACCESO A VARIABLES TIPO BOOL DESDE PAWN 8
5.6 ACCESO A FUNCIONES LADDER 10

6 MÉTODOS PARA UTILIZAR PAWN EN LADDER 11

6.1 INSERTAR CÓDIGO PAWN EN UN DIAGRAMA LADDER 11
6.1.1 EJEMPLO PRÁCTICO 12
6.1.2 ACCESO INMEDIATO 21
6.1.3 CONCLUSIONES 21
6.1.4 CÓDIGO EXTRA 21
6.2 UTILIZAR FUNCIÓN PAWN SFC (SCRIPT FUNCIÓN CODE) 22
6.2.1 EJEMPLO PRÁCTICO 24
6.2.2 CONCLUSIONES 35
6.3 AGREGAR CÓDIGO PAWN MEDIANTE ARCHIVOS SCRIPT .P 36
6.3.1 EJEMPLO PRÁCTICO 37
6.3.2 CONCLUSIONES 43

7 ABREVIACIONES Y TÉRMINOS EMPLEADOS 44

8 HISTORIAL DE REVISIONES 44

9 INFORMACIÓN LEGAL 45

9.1 AVISO DE EXENCIÓN DE RESPONSABILIDAD 45

10 INFORMACIÓN DE CONTACTO 46

11 CONTENIDO 47

Copyright Slicetex Electronics 2013

www.slicetex.com