SIEMENS

SIMATIC

SCL para SIMATIC S7-300/400 Programación de bloques

Manual

Referencia del manual:
6ES7811-1CA02-8DA0-01

Consignas de seguridad para el usuario

Este manual contiene las informaciones necesarias para la seguridad personal así como para la prevención de daños materiales. Las informaciones están puestas de relieve mediante señales de precaución. Las señales que figuran a continuación representan distintos grados de peligro:

Peligro

Significa que, si no se adoptan las medidas preventivas adecuadas, **se producirá** la muerte, lesiones corporales graves o daños materiales considerables.

Precaución

Significa que, si no se adoptan las medidas preventivas adecuadas, **puede producirse** la muerte, lesiones corporales graves o daños materiales considerables.

Cuidado

Significa que, si no se adoptan las medidas preventivas adecuadas, pueden producirse lesiones corporales o daños materiales.

Nota

Se trata de una información importante, sobre el producto o sobre una parte determinada del manual, sobre la que se desea llamar particularmente la atención.

Personal cualificado

La puesta en funcionamiento y el servicio del equipo sólo deben ser llevados a cabo conforme con este manual.

Sólo está autorizado a intervenir en este equipo el **personal cualificado**. En el sentido del manual se trata de personas que disponen de los conocimientos técnicos necesarios para poner en funcionamiento, conectar a tierra y marcar los aparatos, sistemas y circuitos de acuerdo con las normas estándar de seguridad.

Uso conforme

Considere lo siguiente:

Precaución

El equipo o los componentes del sistema sólo se podrán utilizar para los casos de aplicación previstos en el catálogo y en la descripción técnica, y sólo en unión de los equipos y componentes de proveniencia tercera recomendados y homologados por Siemens.

Marca registrada

SIMATIC®, SIMATIC NET® y SIMATIC HMI® son marcas registradas por SIEMENS AG

Los restantes nombres y designaciones contenidos en el presente impreso pueden ser marcas registradas cuya utilización por terceros para fines propios pueden violar los derechos de los proprietarios.

Siemens AG
Berich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

Exención de responsabilidad

Hemos probado el contenido de esta publicación con la concordancia descrita para el hardware y el software. Sin embargo, es posible que se den algunas desviaciones que nos impiden tomar garantía completa de esta concordancia. El contenido de esta publicación está sometido a revisiones regularmente y en caso necesario se incluyen las correcciones en la siguiente edición. Agradecemos sugerencias.

Prólogo

Finalidad del manual

Este manual le ayuda en la creación de programas de usuario mediante el lenguaje de programación SCL. Se explican los procedimientos principales para la creación de programas con el editor SCL, el compilador SCL y el depurador SCL.

El manual contiene además una sección de referencia sobre los elementos del lenguaje de programación SCL. En ella se describen la sintaxis y el modo de funcionamiento de cada uno de los elementos.

Destinatarios

Este manual está dirigido a programadores de programas S7, operadores y personal de mantenimiento que dispongan de conocimientos básicos sobre los autómatas programables, así como del software básico STEP 7.

Ambito de validez del manual

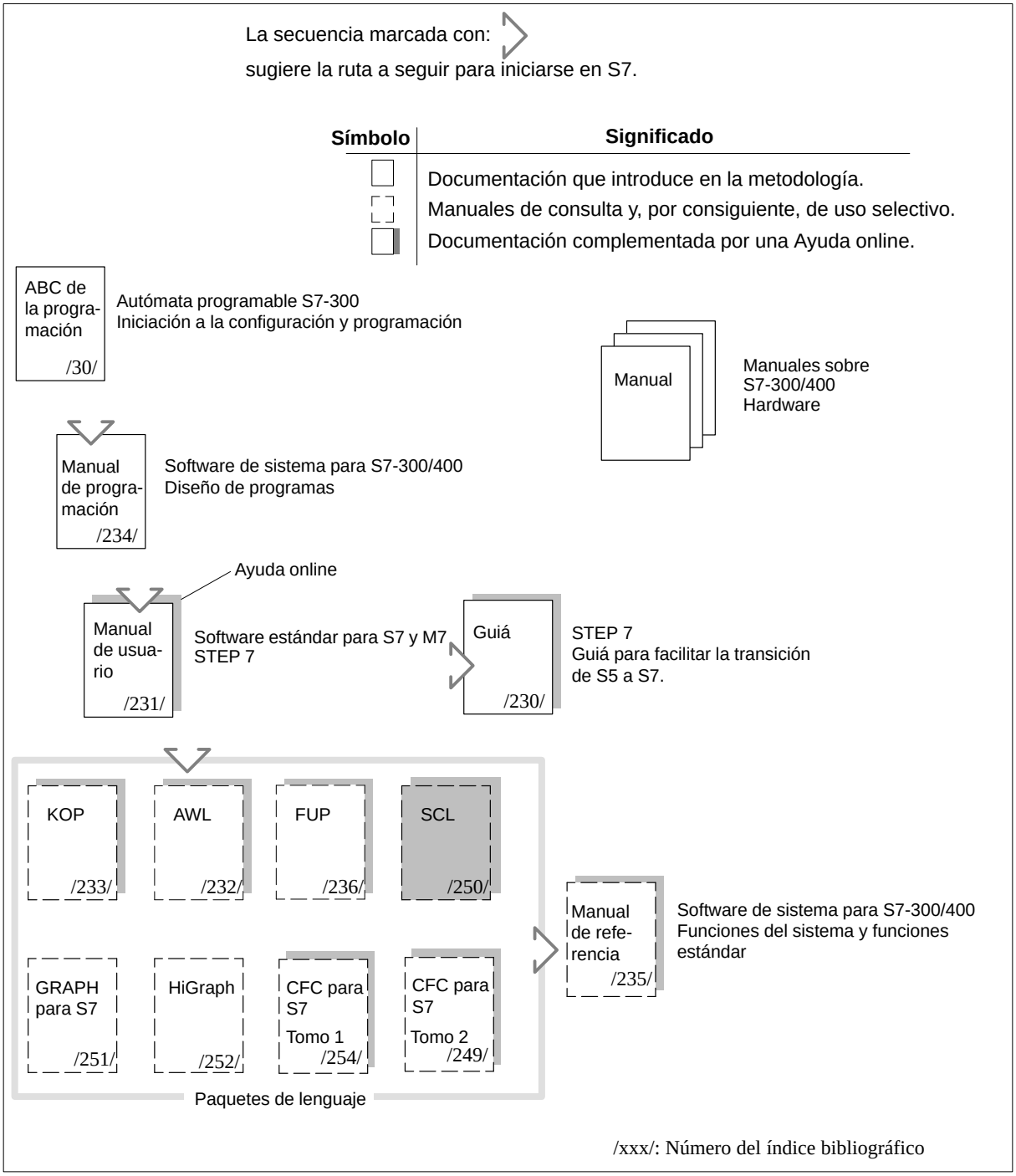
El presente manual es válido para la versión 3.0 del software de programación STEP 7. Se aplica para el paquete de software básico STEP 7.

Cumplimiento de la norma IEC 1131-3

SCL es acorde con el lenguaje ”Structured Control Language” definido en la norma DIN EN-61131-3 (int. IEC 1131-3), si bien existen considerables diferencias en las operaciones. Encontrará información más precisa sobre el cumplimiento de las normas en la tabla del archivo NORM.TAB de STEP 7.

Integración en el conjunto de la documentación S7 300/400

Para fines de configuración y programación de un autómata programable S7 se dispone de una amplia documentación de usuario, prevista para su uso selectivo. Las explicaciones siguientes y la figura siguiente ilustran la aplicación de dicha documentación de usuario.



Título	Contenido
ABC de la programación S7-300 Iniciación a la configuración y programación	El ABC ofrece una introducción bastante simple en la metodología de la configuración y programación de un S7-300/400. Se presta especialmente para aquellos que utilizan por primera vez un autómata programable S7.
Manual de programación Diseño de programas S7-300/400	El manual de programación ”Diseño de programas S7-300/400” ofrece los conocimientos básicos sobre la estructura del sistema operativo y del programa de usuario de una CPU S7. Sirve a los nuevos usuarios de un S7-300/400 para obtener una visión general en la metodología de programación para, basándose en ella, diseñar su programa de usuario.
Manual de referencia Funciones del sistema y funciones estándar S7-300/400	Las CPU S7 contienen funciones del sistema y bloques de organización integrados en el sistema operativo que se pueden utilizar en la programación. El manual ofrece una visión general sobre las funciones del sistema disponibles en el S7, los bloques de organización y las funciones estándar cargables, así como – en calidad de información de consulta – descripciones detalladas de interfaces para su aplicación en el programa de usuario.
Manual de usuario STEP 7	El manual de usuario ”STEP 7” explica la aplicación básica y las funciones del software de automatización STEP 7. Tanto al nuevo usuario del STEP 7 como al experto de STEP 5, el manual ofrece una visión general de cómo proceder para efectuar la configuración, programación y puesta en servicio de un S7-300/400. Al trabajar con el software se puede acceder puntualmente a la ayuda online, la cual asiste en cualquier cuestión de detalle relativa a la aplicación del software.
Guiá Para facilitar la transición de S5 a S7.	El guiá ”Para facilitar la transición de S5 a S7” se requiere para convertir los programas S5 existentes y cargarlos luego en las CPU S7. Ofrece una visión general sobre el funcionamento y la aplicación del convertidor; el manejo detallado de las funciones del convertidor se describe en la ayuda online. Esta incluye también las informaciones de interface de las funciones S7 convertidas disponibles.
Manuales AWL, KOP, FUP, SCL[1]	Los manuales de los paquetes de lenguaje AWL, KOP, FUP y SCL contienen tanto las instrucciones de usuario como también la descripción del lenguaje tratado. Para la programación de un S7-300/400 se requiere un solo lenguaje pero, en caso necesario, se pueden mezclar varios lenguajes dentro de un proyecto. Para la primera aplicación de los lenguajes, es recomendable familiarizarse con el manual que trata sobre la metodología de programación. En las operaciones con el software se puede utilizar la Ayuda online, la cual ofrece detalles sobre la aplicación de los editores/compiladores correspondientes.
Manuales GRAPH[1] , HiGraph[1], CFC[1]	Los lenguajes GRAPH, HiGraph, CFC ofrecen posibilidades adicionales para implementar controles secuenciales, de estado o interconexión gráfica de bloques. Los manuales contienen tanto las instrucciones de usuario como la descripción del lenguaje tratado. Al usar por primera vez un lenguaje se recomienda familiarizarse con el manual sobre la metodología de programación. Al trabajar con el software se puede utilizar la ayuda online (excepción HiGraph), la cual ofrece detalles sobre la aplicación de los editores/compiladores.

1 Paquetes de opciones para el software del sistema S7-300/400

Ayudas para acceder al manual

Este manual presupone conocimientos teóricos de programación con S7 que puede repasar en el manual de programación **/234/**. SCL se situa encima del software básico STEP 7. El usuario ya debe tener conocimientos en relación con el software básico, descrito en el manual de usuario **/231/**.

Este manual se compone de las áreas temáticas siguientes:

- El capítulo 1 ofrece una panorámica de las posibilidades de programación de un control secuencial con SCL.
- En el capítulo 2 se tratará, ayudándose de un ejemplo, un proceso de diseño ayudándose que podrá ejecutar usted mismo.
- Los capítulos 3 a 6 le muestran el manejo del entorno de desarrollo de SCL. Conocerá el editor, el compilador y el depurador de SCL.
- Los capítulos 7 a 19 componen la parte de referencia, que pretende aportarle datos precisos sobre el funcionamiento de cada una de las instrucciones SCL.

En los anexos encontrará:

- La exposición completa de la sintaxis de SCL.
- Un glosario en el que se explican conceptos importantes.
- Un índice de referencias cruzadas que le ayudará a encontrar con rapidez los pasajes referidos a palabras clave importantes.

Convenciones

Las referencias de documentación adicional se indican a través de índices de bibliografía escritos entre barras /.../. Con estos números se puede localizar el título exacto de la documentación correspondiente en el índice bibliográfico.

Soporte complementario

Si necesita informaciones relativas a la utilización del software descrito y no encontradas en la documentación en papel ni en la ayuda online, contactar con el interlocutor especializado de la sucursal, delegación o agencia de Siemens más próxima. Las direcciones figuran en el anexo de **/70/** a **/100/**, así como en los catálogos y en Compuserve (`go autforum`). Asimismo puede contactar nuestro servicio de atención al cliente:

Tel.: +49(911) 895-7000 (fax 7001)

En caso de comentarios o sugerencias que afecten al propio manual, sírvase rellenar la hoja de respuesta que para dicho fin figura al final del manual y envíarla a la dirección indicada. También desearíamos recibir en dicha hoja la opinión que le merece el presente manual.

Para facilitar la introducción en el sistema de automatización SIMATIC S7 ofrecemos una gama completa de cursillos de formación. Para más información sobre el tema, contactar con su centro de formación regional o dirígase al centro de formación central en:

D–90327 Nürnberg, Tel. 911 / 895 3154.

Observación

La parte de este manual que está dedicada al usuario no incluye instrucciones detalladas sobre los pasos específicos a dar en cada situación concreta, sino que pretende explicar de modo general los procedimientos a seguir. En la Ayuda online encontrará información detallada sobre cada cuadro de diálogo del software y sobre el modo en que deberá proceder en cada caso.

Indice

Parte 1:
Diseño de programas

Panorámica del producto

1

Lenguaje de programación SCL

Además de las clásicas funciones de control los sistemas de automatización deben asumir cada vez con más frecuencia tareas de gestión de datos y funciones matemáticas más complejas. Enfocado especialmente para la solución de dichas tareas le ofrecemos SCL para SIMATIC S7-300/400 (Structured Control Language), el lenguaje de programación normalizado según IEC 113-3 que le facilita la labor de programación.

SCL le sirve de apoyo en las funciones de control ”normales” y en numerosas aplicaciones, siendo superior a los lenguajes de programación ”clásicos” en los siguientes campos de aplicación:

- gestión de datos,
- optimización de procesos,
- gestión de recepciones,
- funciones matemáticas/estadísticas.

Características técnicas

Para poder trabajar con SCL necesita una unidad de programación SIMATIC o un PC (a partir de procesador 80486, con 16 MBytes de memoria principal).

Elementos de lenguaje	
Operadores	potencia/aritmética/ comparaciones/combinaciones lógicas
Funciones	temporizadores/contadores/ llamadas a bloques de función
Estructuras de control	bucles (FOR/WHILE/REPEAT) alternativas (IF THEN/CASE/GOTO)
Tipos de datos	
simples	BOOL/BYTE/WORD/DWORD/INT/DINT/REAL DATE/TIME/TIME_OF_DAY/S5TIME
compuestos	cadenas/arrays/estructuras/estructuras de usuario/DATE_AND_TIME

Indice del capítulo

1.1 ¿Qué es SCL?

Lenguaje de alto nivel

SCL (*Structure Control Language)* es un lenguaje de programación de alto nivel orientado a PASCAL. El lenguaje se basa en una norma para PLC (autómatas programables).

La norma DIN EN-61131-3 (int. IEC 1131-3) normaliza los lenguajes de programación para autómatas programables. La base de SCL es la parte *"texto estructurado"*. En la "Compliance List" del archivo NORM.TAB de STEP 7 encontrará información más detallada sobre las normas que cumple este lenguaje.

Además de elementos de lenguaje de alto nivel, SCL incluye también elementos típicos del PLC, como entradas, salidas, temporizadores, marcas, llamadas a bloques, etc., que son elementos del propio lenguaje. Es decir, SCL completa y amplía el software de programación STEP 7 con sus lenguajes de programación KOP y AWL.

Entorno de desarrollo

Para utilizar de forma óptima y aplicar en la práctica SCL existe un entorno de desarrollo de gran capacidad adaptado tanto a las características específicas de SCL como a las de STEP 7. Este entorno de desarrollo está formado por los siguientes componentes:

- Un **editor**, para elaborar programas compuestos de funciones (FC), bloques de función (FB), bloques de organización (OB), bloques de datos (DB) y tipos de datos de usuario (UDT). El programador cuenta con la ayuda de potentes funciones.
- Un **compilador de lotes,** para compilar a código máquina MC7 el programa previamente editado. El código MC7 generado puede ejecutarse en todas las CPU del sistema de automatización S7-300/400 a partir de la CPU 314.
- Un **depurador,** que permite buscar errores lógicos de programación en una compilación sin errores. La búsqueda de errores se realiza en lenguaje fuente.

Cada uno de los componentes se maneja con sencillez y comodidad, puesto que corren en Windows 95 y aprovechan todas las ventajas de este sistema operativo.

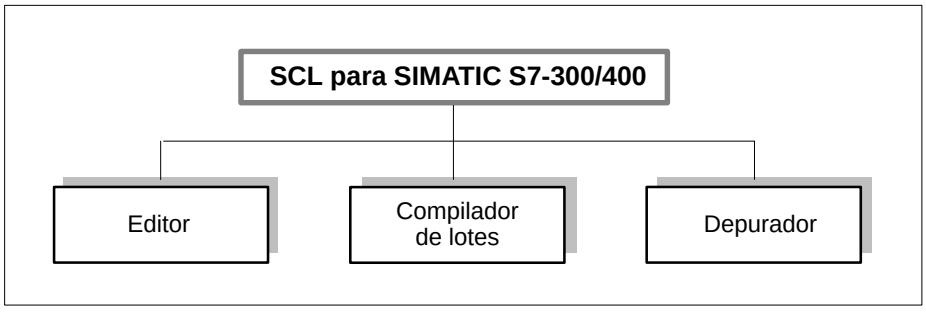


Figura 1-1 *Entorno de desarrollo de SCL*

1.2 ¿Qué ventajas le ofrece SCL?

Lenguaje de programación de muy alto nivel

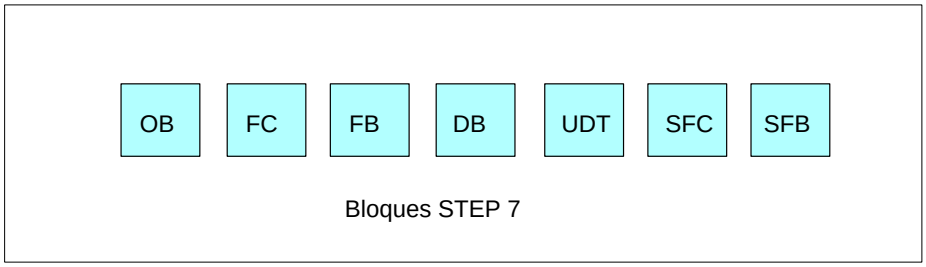
SCL le ofrece todas las ventajas de un lenguaje de programación de muy alto nivel. Pero SCL incluye también algunas características que fueron diseñadas específicamente para servir de ayuda a la programación estructurada; p.ej.:

- la estructura de bloques de STEP 7,
- bloques predefinidos,
- compatibilidad con STEP 5.

Estructura de bloques de STEP 7: calidad acreditada

SCL está diseñado de forma óptima para solucionar todas las tareas que se presentan en proyectos de automatización, de forma que, junto con STEP 7, le permite trabajar con eficacia en todas las fases del proyecto.

SCL apoya en particular el concepto de bloques de STEP 7 y permite, junto con AWL y KOP, programar bloques conforme a las normas.



Tipos de bloques

Los bloques STEP 7 son partes de un programa de usuario delimitadas por su función, su estructura o el fin para el que se utilizan. Con SCL puede crear los siguientes bloques:

Abrev.	Tipo de bloque	Función
OB	Bloque de organización	Interface entre sistema operativo y programa de usuario.
FC	Función	Módulo sin memoria con posibilidad de transferencia de parámetros.
FB	Bloque de función	Módulo con posibilidad de transferencia de parámetros y con memoria.
DB	Bloque de datos	Bloque para depositar datos de usuario.
UDT	Tipo de datos de usuario	Bloque para depositar un tipo de datos de usuario.

Bloques predefinidos

Usted no tiene necesidad de programar todas las funciones; también puede acudir a bloques predefinidos. Estos son bloques que existen en el sistema operativo de las unidades centrales de procesamiento o en librerías *(S7lib)* del paquete básico STEP 7, y que pueden utilizarse, p.ej., para la programación de funciones de comunicaciones. En concreto se trata de los siguientes tipos de bloques:

Abrev.	Tipo de bloque	Función
SFC	Función de sistema	Características iguales que las de una función (FC)
SFB	Bloque de función del sistema	Características iguales que las de un bloque de función (FB)

Posibilidad de mezclar los bloques

Los bloques programados con SCL puede mezclarlos con bloques programados en AWL, KOP y FUP. Esto significa que un bloque programado con SCL puede llamar a otro bloque programado en AWL, en KOP o en FUP. Consecuentemente los bloques SCL también pueden ser llamados en programas AWL, KOP y FUP. En definitiva, los lenguajes de programación de STEP 7 y SCL (paquete opcional) se complementan de forma óptima.

Capacidad de recompilación

En aplicaciones prácticas concretas, los bloques SCL pueden recompilarse al lenguaje de programación de STEP 7 AWL (lista de instrucciones). No es posible la recompilación a SCL.

Compatibilidad con STEP 5

A excepción de algunos casos concretos, los bloques que haya programado con SCL en STEP 5 son compatibles con las versiones superiores; es decir, estos bloques también pueden editarse, compilarse y comprobarse con SCL para STEP 7.

Métodos de programación

SCL sirve de ayuda a la programación estructurada mediante métodos de ingeniería de software.

Facilidad de aprendizaje

SCL se aprende fácilmente con un poco de experiencia en un lenguaje de programación de alto nivel, puesto que el repertorio de estructuras del lenguaje SCL se orienta a lenguajes de programación de alto nivel.

1.3 Características del entorno de desarrollo

Editor

El editor SCL es un editor de texto con el que se puede editar cualquier tipo de texto. La tarea central que puede realizar con él es la de generar y editar archivos fuente para programas de STEP 7. En un archivo fuente puede programar uno o varios bloques:

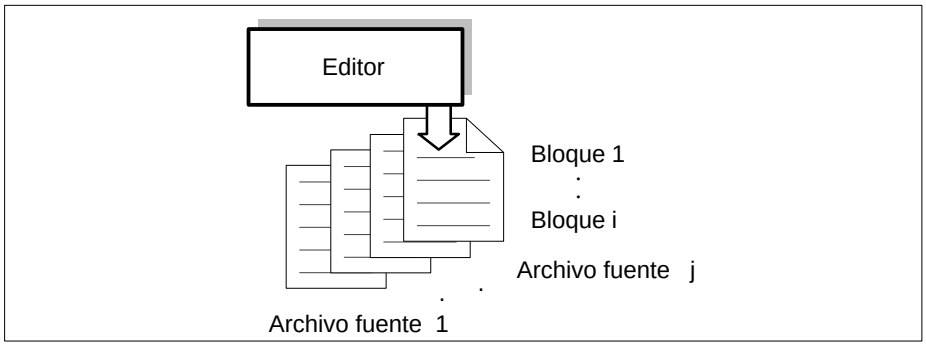


Figura 1-2 *Editor SCL*

El editor SCL permite:

- editar un archivo fuente completo con uno o varios bloques;
- editar un archivo de control de compilación, con el que podrá automatizar la compilación de varios archivos fuente;
- utilizar funciones complementarias que le permitirán una edición cómoda del texto fuente, p.ej., buscar y reemplazar;
- ajustar de forma óptima el editor a sus necesidades.

Durante la entrada de datos se realiza una verificación de la sintaxis.

Compilador

Después de que ha creado sus archivos fuente con el editor SCL, debe compilarlos a código MC7.

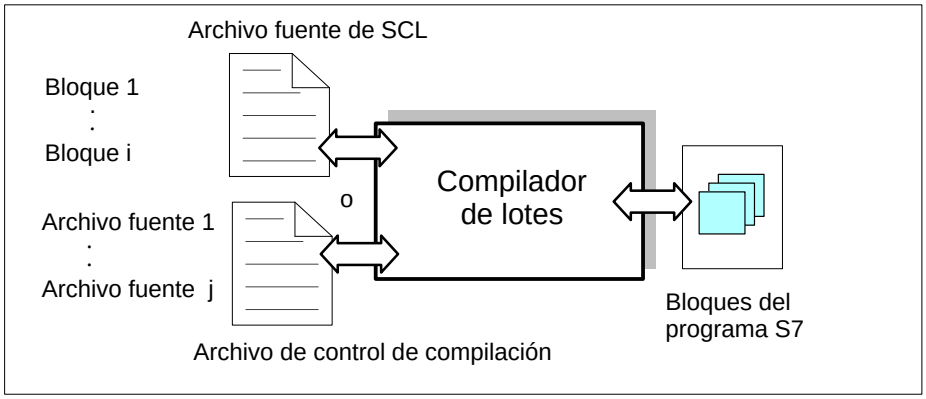


Figura 1-3 *Compilador SCL*

El compilador SCL permite:

- compilar un archivo fuente de SCL con varios bloques en un solo proceso de compilación;
- compilar varios archivos fuente SCL por medio de un archivo de control de compilación que contiene los nombres de los archivos fuente;
- ajustar de forma óptima el compilador a sus necesidades;
- visualizar todos los errores y alarmas que se producen durante la compilación;
- localizar con facilidad el punto erróneo del texto fuente, y opcionalmente describir el error e indicar datos para remediarlo;

Depurador

El depurador SCL ofrece la posibilidad de controlar un programa cuando se está ejecutando en el PLC, y por lo tanto, encontrar posibles errores lógicos.

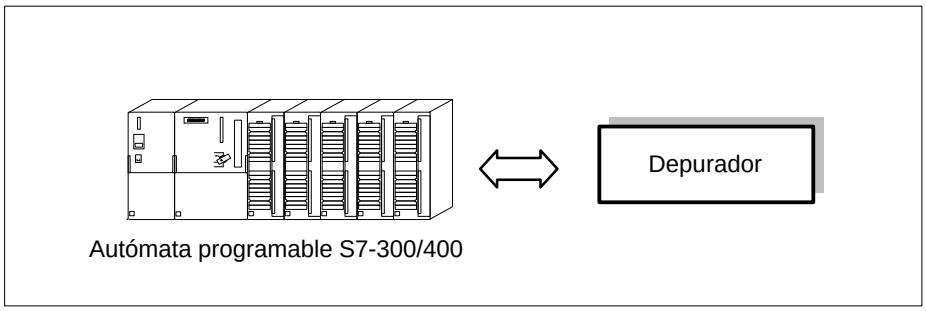


Figura 1-4 *Depurador SCL*

SCL le ofrece dos modos de test diferentes:

- observar **progresivamente**.
  En este modo se completa la ejecución lógica del programa. Usted puede ejecutar el algoritmo del programa instrucción por instrucción y visualizar en una ventana de resultados las variaciones producidas en las variables editadas;
- observar **continuamente**.
  Le permite efectuar un test de un grupo de instrucciones dentro de un bloque del archivo fuente. Mientras se realiza el test se visualizan, ordenados cronológicamente, los valores de las variables y parámetros, y se actualizan cíclicamente siempre que sean posible.

Paquete básico STEP 7

El entorno de desarrollo SCL ofrece la posibilidad de utilizar directamente desde SCL funciones del paquete básico STEP 7, como visualizar y modificar el estado operativo de la CPU, y ajustar la hora.

Diseñar un programa SCL

2

¿Qué describe este capítulo?

La práctica demuestra que la forma más sencilla y rápida de programar es estructurar la tarea; es decir, descomponerla en partes individuales independientes entre sí. SCL le ayuda a hacerlo. Con SCL podrá diseñar de forma racional y eficaz cada uno de los bloques individuales.

Este capítulo describe cómo puede diseñar y aplicar un programa de usuario en SCL. La descripción se ilustrará con un programa de ejemplo que usted podrá ejecutar con los datos de test que se proporcionan y con sus propios módulos de entrada y salida.

Indice del capítulo

2.1 Resumen

Objetivo

La primera parte de este manual que observa el diseño de programas pretende responder a las cuestiones que suelen plantearse habitualmente al principio, como las que indicamos a continuación:

- ¿Cómo puedo proceder para crear programas en SCL?
- ¿Qué medios ofrece el lenguaje SCL para solucionar la tarea planteada?
- ¿De qué funciones de test dispongo?

Medios del lenguaje SCL

En el ejemplo se presentarán, entre otros, los siguientes elementos del lenguaje SCL:

- Estructura y utilización de los diferentes tipos de bloque en SCL
- Llamada a los bloques con transferencia y valoración de parámetros
- Diferentes formatos de entrada y salida
- Programación con tipos de datos simples y arrays
- Inicialización de las variables
- Estructura del programa con utilización de ramificaciones y bucles

Requisitos de hardware para el ejemplo

Puede ejecutar el programa de ejemplo con SIMATIC S7-300 o con SIMATIC S7-400, para lo cual necesita los siguientes periféricos:

- un módulo de entrada con 16 canales
- un módulo de salida con 16 canales

Funciones de test

El programa está estructurado para permitirle realizar un test rápido mediante los interruptores de la entrada y las indicaciones de la salida. Para realizar un test detallado utilice las funciones de test de SCL (v. cap. 6).

Además de las funciones de SCL, dispone de toda la funcionalidad del paquete básico STEP 7.

2.2 Tarea planteada

Resumen

Los valores medidos deben registrarse, clasificarse y procesarse a través del módulo de entrada. El rango exigido para los valores medidos está comprendido entre 0 y 255. Por lo tanto se necesita un byte para el registro de entrada.

Las funciones de procesamiento utilizadas son la raíz y el cuadrado. Los resultados deben visualizarse en el módulo de salida, para lo cual se necesita una palabra. El programa se controla con un byte de entrada.

Registro de valores medidos

Un valor medido que se ajusta mediante 8 interruptores de entrada debe recibirse en el área de valores medidos de la memoria exactamente cuando se reconoce un flanco en el interruptor de entrada (v. fig. 2-1). El área de valores medidos debe organizarse como búfer de circulación con un máximo de 8 registros de entrada.

Procesamiento de valores medidos

Cuando en el interruptor de clasificación se reconoce un flanco, los valores guardados en el campo del área de medida deben clasificarse en orden ascendente. Después se calculará la raíz y el cuadrado para cada uno de los valores.

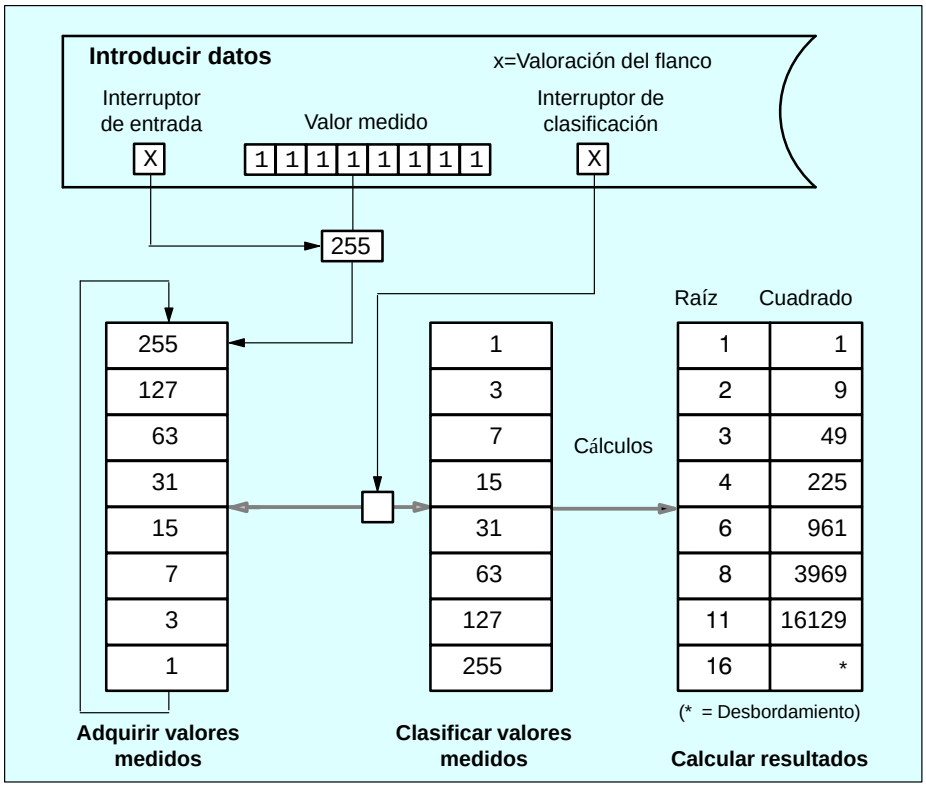


Figura 2-1 *Registro y procesamiento de valores medidos*

Salida ajustable

Dado que en la salida sólo puede visualizarse un valor cada vez, deben existir las siguientes posibilidades de selección:

- Selección de un elemento dentro de una lista
- Selección entre valor medido, raíz y cuadrado

La selección de un elemento dentro de una lista debe efectuarse direccionando un elemento de la lista mediante el siguiente ajuste con interruptores:

- con tres interruptores se ajusta un código que se acepta cuando en el cuarto interruptor (el interruptor de codificación) se reconoce un flanco. De aquí se calcula la dirección que se asignará a la salida.
- Para la salida se disponen tres valores (valor medido, raíz y cuadrado) con la misma dirección. Para poder seleccionar uno de estos valores deben preverse dos conmutadores (v. fig. 2-2).

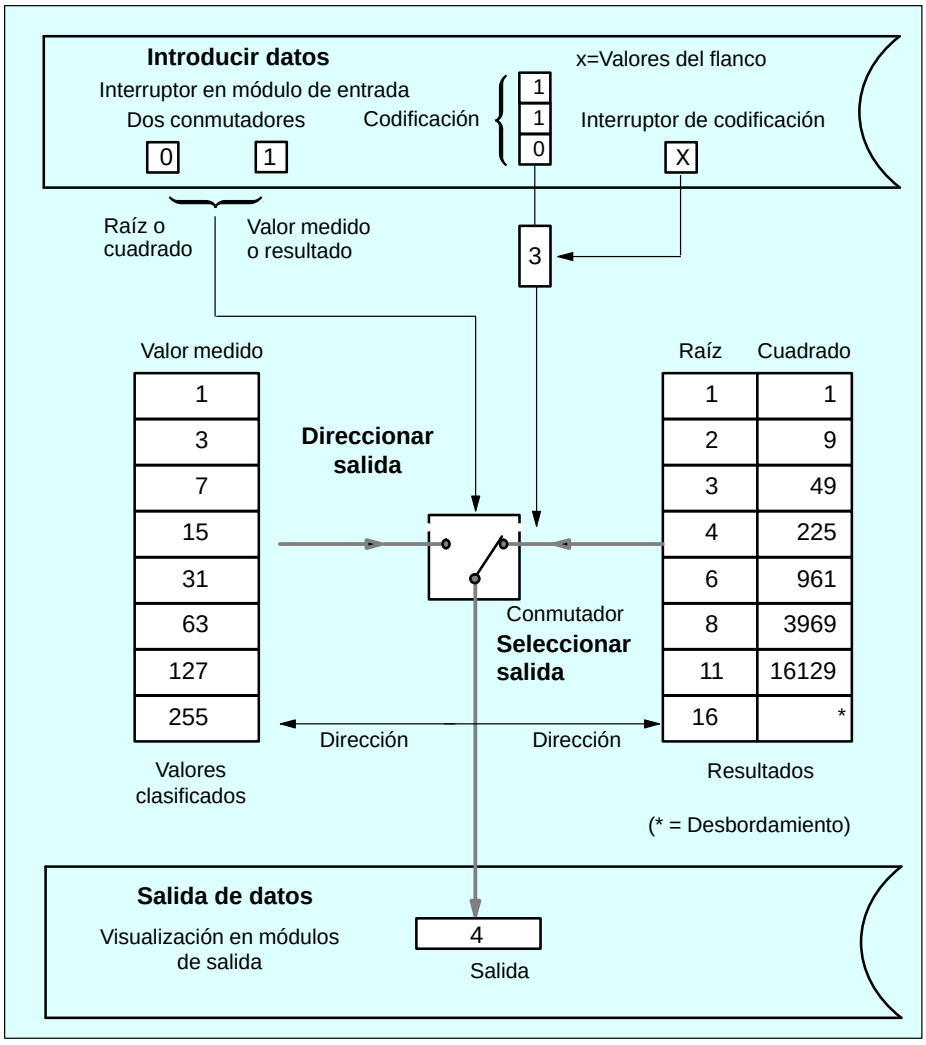


Figura 2-2 *Salida ajustable*

2.3 Solución con bloques SCL

Resumen

La mejor forma de solucionar la tarea descrita es con un programa SCL estructurado. Dicho programa tiene estructura modular; es decir, está organizado en bloques que tratan una o varias tareas parciales. SCL le ofrece al igual que los lenguajes de programación de STEP 7, distintos tipos de bloques. Encontrará más información al respecto en los capítulos 1, 7 y 8.

Proceso de solución

Para la solución puede seguir los siguientes pasos:

1. Listado de las tareas parciales
2. Selección y asignación de los bloques a las tareas parciales
3. Determinación de los interfaces entre los bloques
4. Determinación del interface de entrada/salida
5. Creación de los bloques

2.3.1 Determinación de las tareas parciales

Resumen

Las tareas parciales se representan mediante casillas en la figura 2-3. Las áreas rectangulares sobre fondo gris representan los bloques. La disposición de los bloques lógicos de izquierda a derecha corresponde al orden de llamada:

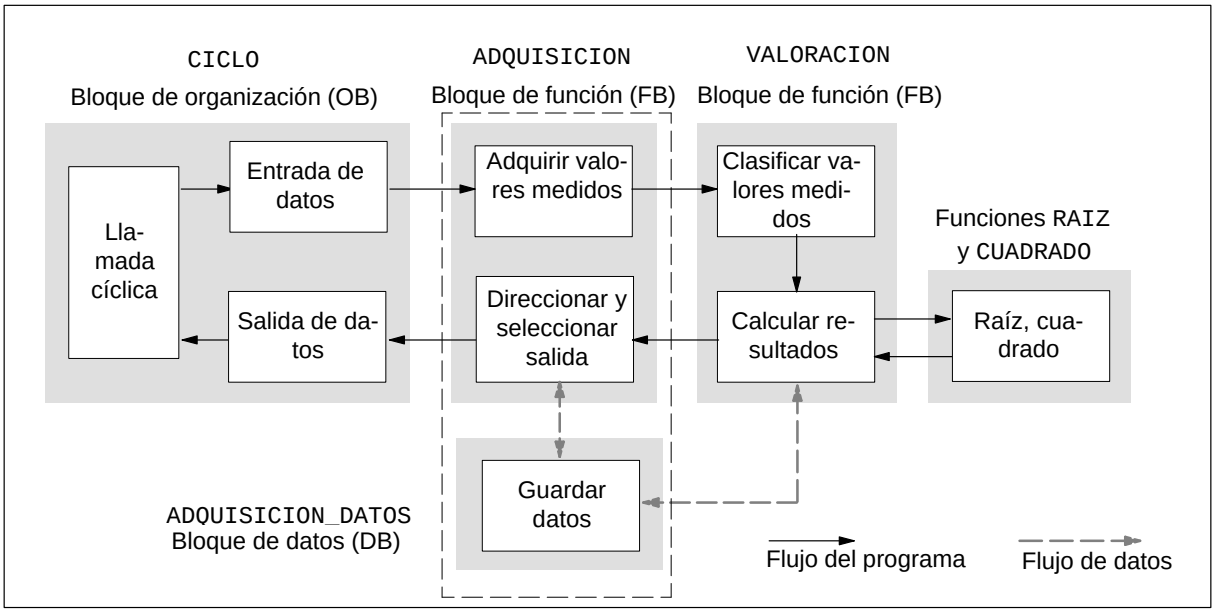


Figura 2-3 *Formación de bloques a partir de las tareas parciales*

2.3.2 Selección y asignación de los bloques a las tareas parciales

Resumen

A continuación aprenderá los criterios utilizados para seleccionar los bloques:

`CICLO`

Los programas de usuario sólo pueden iniciarse en un OB. Como los valores medidos tienen que registrarse cíclicamente, se necesita un OB para llamada cíclica (OB1). Una parte de la ejecución se programa en el OB: en el puesto central se ejecuta *Entrada de datos* y *Salida de datos*.

`ADQUISICION`

Para la tarea parcial de *Adquirir datos* se necesita un bloque con memoria, es decir, un FB, dado que de un ciclo de programa al siguiente deben conservarse determinados datos locales del bloque (p.ej., el búfer de circulación). El lugar para *Guardar datos*, también llamado memoria, es el bloque de datos de instancia `ADQUISICION_DATOS`.

El mismo FB puede asumir la tarea parcial *Direccionar salida* o *Seleccionar salida*, puesto que se dispone de los datos necesarios.

`VALORACION`

Al seleccionar el tipo de bloque para solucionar las tareas parciales *Clasificar valores medidos* y *Calcular resultados* hay que tener en cuenta que debe crearse un búfer de salida que incluya los resultados del cálculo de la raíz y el cuadrado para cada valor medido.

Por este motivo el único bloque posible es un FB. Dado que el FB es llamado por un FB de orden superior, no necesita ningún DB propio. Sus datos de instancia pueden depositarse en el bloque de datos de instancia del FB invocante.

`RAIZ y CUADRADO`

Lo más idóneo para solucionar la tarea parcial *Calcular la raíz* o *Calcular el cuadrado* es una FC, puesto que el resultado puede devolverse como valor de respuesta de la función. Además, para el cálculo no se necesita ningún dato que deba conservarse más de un ciclo de ejecución del programa.

Para calcular la raíz puede utilizarse la función estándar `RAIZ` de SCL. Para calcular el cuadrado debe crearse una función `CUADRADO`, que también ejecuta una comprobación del límite del rango.

2.3.3 Determinación de los interfaces entre los bloques

Resumen

La interface de un bloque se define mediante la declaración de sus **parámetros formales**. En SCL existen las siguientes tablas de declaración:

- Parámetros de entrada: declaración con VAR_INPUT
- Parámetros de salida: declaración con VAR_OUTPUT
- Parámetros de entrada/salida: declaración con VAR_IN_OUT

Cuando se llama a un bloque se le transfieren datos de entrada en forma de **parámetros actuales**. Después de retornar al bloque invocante los datos de salida se preparan para ser recibidos. Una FC puede transferir su resultado en forma de **valor de función** (para más información al respecto, véase el capítulo 16).

ADQUISICION

El OB CICLO no tiene ningún parámetro formal. Llama al FB ADQUISICION y le transfiere el valor medido y los datos de control en sus parámetros formales.

Tabla 2-1 Parámetros formales de ADQUISICION

Nombre del parámetro	Tipo de dato	Tipo de declaración	Descripción
intr_val_med	INT	VAR_INPUT	Valor medido
nue_val	BOOL	VAR_INPUT	Interruptor para aceptar el valor medido en el búfer de circulación
nue_clas	BOOL	VAR_INPUT	Interruptor para clasificar y evaluar valores medidos
sel_funcion	BOOL	VAR_INPUT	Conmutador para seleccionar raíz o cuadrado
seleccion	WORD	VAR_INPUT	Código para seleccionar valor de salida
nue_sel	BOOL	VAR_INPUT	Interruptor para aceptar codificación
sal_resultado	DWORD	VAR_OUTPUT	Salida del resultado de cálculo
sal_v_med	DWORD	VAR_OUTPUT	Salida del valor medido correspondiente

VALORACION

El FB ADQUISICION llama al FB VALORACION. Los datos comunes que tienen estos dos bloques de función son el área de valores medidos que va a clasificarse. Por ello se declara como parámetro de entrada/salida. Para el resultado de cálculo de la raíz y del cuadrado se crea un array estructurado en forma de parámetro de salida. Para los parámetros formales, ver la tabla 2-2:

Tabla 2-2 Parámetros formales de VALORACION

Nombre del parámetro	Tipo de dato	Tipo de declaración	Descripción
Bufer_de_clasificacion	ARRAY[..] OF REAL	VAR_IN_OUT	Campo de valores medidos; equivale al búfer de circulación
Bufer_de_calculo	ARRAY[..] OF STRUCT	VAR_OUTPUT	Campo para resultados: estructura con los componentes ”RAIZ” y ”CUADRADO” del tipo INT

RAIZ y CUADRADO

Las funciones son llamadas por VALORACION. Estas funciones necesitan un valor de entrada y proporcionan su resultado como valor de función (ver tabla 2-3).

Tabla 2-3 Parámetros formales y valores de función de RAIZ y CUADRADO

Nombre	Tipo de dato	Tipo de declaración	Descripción
Valor	REAL	VAR_INPUT	Dato de entrada para RAIZ
RAIZ	REAL	Valor de función	Raíz del valor de entrada
Valor	INT	VAR_INPUT	Dato de entrada para CUADRADO
CUADRADO	INT	Valor de función	Cuadrado del valor de entrada

2.3.4 Determinación del interface de entrada/salida

Resumen

La figura 2-4 muestra el interface de entrada/salida. Tenga en cuenta que en la entrada/salida por bytes el byte de arriba es el más significativo y el de abajo el menos significativo. Por el contrario, en entradas/salidas con palabras ocurre a la inversa.

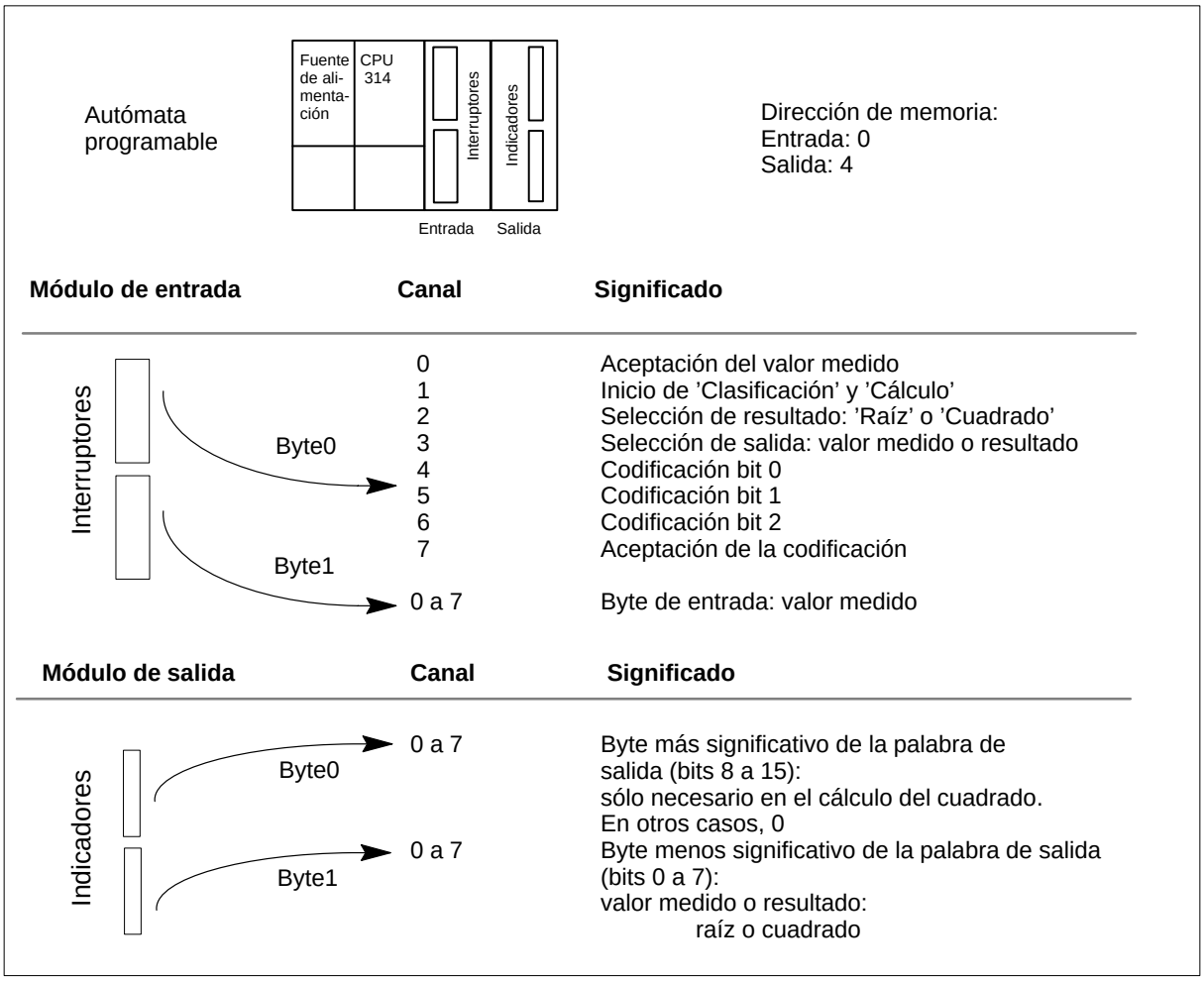


Figura 2-4 *Indicadores e interruptores*

2.3.5 Programación de los bloques

Programación de bloques

Una vez definidos los interfaces puede diseñar los bloques independientemente unos de otros. Lo mejor es hacerlo ”de arriba abajo”, es decir, siguiendo el orden `CICLO`, `ADQUISICION`, `VALORACION` y `CUADRADO`. A continuación se describen los bloques en este mismo orden.

Al compilar los bloques debe tener en cuenta que un bloque debe existir antes de utilizarlo; es decir, antes de ser llamado por otro bloque. Por ello, en la fuente SCL el orden de los bloques es `CUADRADO`, `VALORACION`, `ADQUISICION` y `CICLO` (más información al respecto encontrará en el capítulo 8).

Programación simbólica

La inteligibilidad del programa mejora si se atribuyen **nombres simbólicos** a las direcciones de módulos y a los bloques. Para hacerlo, debe introducir los símbolos en la tabla de símbolos de acuerdo con la figura 2-5 (v. también cap. 7). Los nombres puede formarlos con la convención de nombres para `IDENTIFICADORES` o según los identificadores de señal (p.ej., ”entrada 0.0”; ver anexo A).

Comentario de inicio con tabla de símbolos

En la figura 2-5 puede ver el comentario de inicio de la fuente SCL. Describe el nombre simbólico que debe declarar en la tabla de símbolos para compilar el programa sin errores.

```
(*#########################################################################################

Programa SCL para la adquisición y procesamiento de valores medidos:

      –  A través de entrada 0.0 (interruptor de entrada) se recibe un valor
         medido existente en el módulo de entrada.
      –  Mediante diferentes interruptores puede controlarse el procesamiento
         posterior de los valores medidos.
      –  Todos los valores se memorizan en el área de trabajo del bloque de
         función ADQUISICION, el bloque de datos de instancia ADQUISICION_DATOS.

El programa está programado simbólicamente. Para poder compilarlo debe existir la
asignación de los nombres simbólicos a las direcciones de los módulos y a los bloques que
se ejecutan de la CPU. Para ello se necesita la siguiente tabla de símbolos:

entrada               EB1   BYTE  // Valor medido
entrada 0.0           E0.0  BOOL  // Interruptor de entrada para aceptar el valor medido
interr_clasif         E0.1  BOOL  // Inicio de 'Clasificacion' y 'Calculo'
interr_funcion        E0.2  BOOL  // Selección de resultado 'Raiz' o 'Cuadrado'
interr_salida         E0.3  BOOL  // Selección de salida 'Valor_medido' o 'Resultado'
codificacion          EW0   WORD  // Codificación, bits relevantes 12, 13 y 14
interr_codif          E0.7  BOOL  // Aceptación del código
salida                AW4   INT   // Valor medido o resultado: 'Raiz' o 'Cuadrado'

ADQUISICION           FB10  FB10  // Adquirir valores medidos,
                                  // direccionar y seleccionar salida
ADQUISICION_DATOS     DB10  FB10  // Bloque de datos de instancia de ADQUISICION
VALORACION            FB20  FB20  // Evaluar valores medidos, calcular resultados
CUADRADO              FC41  FC41  // Función de cálculo del cuadrado
CICLO                 OB1   OB1   // Llamada cíclica y entrada/salida

#########################################################################################*)
```

Figura 2-5 *Comentario de inicio con tabla de símbolos*

2.4 Creación del bloque de organización CICLO

Proceso de solución

Se ha elegido un OB1 porque es llamado **cíclicamente** por el sistema STEP 7. Con él se ejecutan las siguientes tareas para el programa:

- Llamar y proporcionar datos de entrada y datos de control para el bloque de función ADQUISICION.
- Aceptar los datos del resultado del bloque de función ADQUISICION
- Salida de los valores para visualización.

Al inicio de la tabla de declaración se sitúa el área de datos temporales de 20 bytes "datos_del_sistema" (v. cap. 8).

```
ORGANIZATION_BLOCK CICLO

(*******************************************************************************
      CICLO equivale a OB1, es decir, es llamado cíclicamente por el sistema S7.
      Parte 1  :  llamada del bloque de función y transferencia de los valores de entrada
      Parte 2  :  aceptación de los valores de salida y salida con conmutación de salida
*******************************************************************************)

VAR_TEMP
      datos_del_sistema    :  ARRAY[0..20] OF BYTE;   // Area para OB1
END_VAR

BEGIN

(*    Parte 1  :  ***************************************************************)

ADQUISICION.ADQUISICION_DATOS(
      intr_val_med   := WORD_TO_INT (entrada),
      nue_val        := "entrada 0.0",//interruptor de entrada como símbolo
      nue_clas       := interruptor de clasificación,
      sel_funcion    := interruptor de función,
      nue_sel        := interruptor de codificación,
      seleccion      := codificación;

(*    Parte 2  :  ***************************************************************)

IF interr_salida THEN                              //Conmutación de salida
      salida   := ADQUISICION_DATOS.sal_resultado;    //Raíz o cuadrado
ELSE
      salida   := ADQUISICION_DATOS.sal_v_med.;       //Valor medido
END_IF;

END_ORGANIZATION_BLOCK
```

Figura 2-6 *Bloque de organización CICLO (OB1)*

Conversiones de tipo de datos

El valor medido tiene en la entrada el tipo BYTE. Debe convertirse a tipo INT: Para ello debe convertirlo de WORD a INT, y el compilador realizará implícitamente la anterior conversión de BYTE a WORD (v. cap. 18). Por el contrario, para la salida no se necesita ninguna conversión, puesto que ya fue declarada en la tabla de símbolos como INT (v. fig. 2-5).

2.5 Creación del bloque de función ADQUISICION

Solución

Se ha elegido el tipo de bloque FB porque hay datos que deben ser guardados desde un ciclo del programa al siguiente. Son las variables estáticas, declaradas en la tabla de declaración VAR, VAR_END (v. tabla 2-4).

Las variables estáticas son variables locales cuyo valor se mantiene en todos los recorridos del bloque. Dichas variables sirven para almacenar valores de un **bloque de función**. Se depositan en el bloque de datos de instancia.

```
FUNCTION_BLOCK ADQUISICION
(*******************************************************************************
      Parte 1 :   Registro de los valores medidos
      Parte 2 :   Iniciar 'Clasificacion' y 'Calculo'
      Parte 3 :   Valorar código y preparar salida
*******************************************************************************)
```

Figura 2-7 *Encabezado del bloque de función ADQUISICION*

Variables estáticas

Tabla 2-4 Variables estáticas de ADQUISICION

Nombre	Tipo de datos	Tipo de declaración	Pre–ajuste	Descripción
valores_medidos	ARRAY [..] OF INT	VAR	8(0)	Búfer de circulación para valores medidos
bufer_resultado	ARRAY [..] OF STRUCT	VAR	–	Array para estructuras con los componentes "raíz" y "cuadrado" del tipo INT
puntero	INT	VAR	0	Directorio del búfer de circulación, donde se registra el siguiente valor medido
ant_val	BOOL	VAR	FALSE	Valor precedente para aceptación de valor medido con "valor nuevo"
ant_clas	BOOL	VAR	FALSE	Valor precedente para clasificación con "clasificación nueva"
ant_sel	BOOL	VAR	FALSE	Valor precedente para aceptación del código con "selección nueva"
direccion	INT	VAR	0	Dirección para salida de valor medido o de resultado
valorar_instancia	VALORACION, = FB 20	VAR	–	Instancia local para el FB VALORACION

Tenga en cuenta los valores predefinidos registrados en las variables al inicializar el bloque (después de cargar la CPU). En la tabla de declaración VAR, END_VAR también se declara la instancia local para el FB VALORACION. El nombre se utilizará posteriormente para la llamada y para acceder a los parámetros de salida. Como memoria de datos se utiliza la instancia global ADQUISICION_DATOS.

Tabla de declaración de ADQUISICION

La tabla de declaración de este bloque consta de las siguientes partes:

- Constantes: entre CONST y END_CONST.
- Parámetros:
  de entrada: entre VAR_INPUT y END_VAR
  de salida: entre VAR_OUTPUT y END_VAR.
- Variables estáticas: entre VAR y END_VAR.
  Aquí se cuenta también la declaración de la instancia local para el bloque VALORACION.

```
CONST
   LIMITE              := 7;
   NUMERO              := LIMITE + 1;
END_CONST

VAR_INPUT
   intr_val_med        :  INT;     //    Nuevo valor medido
   nue_val             :  BOOL;    //    Aceptar valor medido en búfer de circulación
                                   //    "valores_medidos"
   nue_clas            :  BOOL;    //    Clasificar valores medidos
                       :  BOOL;    //    Seleccionar la función de cálculo RAIZ/CUADRADO
   nue_sel             :  BOOL;    //    Aceptar dirección de salida
   seleccion           :  WORD;    //    Dirección de salida
END_VAR

VAR_OUTPUT
   sal_resultado       :  INT;     //    Valor calculado
   sal_v_med           :  INT;     //    Valor medido correspondiente
END_VAR

VAR
   valores_medidos     :  ARRAY[0..LIMITE] OF INT := 8(0);
   bufer_resultado     :  ARRAY[0..LIMITE] OF
        STRUCT
           RAIZ        :  INT;
           CUADRADO    :  INT;
        END_STRUCT;
   puntero             :  INT      :=    0;
   ant_val             :  BOOL     :=    TRUE;
   ant_clas            :  BOOL     :=    TRUE;
   ant_sel             :  BOOL     :=    TRUE;
   direccion           :  INT      :=    0;    //Dirección de salida convertida
   valorar_instancia:  VALORAR;              //Declarar instancia local
END_VAR
```

Figura 2-8 *Tabla de declaración del bloque de función ADQUISICION*

Diseño del área de instrucciones

El proceso del área de instrucciones consta de tres partes:

1ª parte: registrar valores medidos

Cuando el parámetro de entrada cambia de ”valor antiguo” a ”valor nuevo”, en el búfer de circulación se lee un nuevo valor medido.

2ª parte: iniciar clasificación y cálculo

Llamando al bloque de función `VALORACION`, cuando el parámetro de entrada ha cambiado de ”clasificación antigua” a ”clasificación nueva”.

3ª parte: evaluar codificación y preparar salida

La codificación se lee por palabras: según la convención SIMATIC, esto significa que el grupo de interruptores superior (byte0) contiene los 8 bits más significativos de la palabra de entrada, y el grupo de interruptores inferior (byte1) los menos significativos. La figura 2-9 muestra donde se encuentran los interruptores con los que se ajusta la codificación:

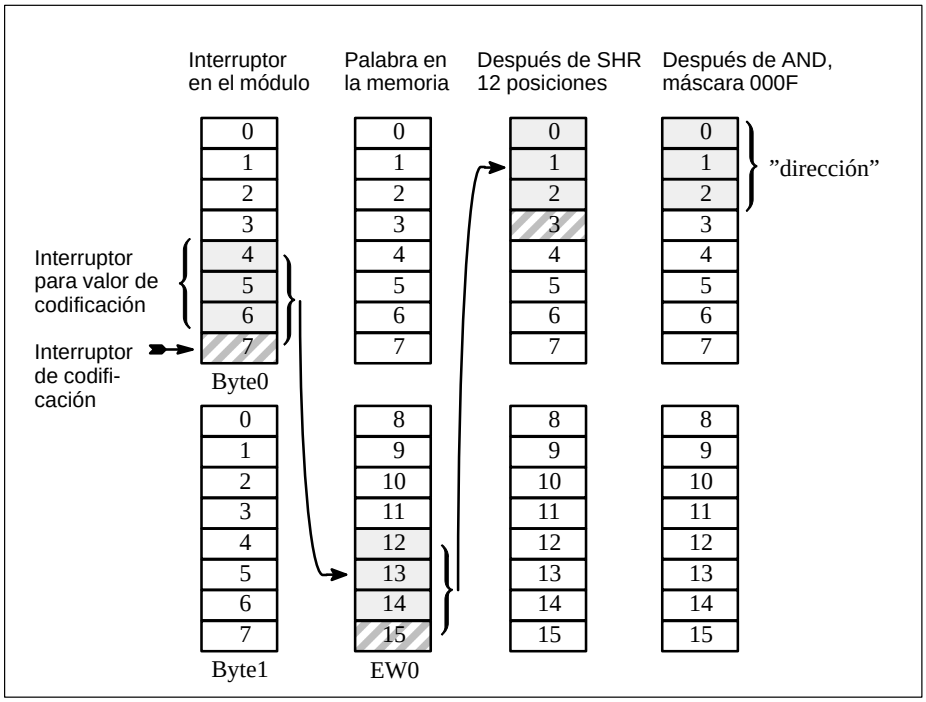


Figura 2-9 *Valoración del código*

Cálculo de la dirección

La figura 2-9 muestra el cálculo de la dirección: la palabra de entrada EW0 contiene en los bits 12 a 14 el código que se acepta cuando en el interruptor de codificación (bit 15) se reconoce un blanco. Desplazando hacia la derecha con la función estándar SHR y ocultando los bits relevantes con una máscara `AND` se determina la ”dirección”.

Con esta dirección se escriben en los parámetros de salida los elementos del array (resultado de cálculo y valor medido correspondiente). El hecho de que la salida sea raíz o cuadrado depende de la ”selección de función”.

Un flanco del interruptor de codificación se reconoce por el cambio de ”selección antigua” a ”selección nueva”.

Organigrama de `ADQUISICION`

La figura 2-10 representa el algoritmo en forma de organigrama:

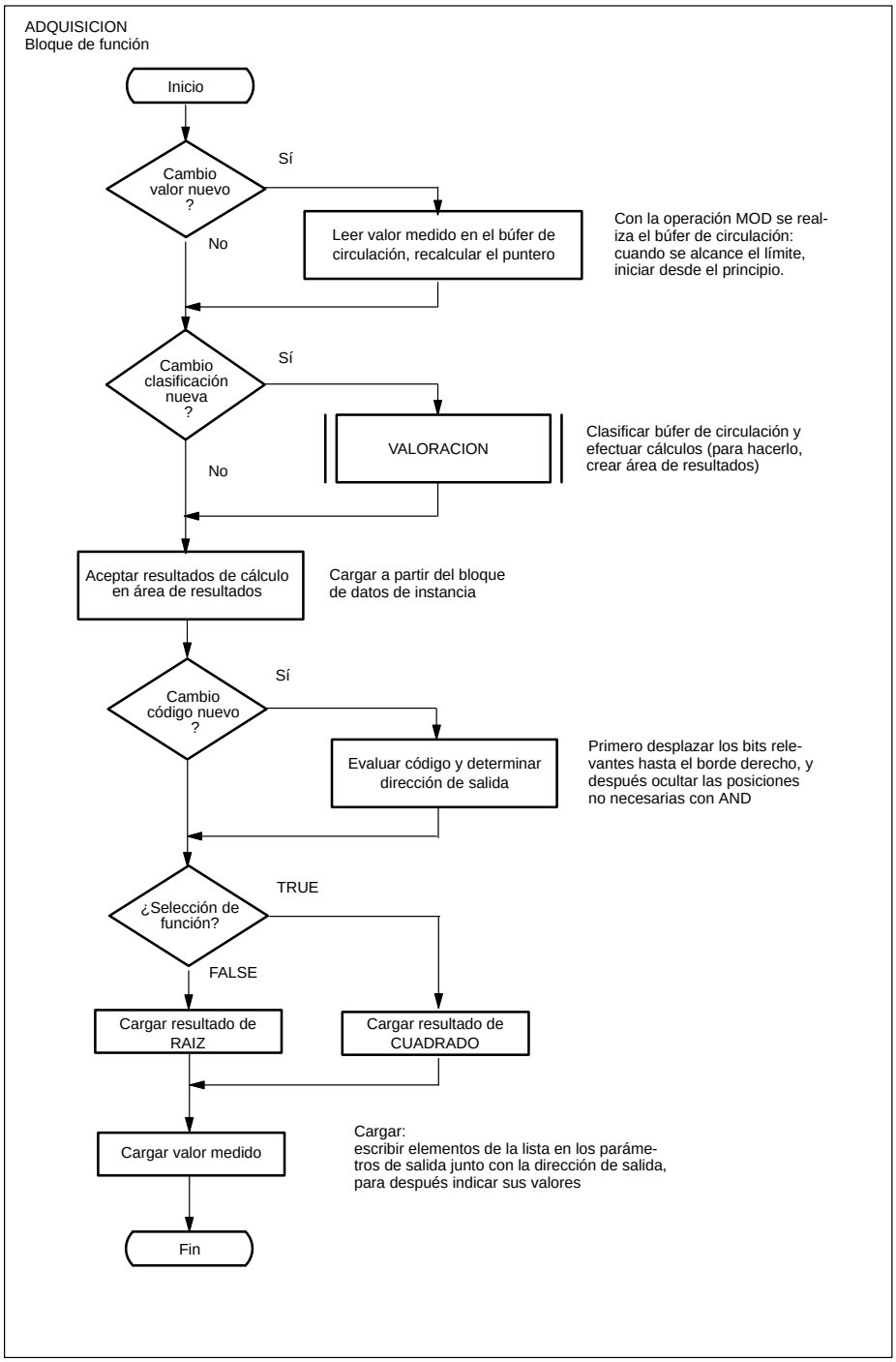


Figura 2-10 *Algoritmo para registrar los valores medidos*

Area de instrucciones de ADQUISICION

La figura 2-11 muestra la formulación en SCL del organigrama representado en la figura 2-10; es decir, el **área de instrucciones** del bloque lógico.

```
BEGIN

(* Parte 1 :   'Adquisicion' de 'valores_medidos' ************************************
               Al cambiar "nue_val" se produce la entrada del valor medido
               Con la operación MOD se ejecuta un búfer de circulación para valores
               medidos.*)

IF valor nuevo <> valor antiguo THEN
      puntero                       :=    puntero MOD NUMERO;
      valores_medidos [puntero]     :=    intr_val_med;
      puntero                       :=    puntero + 1;
END_IF;
ant_val  :=    nue_val;

(* Parte 2 :   Iniciar 'Clasificacion' y 'Calculo' ******************************
               Al cambiar 'nue_clas' se inicia la clasificación del búfer de
               circulación y la ejecución de los cálculos con los valores medidos.
               Los resultados se guardan en un nuevo array, 'bufer_calculo'.*)

IF nue_clas <> ant_clas THEN
      puntero  := 0; //Inicializar puntero del búfer de circulación
      valorar_instancia (bufer_clasif := valores_medidos); //Llamar VALORACION
END_IF;
ant_clas := nue_clas;

bufer_resultado := valorar_instancia.bufer_calculo; // CUADRADO y RAIZ

(* Parte 3 :   Valorar código y preparar salida *********************
               Al cambiar 'nue_sel' se determina de nuevo el código para el
               direccionamiento del elemento de array para la salida: los datos
               relevantes de 'seleccion' se ocultan y se transforman en entero.
               Dependiendo de la posición del interruptor de "sel_funcion" en
               la salida se dispondrá 'RAIZ' o 'CUADRADO' *)

IF nue_sel <> ant_sel THEN
      direccion   := WORD_TO_INT(SHR(IN := seleccion, N := 12) AND 16#0007);
END_IF;
ant_sel  :=    nue_sel;

IF sel_funcion THEN
      sal_resultado  :=    bufer_resultado [direccion].cuadrado;
ELSE
      sal_resultado  :=    bufer_resultado [direccion].raiz;
END_IF;


sal_v_med   :=    valores_medidos [direccion]; //indicación del valor medido

END_FUNCTION_BLOCK
```

Figura 2-11 *Area de instrucciones del bloque de función ADQUISICION*

2.6 Creación del bloque de función VALORACION

Tabla de declaración de VALORACION

La tabla de declaración de este bloque consta de las siguientes partes:

- Constantes: entre CONST y END_CONST
- Parámetros:
  de entrada/salida: entre VAR_IN_OUT y END_VAR,
  de salida: entre VAR_OUTPUT y END_VAR
- Variables temporales: entre VAR_TEMP y END_VAR

```
FUNCTION_BLOCK VALORACION
(*******************************************************************************
      Parte 1 :   Clasificar el búfer de circulación con los valores medidos
      Parte 2 :   Iniciar el cálculo de los resultados
*******************************************************************************)
```

Figura 2-12 *Encabezado del bloque de función VALORACION*

```
CONST
      LIMITE         := 7;
END_CONST

VAR_IN_OUT
      bufer_clasif           :     ARRAY[0..LIMITE] OF INT;
END_VAR

VAR_OUTPUT
      bufer_calculo          :     ARRAY[0..LIMITE] OF
         STRUCT
            raiz             :     INT;
            cuadrado         :     INT;
         END_STRUCT;
END_VAR

VAR_TEMP
      cambiar                :     BOOL;
      indice, ayuda          :     INT;
      n_valor, n_resultado   :     REAL;
END_VAR
```

Figura 2-13 *Tabla de declaración del bloque de función VALORACION*

Proceso

El parámetro de entrada/salida "bufer_clasif" se combina lógicamente con el búfer de circulación "valores_medidos"; es decir, el contenido original del búfer se sobreescribe sustituyéndose por los valores clasificados.

Para los resultados de cálculo se crea el nuevo array "bufer_calculo", configurado como parámetro de salida. Sus elementos están estructurados para contener la raíz y el cuadrado de cada valor medido.

En la figura 2-14 puede ver la relación existente entre los arrays descritos.

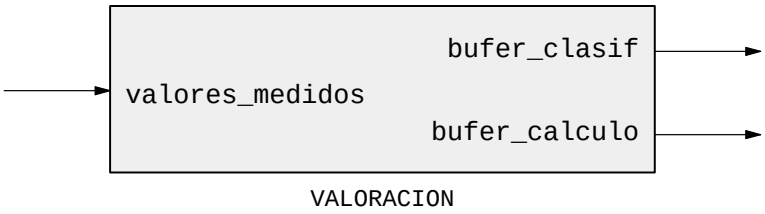


Figura 2-14 *Interface del FB VALORACION*

Este interface muestra el núcleo del intercambio de datos para el procesamiento de los valores medidos. Los valores se guardan en el bloque de datos de instancia AD-QUISICION_DATOS, puesto que en el FB invocante ADQUISICION invocante se había creado una instancia local para el FB VALORACION.

Diseño del área de instrucciones

Primero se clasifican los valores medidos en el búfer de circulación y después se efectúan los cálculos:

- Método del algoritmo de clasificación

Aquí se utiliza el método de intercambio permanente de valores para clasificar el búfer de valores medidos. Es decir, se comparan entre sí dos valores consecutivos y se intercambian hasta que se llega al orden de clasificación deseado. El búfer utilizado es el parámetro de entrada/salida "bufer_clasif".

- Inicio del cálculo

Cuando ha concluido la clasificación, para efectuar el cálculo se recorre un bucle en el que se llama a las funciones CUADRADO para calcular el cuadrado, y RAIZ para calcular la raíz. Sus resultados se guardan en el array estructurado "bufer_calculo".

Organigrama de `VALORACION`

La figura 2-15 representa el algoritmo en forma de organigrama:

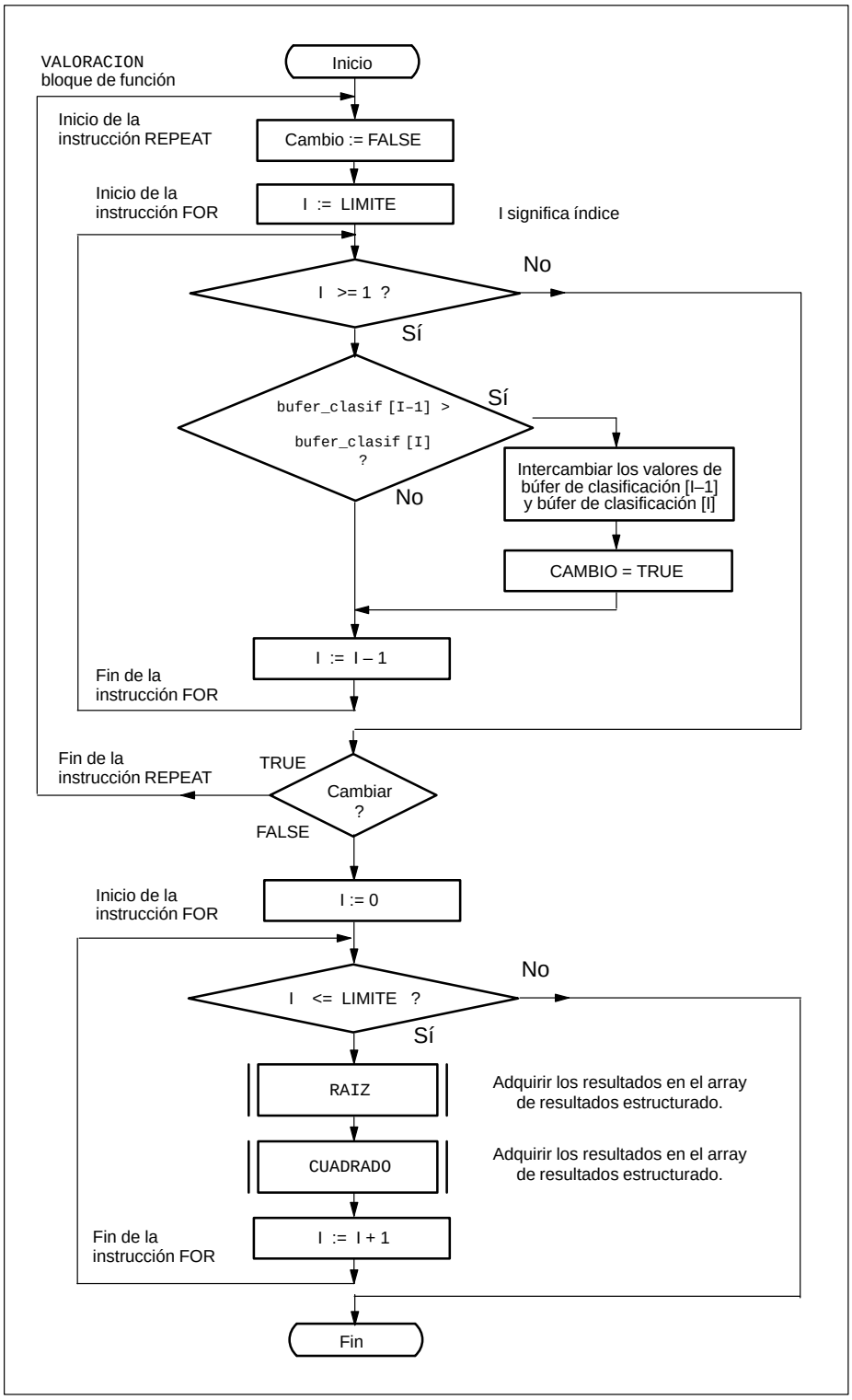


Figura 2-15 *Algoritmo para valorar los valores medidos*

Area de instrucciones de VALORACION

La figura 2-16 muestra cómo formular en SCL el organigrama representado en la figura 2-15; es decir, el área de instrucciones del bloque lógico:

```
BEGIN

(* Parte 1  'Clasificacion' : ******************************************************
            Clasificación según el proceso "bubble sort": intercambiar por parejas los
            valores hasta que el búfer de valores medidos esté clasificado*)

REPEAT
      cambiar     := FALSE;

      FOR indice  := LIMITE TO 1 BY -1 DO
         IF bufer_clasif [indice-1] > bufer_clasif [indice] THEN
            ayuda                  :=    bufer_clasif [indice];
            bufer_clasif [indice]  :=    bufer_clasif [indice-1] ;
            bufer_clasif [indice-1] :=   ayuda;
            cambiar                :=    TRUE;
         END_IF;
      END_FOR;

UNTIL NOT cambiar
END_REPEAT;


(* Parte 2  'Calculo' : ************************************************************
            Cálculo de la raíz con la función estándar RAIZ y
            obtención del cuadrado con la función CUADRADO.*)

FOR indice  := 0 TO LIMITE BY 1 DO
      n_valor        := INT_TO_REAL (bufer_clasif [indice]);
      n_resultado    := RAIZ(n_valor);
      bufer_calculo [indice].raiz        :=    REAL_TO_INT (n_resultado);
      bufer_calculo [indice].cuadrado    :=    CUADRADO (bufer_clasif [indice]);
END_FOR;

END_FUNCTION_BLOCK
```

Figura 2-16 *Area de instrucciones del bloque de función VALORACION*

2.7 Creación de la función CUADRADO

Diseño del área de instrucciones

Primero se comprueba si el valor de entrada supera el límite para el cual el resultado sobrepasaría el rango de números enteros. En este caso se registraría el valor máximo de números enteros. En caso contrario se efectuaría la operación de elevar al cuadrado. El resultado se transfiere como valor de función.

```
FUNCTION CUADRADO : INT

(*******************************************************************************
Esta función proporciona como valor de función el cuadrado del valor de entrada, o,
en caso de desbordamiento, el valor máximo que puede representarse con enteros.
*******************************************************************************)

VAR_INPUT
      valor   :   INT;
END_VAR

BEGIN
IF valor <= 181 THEN
      CUADRADO    :=    valor * valor;    // Cálculo del valor de la función
ELSE
      CUADRADO    :=    32_767;           // Definir valor máximo en desbordamiento
END_IF;

END_FUNCTION
```

Figura 2-17 *Función CUADRADO*

2.8 Datos de test

Requisitos

Para el test necesita un módulo de entrada en la dirección 0 y un módulo de salida en la dirección 4 (v. fig. 2-4).

Antes de efectuar el test, colocar los 8 interruptores superiores del módulo de entrada hacia la izquierda (”0”) y los 8 interruptores inferiores hacia la derecha (”1”).

Cargue previamente los bloques en la CPU dado que también se comprueban los valores iniciales de las variables inmediatamente después.

Pasos del test

Efectue los pasos del test según la tabla 2-5.

Tabla 2-5 Pasos del test

Paso	Acción	Resultado
1	Active el código ”111” (E0.4, E0.5 y E0.6) y acéptelo con el interruptor de codificación (E0.7).	Todas las salidas del módulo de salida (byte de menor valor) se activan y se iluminan los indicadores.
2	Visualice la raíz correspondiente colocando el interruptor de salida (E0.3) en posición ”1”.	Los indicadores de la salida corresponden al valor binario ”10000” (=16).
3	Visualice el cuadrado correspondiente colocando el interruptor de función (E0.2) en posición ”1”.	En la salida se iluminan 15 indicadores. Esto significa que se ha producido un desbordamiento, puesto que 255 x 255 arroja un valor demasiado grande para el rango de números enteros.
4a	Recupere la posición ”0” del interruptor de salida (E0.4).	Se visualiza de nuevo el valor medido: todas las indicaciones en las salidas del byte de salida de menor valor están activadas.
4b	Ajuste en la entrada el valor 3 (es decir, valor binario ”11”) como nuevo valor medido.	La salida no cambia todavía.
5a	Observar la lectura del valor medido: ajuste el código al valor ”000” y confírmelo con el interruptor de codificación (E0.7), para poder observar posteriormente el valor introducido.	En la salida se visualiza 0; es decir, no se ilumina ningún indicador.
5b	Conmute el interruptor de entrada ”entrada 0.0” (E0.0). De esta forma se lee el valor ajustado en la cuarta fase del test.	En la salida se visualiza el valor medido 3 (valor binario ”11”).
6	Inicie la clasificación y el cálculo conmutando el interruptor de clasificación (E0.1).	En la salida aparece de nuevo 0, puesto que debido al proceso de clasificación el valor medido se ha desplazado hacia arriba en el campo.
7	Visualizar el valor medido después de la clasificación: ajuste el código ”110” (E0.6=1, E0.5=1, E0.4= 0 de EB0, corresponde a bit 14, bit 13, bit 12 de EW0) y confírmelo conmutando el interruptor de codificación.	En la salida se visualizará de nuevo el valor medido ”11”, puesto que es el segundo valor más elevado del array.
8a	Visualizar los resultados correspondientes: conmutando el interruptor de salida (E0.3) se visualiza el cuadrado del valor medido de la fase 7 del test.	Se visualiza el valor de salida 9 o el valor binario ”1001”.
8b	Conmutando el interruptor de función (E0.2) obtendrá también la raíz.	Se visualiza el valor de salida 2 o el valor binario ”10”.

Test complementario

Las explicaciones referentes a los interruptores de manejo de la tabla 2-6 y las plantillas de test de la tabla 2-7 le facilitarán la definición de las fases del test:

- La entrada se realiza a través de interruptores: los 8 interruptores superiores sirven para el manejo, y los 8 inferiores para ajustar el valor medido.
- La salida se realiza mediante indicadores: en el grupo superior aparece el byte de salida de mayor valor, y en el grupo inferior el de menor valor.

Tabla 2-6 Interruptores de manejo

Interruptor de manejo	Nombre	Explicación
Canal 0	Interruptor de entrada	Conmutación para confirmación de valor medido
Canal 1	Interruptor de clasificación	Conmutación para clasificación/valoración
Canal 2	Interruptor de función	Interruptor hacia la izquierda (”0”): raíz, Interruptor hacia la derecha (”1”): cuadrado
Canal 3	Interruptor de salida	Interruptor hacia la izquierda (”0”): valor medido, Interruptor hacia la derecha (”1”): resultado
Canal 4	Codificación	Dirección de salida bit 0
Canal 5	Codificación	Dirección de salida bit 1
Canal 6	Codificación	Dirección de salida bit 2
Canal 7	Interruptor de codificación	Conmutación para confirmación de código

La tabla 2-7 contiene 8 valores medidos que han sido clasificados.

Introduzca los valores en cualquier orden. Ajuste la combinación binaria deseada y acepte el valor correspondiente conmutando el interruptor de entrada. Una vez introducidos todos los valores inicie la clasificación y valoración conmutando el interruptor de clasificación. Después puede visualizar los valores medidos clasificados o los resultados (raíz o cuadrado).

Tabla 2-7 Plantillas de test para raíz y cuadrado

Valor medido	Raíz	Cuadrado
0000 0001 = 1	0, 0000 0001 = 1	0000 0000, 0000 0001 = 1
0000 0011 = 3	0, 0000 0010 = 2	0000 0000, 0000 1001 = 9
0000 0111 = 7	0, 0000 0011 = 3	0000 0000, 0011 0001 = 49
0000 1111 = 15	0, 0000 0100 = 4	0000 0000, 1110 0001 = 225
0001 1111 = 31	0, 0000 0110 = 6	0000 0011, 1100 0001 = 961
0011 1111 = 63	0, 0000 1000 = 8	0000 1111, 1000 0001 = 3969
0111 1111 = 127	0, 0000 1011 = 11	0011 1111, 0000 0001 = 16129
1111 1111 = 255	0, 0001 0000 = 16	0111 111, 1111 1111 = Indicación de desbordamiento

Parte 2:
Manejo y comprobación

Instalación del software SCL

3

Resumen

Un programa de instalación (setup) le permite instalar el software SCL guiado por menús. Hay que llamar al programa de instalación con el procedimiento estándar para instalación de software habitual en Windows 95.

Requisitos para la instalación

Para poder instalar el software SCL necesita:

- una unidad de programación o un PC con el paquete básico STEP 7 ya instalado
  - procesador 80486 (o superior) y
  - memoria ampliada RAM de 16 MB
- un monitor color, teclado y ratón, soportados por Microsoft Windows 95
- un disco duro con una capacidad de memoria libre de 78 MB
  (10 MB para datos de usuario, 60 MB para archivos de intercambio (Swap),
  8 MB para el paquete opcional SCL)
- como mínimo, 1 MB de memoria libre en la unidad de disco C: para la instalación (los archivos de instalación se borran una vez concluida ésta)
- el sistema operativo Windows 95
- un interface MPI entre la PG/PC y el PLC:
  - o bien un cable PC/MPI para conectar al interface de comunicación de su aparato,
  - o bien una tarjeta MPI instalada en su aparato. Algunas unidades de programación ya disponen del interface MPI.

Indice del capítulo

3.1 Autorización / Permiso de utilización

Introducción

Para utilizar el paquete de software SCL se necesita una autorización específica del producto (derecho de utilización). El software protegido sólo puede utilizarse cuando en el disco duro de la PG o el PC correspondiente se ha reconocido la autorización necesaria para el programa o el paquete de software.

Disquete de autorización

Para la autorización necesita poseer el disquete de autorización protegido contra escritura que se suministra junto con el software. Dicho disquete incluye la autorización y el programa AUTHORS, necesario para visualizar, instalar y desinstalar la autorización.

El número de las posibles autorizaciones está definido en el disquete de autorización mediante un contador de autorizaciones. Cada autorización rebaja el contador 1 unidad. Cuando el contador alcanza el valor 0, con dicho disquete no es posible realizar ninguna autorización más.

En el manual del usuario **/231/** encontrará más indicaciones y reglas para el manejo de las autorizaciones.

Cuidado

Observe las indicaciones que figuran en el archivo LEAME.TXT incluido en el disquete de autorización. En caso de incumplimiento de esta observación existe el peligro de pérdida irrecuperable de la autorización.

Autorización en caso de primera instalación

Deberá proceder a ejecutar la autorización cuando así lo solicite el mensaje correspondiente durante la primera instalación. Proceda de la siguiente forma:

1. Introduzca el disquete de autorización cuando así se le requiera en la pantalla.
2. A continuación, confirme la autorización.

La autorización de uso se transfiere a una unidad de disco física (es decir, su ordenador ”se da cuenta” de que usted tiene autorización).

Ejecución posterior de la autorización

Cuando arranque el software SCL y no exista autorización, recibirá el correspondiente mensaje. Para ejecutar con posterioridad la autorización, arranque el programa AUTHORS en el disquete de autorización. Con él puede visualizar, instalar y desinstalar autorizaciones. El programa está guiado por menús.

Nota

Al instalar la autorización para SCL indique siempre como unidad de disco de destino la unidad de disco C:.

Desinstalar autorización

Cuando se necesita una nueva autorización (p.ej., cuando desee formatear de nuevo la unidad de disco en la que se encuentra la autorización), primero debe ”salvar” la autorización. Para ello necesita el disquete de autorización original.

Para transferir de nuevo la autorización al disquete de autorización, proceda de la siguiente forma:

1. Introduzca el disquete de autorización original en la unidad de disco de 3,5 ”.
2. Abra el programa AUTHORS.EXE del disquete de autorización.
3. Seleccione el comando de menú **Autorización ▸ Desinstalar**.
4. A continuación, en el cuadro de diálogo que aparece indique la unidad de disco en la que se encuentra la autorización y confirme el diálogo. Se visualizará una lista con todas las autorizaciones de la unidad de disco en cuestión.
5. Marque la autorización que desea desinstalar y confirme el diálogo. Si el proceso ha concluido sin errores recibirá el siguiente mensaje:
   ”**Autorización** <Nombre> **retirada de disquetera** <X:>.”
6. Confirme el mensaje.

   A continuación volverá a visualizarse el cuadro de diálogo con la lista de las autorizaciones restantes existentes en la unidad de disco. Si no desea desinstalar ninguna autorización más, cierre el cuadro de diálogo.

Después podrá utilizar de nuevo ese disquete para instalar autorizaciones.

Si su disco duro está defectuoso...

Si en su disco duro se presenta un error antes de que pueda salvar la autorización, diríjase a su representante SIEMENS.

3.2 Instalación / Desinstalación del software SCL

Resumen

SCL incluye un programa de instalación (setup) que efectúa automáticamente la instalación. En la pantalla aparecerán mensajes que le solicitarán datos de entrada y le guiarán paso a paso durante todo el proceso de instalación.

Preparativos

Antes de comenzar la instalación debe arrancar Windows 95 y debe estar cargado el paquete básico STEP 7.

Arrancar el programa de instalación

Proceda de la siguiente forma:

1. En Windows 95 arranque el diálogo de instalación de software haciendo doble clic sobre el botón ”Agregar o quitar programas” del ”Panel de control”.
2. Haga clic sobre ”Instalar”.
3. Introduzca el soporte de datos (disquete 1) o la CD–ROM y haga clic sobre ”Siguiente”. Windows 95 buscará automáticamente el programa de instalación SETUP.EXE.
4. Siga paso a paso las instrucciones que le indique el programa de instalación.

El programa le guía paso a paso a través del proceso de instalación. Siempre puede pasar a la fase posterior o anterior.

Si ya hay instalada una versión de SCL

Si el programa de instalación detecta que ya se ha efectuado una instalación de SCL en el sistema de creación, se indicará el mensaje correspondiente y usted podrá elegir entre las siguientes opciones:

- cancelar la instalación (para posteriormente desinstalar la versión de SCL antigua bajo Windows 95 y arrancar de nuevo la instalación), o
- continuar la instalación, con lo que la nueva versión se escribirá sobre la versión antigua.

No obstante, antes de proceder a la instalación es conveniente que desinstale cualquier versión anterior existente. Sobreescribir una versión anterior tiene la desventaja que en una desinstalación posterior no se suprimen las partes existentes procedentes de una instalación previa.

Durante el proceso de instalación se le presentarán en pantalla, dentro de cuadros de diálogo, cuestiones u opciones que podrá seleccionar. Lea las instrucciones que siguen para poder responder con más rapidez y facilidad a los diálogos.

Desinstalación

Utilice el procedimiento habitual de desinstalación bajo Windows 95:

1. En Windows 95, arranque el diálogo para instalación de software haciendo doble clic sobre el botón ”Agregar o quitar programas” del ”Panel de control”.
2. En la lista de programas instalados que se muestra a continuación, seleccione la entrada STEP 7 y active el botón de comando para quitar programas.
3. Si aparecen los cuadros de diálogo ”Desinstalar programa seleccionado” y tiene alguna duda, haga clic sobre el botón de comando ”No”.

Sobre el alcance de la instalación

Todos los lenguajes del interface de usuario y todos los ejemplos necesitan aprox. 8 MB de memoria.

Sobre la autorización

En el proceso de instalación se comprueba si ya existe una autorización en el disco duro. Si se detecta una autorización, aparece un mensaje indicando que el software sólo puede utilizarse con autorización. Si lo desea, puede instalar de inmediato la autorización o continuar el proceso de instalación y recuperar posteriormente la autorización.

En el primer caso, cuando se le solicite en pantalla introduzca el disquete de autorización y confirme. En el apartado 3.1 encontrará indicaciones sobre la autorización.

Sobre la conclusión de la instalación

Cuando la instalación ha concluido con éxito, se indica en pantalla con el pertinente mensaje.

Errores durante la instalación

Los siguientes errores producen la interrupción de la instalación:

- Cuando inmediatamente después de arrancar el Setup se produce un error de inicialización, es muy probable que el programa SETUP.EXE no se haya arrancado bajo Windows 95.
- No hay memoria suficiente: necesita al menos 8 MB de memoria libre en su disco duro.
- Disquete defectuoso: si constata que un disquete está defectuoso, diríjase a su representante Siemens.
- Error del usuario: comience de nuevo la instalación y siga cuidadosamente las instrucciones.

Manejo de SCL

4

Resumen

El presente capítulo le familiariza con el manejo de SCL. Le informa sobre los interfaces de usuario del editor SCL.

Indice del capítulo

4.1 Arranque del software SCL

Arranque desde el interface de Windows

Después de instalar el software SCL en su PG, puede arrancar SCL mediante el botón de comando ”Inicio” de la barra de tareas de Windows 95 (bajo ”SIMATIC / STEP 7”).

Arranque desde el Administrador SIMATIC

La forma más rápida de arrancar SCL en el Administrador SIMATIC es posicionar el puntero del ratón sobre una fuente SCL y hacer doble clic. En **/231/** encontrará información al respecto.

La figura 4-1 muestra la ventana SCL que aparece después de arrancar el software.

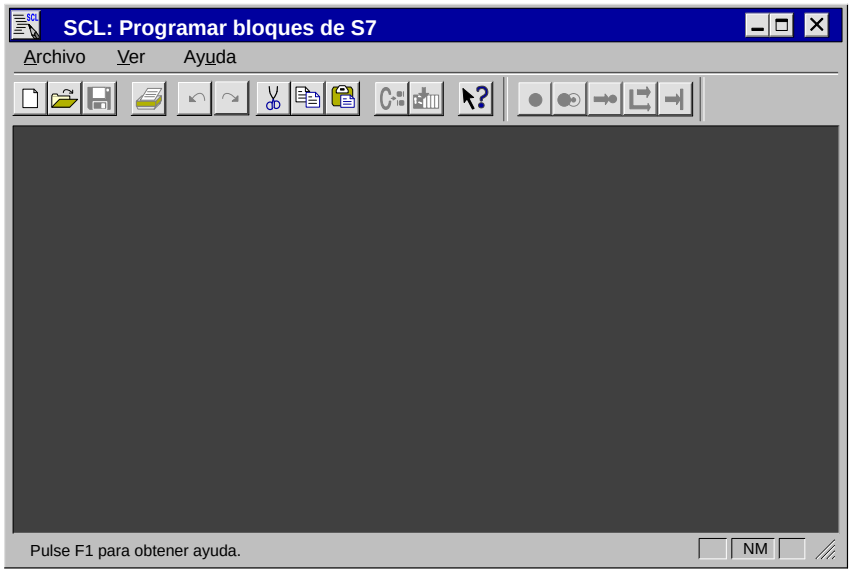


Figura 4-1 *Ventana SCL*

Nota

Puede consultar más información sobre las condiciones y opciones estándar de Windows 95, o bien en su manual de usuario de Windows, o bien online en el tutorial de Windows 95.

4.2 Adaptación del interface de usuario

Resumen

Al igual que otras ventanas de STEP 7, las ventanas SCL constan de los siguientes componentes estándar (v. fig. 4-2):

- **Barra de título:**
  incluye el título de la ventana y símbolos para el control de ventanas.
- **Barra de menús:**
  incluye todos los menús de que se dispone en la ventana.
- **Barra de herramientas:**
  incluye botones que le permitirán ejecutar rápidamente los comandos más frecuentemente utilizados.
- **Area de trabajo:**
  incluye las diferentes ventanas en las que puede editar el texto del programa o leer datos del compilador o datos de test.
- **Barra de estado:**
  indica el estado y otros datos sobre el objeto seleccionado.

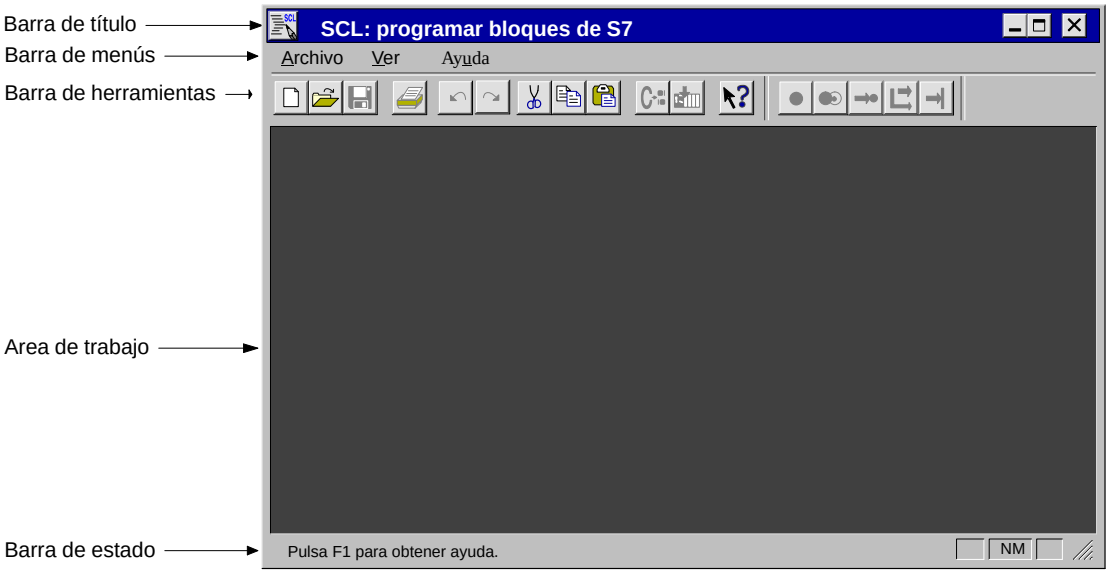


Figura 4-2 *Componentes de la ventana SCL*

Cambios posibles de componentes

Usted puede adaptar a sus necesidades personales los siguientes componentes:

- Visualización de la barra de herramientas
- Visualización de la barra de puntos de parada
- Visualización de la barra de estado

Adaptar la barra de herramientas

Usted puede activar o desactivar la visualización de la barra de herramientas utilizando el comando de menú **Ver ▸ Barra de herramientas**. Una marca de verificación situada detrás del comando de menú indica si el comando está activado o desactivado.

Adaptar la barra de puntos de parada

Igualmente puede activar o desactivar la barra de puntos de parada utilizando el comando de menú **Ver ▸ Barra de puntos de parada**. Una marca de verificación situada detrás del comando de menú indica si el comando está activado o desactivado.

Adaptar la barra de estado

También puede activar o desactivar la barra de estado utilizando el comando de menú **Ver ▸ Barra de estado**. Una marca de verificación situada detrás del comando de menú indica si el comando está activado o desactivado.

Adaptar el entorno de desarrollo

El editor y el compilador le permiten efectuar determinados ajustes de gran utilidad:

- Ajustes al crear bloques
- Ajustes del compilador
- Ajustes del editor

Crear bloques

También puede ajustar, p.ej., si desea que al compilar se sobreescriban bloques ya existentes. A tal efecto elija el comando de menú **Herramientas ▸ Preferencias** y en el cuadro de diálogo ”Preferencias” haga clic en la ficha ”Crear bloque”. En el apartado 5.5 encontrará una descripción detallada de las opciones disponibles.

Adaptar el compilador

También puede adaptar el proceso de compilación a sus necesidades. En el apartado 5.5 encontrará una descripción detallada de las opciones.

Para ello, seleccione el comando de menú **Herramientas ▸ Preferencias**, y en el cuadro de diálogo ”Preferencias” haga clic sobre la ficha ”Compilador”.

Adaptar el editor

Puede ajustar el ancho del tabulador, el guardado antes de compilar, la visualización de números de línea y otras opciones. Para ello, seleccione el comando de menú **Herramientas ▸ Preferencias**, y en el cuadro de diálogo ”Preferencias” haga clic sobre la ficha ”Editor”.

4.3 Trabajo con el editor SCL

Resumen

En principio, la fuente SCL está formada por texto continuo. Para introducir el texto contará con la ayuda de funciones de tratamiento de texto del editor SCL, especialmente adaptado al lenguaje SCL.

Ventana del editor

El objeto fuente para su programa de usuario puede introducirlo en la ventana de trabajo mediante el teclado. Tiene la posibilidad de abrir varias ventanas del mismo objeto fuente o de otro objeto fuente. Mediante el menú 'Ventana' puede organizar las ventanas.

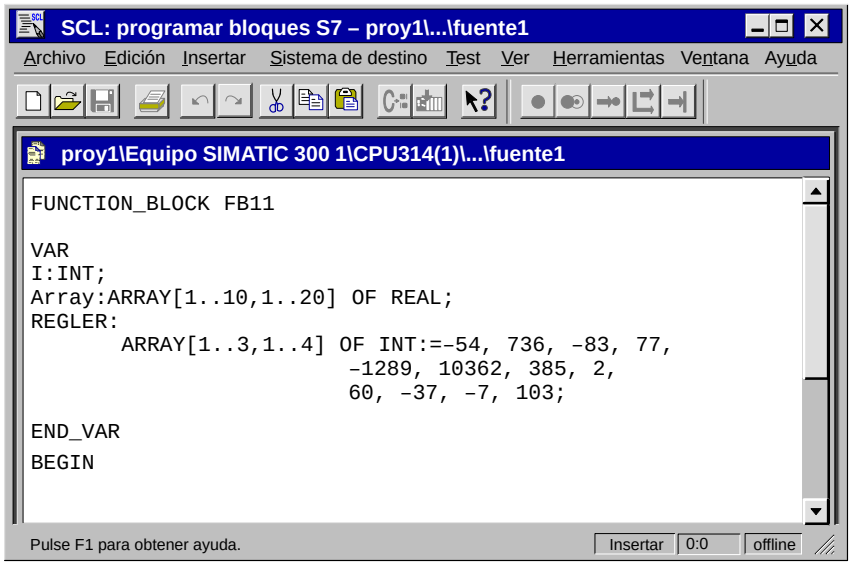


Figura 4-3 *Ventana del editor SCL*

Seleccionar texto

En SCL puede seleccionar texto posicionando el cursor al principio del área que va a seleccionarse y manteniendo pulsado el botón izquierdo del ratón mientras desplaza el cursor a lo largo del área deseada.

Además, puede:

- seleccionar el texto completo de una fuente eligiendo el comando de menú **Edición ▸ Seleccionar todo**,
- seleccionar una sola palabra, haciendo doble clic en la misma,
- seleccionar una línea, haciendo tres clics en la misma.

Buscar y reemplazar

Con el comando de menú **Edición ▸ Buscar/reemplazar** se abre un cuadro de diálogo con el que puede buscar una cadena de caracteres y reemplazarla por otros objetos de texto.

Insertar plantillas

La inserción de plantillas le permite una programación eficaz y le facilita el cumplimiento de la sintaxis. En SCL puede:

- Insertar plantillas para bloques, seleccionando el comando de menú **Insertar ▸ Plantilla de bloques**.
- Insertar plantillas para estructuras de control, seleccionando el comando de menú **Insertar ▸ Estructura de control**.

Cortar, copiar, pegar, borrar

Puede cortar, copiar, pegar y borrar objetos de texto de la forma habitual. Los comandos de menú correspondientes se encuentran en el menú **Edición**.

Por regla general los objetos también pueden desplazarse o copiarse mediante la operación de ”Arrastrar y soltar” (Drag&Drop).

Ir a... (instrucción GOTO)

Con el comando de menú **Edición ▸ Ir a...** se abre un cuadro de diálogo a través del cual podrá posicionar el punto de inserción al inicio de la línea que desee, introduciendo el número de línea y confirmando con ’Aceptar’.

Deshacer, restablecer

Con el comando de menú **Edición ▸ Deshacer** puede deshacer una acción; por ejemplo, deshacer el borrado de una línea.
El comando de menú **Edición ▸ Restablecer** le permite restablecer una acción deshecha anteriormente.

Programar con SCL

5

Resumen

Para programar es necesario realizar una serie de pasos. El presente capítulo describe dichos pasos de trabajo y le presenta una posible secuencia de los mismos.

Indice del capítulo

5.1 Crear programas de usuario en SCL

Requisitos para la creación de programas

Antes de crear un programa con SCL debe realizar las siguientes tareas:

1. Instale un proyecto con el Administrador SIMATIC.
2. Con el Administrador SIMATIC asigne a cada CPU la dirección de comunicación en la red.
3. Configure y parametrice la unidad central de procesamiento y los módulos de señales.
4. En caso de que desee utilizar direcciones simbólicas para áreas de memoria de la CPU o para identificaciones de bloque, debe crear una tabla de símbolos.

Creación de la tabla de símbolos

Si desea utilizar en su programa SCL direcciones simbólicas para áreas de memoria de la CPU o nombres de bloques, debe crear una tabla de símbolos. En el proceso de compilación SCL acudirá a dicha tabla. La tabla de símbolos se crea con STEP 7; igualmente, los valores se introducen en la tabla de símbolos con STEP 7.

Puede abrir la tabla de símbolos con el Administrador SIMATIC o directamente con SCL a través del comando de menú **Herramientas ▸ Tabla de símbolos**.

Además es posible importar y procesar las tablas de símbolos de que se dispone en forma de archivos de texto, y que pueden crearse con cualquier editor de textos (más información al respecto se encuentra en **/231/**).

¿Cómo proceder?

Para crear con SCL un programa de usuario lo primero que debe hacer es crear una fuente SCL. En ella puede editar uno o varios bloques (OB, FB, FC, DB y UDT), y a continuación compilarlos en un proceso por lotes. Al compilar la fuente, los bloques que contiene se depositan en el contenedor ”Programa de usuario” (<AP-off>, v. fig. 5-1) del mismo programa S7 en el que está guardada la fuente.

Puede proceder a crear y editar la fuente SCL con el editor integrado o con un editor estándar. Las fuentes que haya generado con un editor estándar debe importarlas al proyecto con el Administrador SIMATIC. Después puede abrirlas para procesarlas o compilarlas.

5.2 Crear y abrir una fuente SCL

Resumen

Las fuentes que haya creado con SCL puede integrarlas en la estructura de un programa S7 de la siguiente forma:

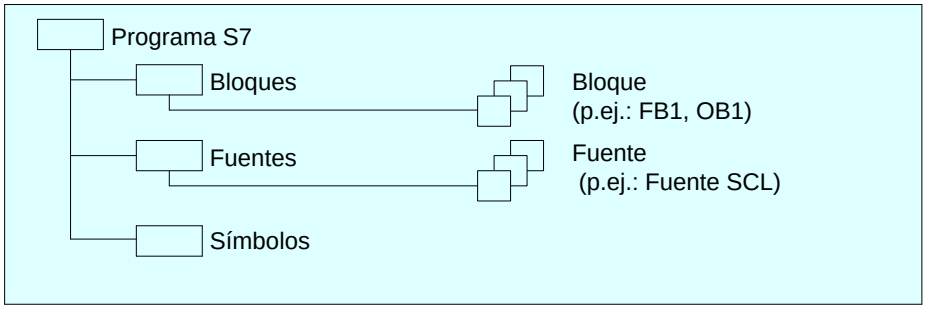


Figura 5-1 *Estructura de un programa S7 en el Administrador SIMATIC*

Creación de la fuente SCL

Para crear una fuente para SCL proceda de la siguiente forma:

1. Seleccione el comando de menú **Archivo ▸ Nuevo** o haga clic sobre el botón ”Nuevo” de la barra de herramientas.
2. En el cuadro de diálogo ”Nuevo” seleccione el objeto deseado y en el programa S7 correspondiente seleccione el contenedor fuente (SO).
3. Abra el contenedor de fuentes y seleccione el comando de menú **Insertar ▸ Software S7 ▸ Fuente SCL**.
4. Seleccione la fuente y elija el comando de menú **Edición ▸ Propiedades del objeto**. Introduzca el nombre del objeto fuente en la ficha ”General”. El nombre puede tener como máximo 24 caracteres. En el nombre de las fuentes es relevante la escritura mayúscula o minúscula.
5. Haga doble clic sobre la fuente. Se abrirá una ventana vacía en la que puede editar la fuente SCL.

Abrir una fuente SCL

Puede abrir un objeto fuente que haya generado con SCL y que haya guardado, para procesarlo o compilarlo.

Proceda de la siguiente forma:

1. Seleccione el comando de menú **Archivo ▸ Abrir**, o haga clic sobre el botón ”Abrir”.
2. En el cuadro de diálogo ”Abrir” seleccione el objeto deseado y el programa S7 correspondiente.
3. Asegúrese que esté seleccionado el filtro ”fuente SCL” y seleccione el contenedor fuente (SO).
4. En el cuadro de diálogo se visualizarán todas las fuentes SCL del programa S7. Seleccione el objeto deseado y confirme con ”Aceptar”, o haga doble clic sobre la fuente.

Las fuentes que haya generado con un editor estándar puede abrirlas de igual forma después de importarlas al proyecto con el Administrador SIMATIC.

5.3 Introducción de declaraciones, instrucciones y comentarios

Resumen

Una fuente SCL debe introducirla ateniéndose a reglas sintácticas predefinidas. Estas reglas son parte de la *descripción del lenguaje* y se relacionan íntegramente en el anexo.

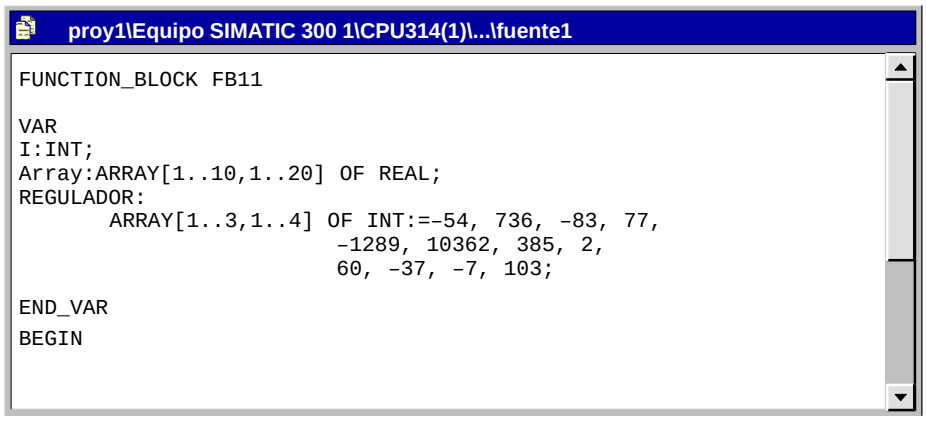


Figura 5-2 *Fuente SCL*

Reglas

Para introducir datos respete las siguientes convenciones:

- En una fuente SCL puede formularse un número cualquiera de bloques lógicos (FB, FC, OB), bloques de datos (DB) y tipos de datos de usuario (UDT), teniendo cada tipo de bloque una estructura típica (v. cap. 8).
- La escritura mayúscula y minúscula sólo es relevante en los identificadores simbólicos (p.ej.: en nombres de variables y literales de caracteres).
- Los bloques llamados anteceden a los bloques invocantes.
- Los tipos de datos de usuario (UDT) anteceden a los bloques en los que se usan aquellos.
- Los bloques de datos globales anteceden a los bloques que acceden a ellos.
- Siga las normas de formato y sintaxis que se indican en la parte de este Manual titulada *Descripción del lenguaje*.

5.4 Guardar e imprimir una fuente SCL

Guardar una fuente SCL

En SCL se entiende por ”Guardar” guardar archivos fuente. En SCL, los bloques se generan al compilar el archivo fuente y se depositan automáticamente en el directorio correspondiente del programa.

Dispone de varias posibilidades para guardar una fuente SCL:

- Seleccione el comando de menú **Archivo ▸ Guardar** o haga clic sobre el botón ”Guardar” de la barra de herramientas.

  La fuente SCL se actualiza.

- Si desea crear una copia de la fuente SCL actual, seleccione el comando de menú **Archivo ▸ Guardar como**. Aparece el cuadro de diálogo ”Guardar como”, en el que puede indicar el nombre y la vía en que debe guardarse la copia.

- Si ejecuta el comando de menú **Archivo ▸ Cerrar** y aún no ha guardado una fuente SCL que haya modificado, se le preguntará si desea guardar las modificaciones, despreciarlas o cancelar el comando **Cerrar**.

  En lugar del comando de menú **Archivo ▸ Cerrar**, también puede hacer clic sobre el botón ”Cerrar” de la línea de título.

  Si desea abandonar SCL mediante el comando de menú **Archivo ▸ Salir** y las fuentes que se hallan abiertas no se han guardado en su estado actual, para cada fuente aparecerá un diálogo con el que puede guardar o desechar las modificaciones efectuadas.

Imprimir un objeto fuente

En todo momento puede sacar en la impresora una copia impresa de los bloques, declaraciones e instrucciones que se encuentren en su fuente SCL. Para ello es necesario que haya instalado y configurado la impresora mediante el control del sistema Windows. Proceda de la siguiente forma:

- Haga clic sobre el botón ”Imprimir” de la barra de herramientas o seleccione el comando de menú **Archivo ▸ Imprimir**. Aparece un cuadro de diálogo en el que puede seleccionar diferentes opciones de impresión (como p.ej.: el área de impresión o el número de copias).

  Confirme con ”Aceptar” para enviar el documento a la impresora.

Definir el diseño de página

Con el comando de menú **Archivo ▸ Preparar página** puede definir el formato de página de su copia impresa.

Crear líneas de encabezado y pies de página

En el Administrador SIMATIC puede ajustar las líneas de encabezado y los pies de página de los documentos que van a imprimirse utilizando el comando de menú **Archivo ▸ Encabezado/Pie de página**.

Presentación preliminar

Con el comando de menú **Archivo ▸ Presentación preliminar** puede comprobar, visualizando la página, los ajustes efectuados, antes de enviar el documento a la impresora. Aquí no es posible editar.

5.5 Proceso de compilación

Resumen

Antes de poder arrancar o realizar una prueba de su programa debe compilarlo. Al iniciar el proceso de compilación (ver más abajo) se activa el compilador, que tiene las características siguientes:

- El compilador trabaja por lotes; es decir, procesa una fuente SCL como una unidad. No son posibles compilaciones parciales (p.ej.: por líneas).
- El compilador comprueba la sintaxis de una fuente SCL y a continuación muestra todos los errores que ha encontrado durante la compilación.
- Genera bloques con información sobre el test si la fuente SCL está libre de errores y la opción correspondiente (ver más abajo) está marcada. Debe seleccionar la opción 'TestInfo' en todos los programas que desee someter a un test con lenguaje de alto nivel SCL.
- Cada vez que se llama a un bloque de función el compilador genera un bloque de datos de instancia, siempre que no exista previamente.

Ajuste del compilador

Usted tiene la posibilidad de adaptar el proceso de compilación a sus necesidades individuales. Para ello, seleccione el comando de menú **Herramientas** ▸ **Preferencias**, y en el cuadro de diálogo "Preferencias" haga clic sobre la ficha "Compilador". Las opciones puede activarlas o desactivarlas haciendo clic con el ratón.

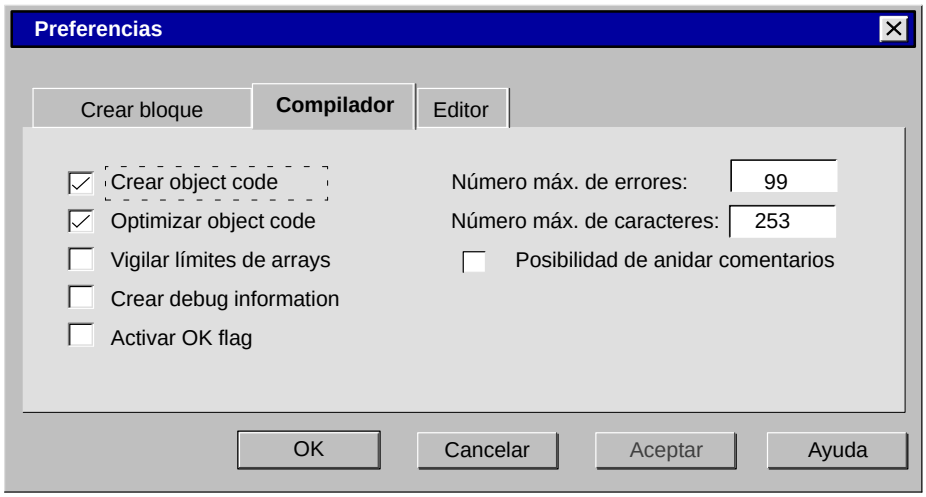


Figura 5-3 *Cuadro de diálogo "Preferencias", ficha "Compilador"*

Opciones

Las opciones tienen los siguientes significados:

- **Número máx. de errores:** El compilador interrumpe la compilación de una fuente SCL cuando se alcanza el número indicado.
- **Crear object code**: Generar (Sí/No) código ejecutable en un PLC.
- **Optimizar object code:** Generar código abreviado. Cuando está seleccionada la opción ”TestInfo” no son posibles todas las optimizaciones.
- **Vigilar límites de arrays:** Para comprobar el tiempo de ejecución y si los índices del array se encuentran dentro del rango permitido (según la declaración del array). Cuando un índice del array sobrepasa el rango permitido, la marca OK se coloca en posición FALSE (con la condición de que esté activada la opción ”Activar OK flag”).
- **Crear debug information**: Generar (Sí/No) información sobre el test, la cual es necesaria para efectuar un test con el depurador de lenguaje de alto nivel.
- **Activar OK flag**: El tiempo de ejecución todas las operaciones erróneas deben colocar la variable OK en posición FALSE.
- **Número máx. de caracteres:** Reducir la longitud estándar del tipo de datos ”ARRAY”. En la configuración básica la longitud estándar es de 254 caracteres.
- **Posibilidad de anidar comentarios:** En la fuente SCL pueden anidarse varios comentarios unos dentro de otros (Sí/No).

Crear bloque

En la ficha ”Crear bloque” dispone de otras posibilidades de influir en el proceso de compilación:

- Puede ajustar si los bloques ya existentes deben sobreescribirse o no durante la compilación.
- Puede hacer que al compilar una fuente se generen automáticamente datos de referencia. No obstante, cuando esta opción está seleccionada el proceso de compilación se prolonga.
- Haga clic sobre la opción ”Considerar atributo de sistema ’S7_server’” cuando el bloque sea relevante para configurar mensajes o enlaces. Esta opción también prolonga el proceso de compilación.

Iniciar el proceso de compilación

Tiene dos posibilidades de iniciar el proceso de compilación:

- Seleccionar el comando de menú **Archivo ▸ Compilar**, o
- hacer clic sobre el botón ”Compilar” de la barra de herramientas.

Para cerciorarse de que siempre está compilando la versión más reciente de su fuente SCL es aconsejable seleccionar el comando de menú **Herramientas ▸ Preferencias** y, antes de compilar, hacer clic sobre la opción Guardar en la ficha ”Editor”. El comando de menú **Archivo ▸ Compilar** guarda la fuente SCL implícitamente.

Después de la compilación

Después del proceso de compilación se le avisará que la compilación se ha realizado sin errores, o se le indicarán los errores con alarmas en una ventana, como indica la figura 5-4.

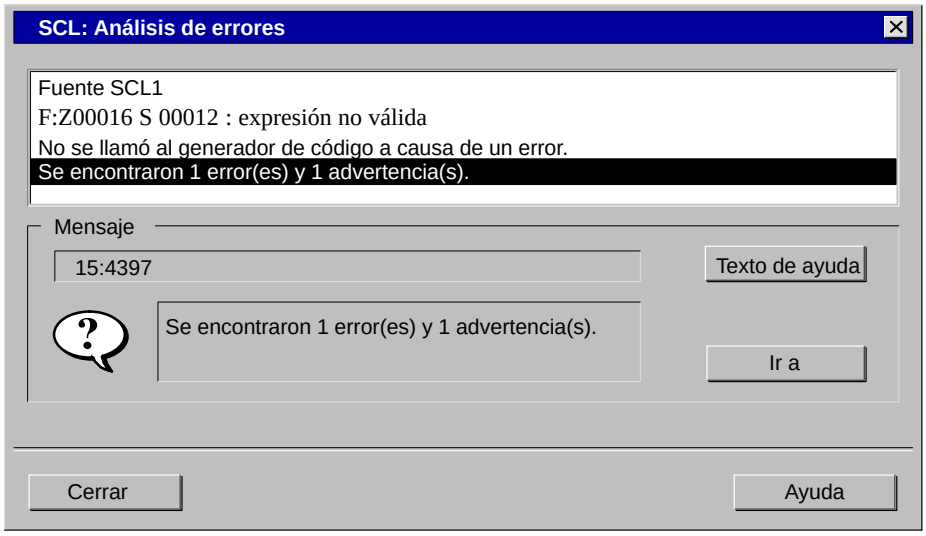


Figura 5-4 *Ventana de mensajes de error y alarmas*

Encontrar causas de errores y alarmas

Cada mensaje indica la línea y columna a que se refiere y una breve descripción. Puede consultar una descripción detallada del error o de la alarma seleccionando el mensaje deseado y haciendo clic sobre el botón de comando ”Ayuda”.

Haciendo doble clic sobre un mensaje puede posicionar el cursor en el punto correspondiente de la fuente SCL.

Estas dos posibilidades le permiten localizar y corregir con rapidez y sencillez errores y alarmas.

5.6 Transferencia del programa de usuario creado al PLC

Resumen

Al compilar una fuente SCL se generan los bloques a partir de la fuente, y se guardan en el contenedor ”Bloques” del programa S7. Posteriormente, en SCL sólo se podrán cargar estos bloques en la CPU a partir de la unidad de programación.

Si desea transferir otros bloques del programa S7 al autómata programable, utilice el Administrador SIMATIC.

Requisitos

Para cargar el programa de usuario en el PLC deben cumplirse los siguientes requisitos:

- Entre la unidad de programación y el autómata programable debe existir conexión.
- Los bloques que van a cargarse deben estar compilados sin errores.

Borrado total de la memoria de la CPU

Con la función ”Borrado total” puede borrar online el programa de usuario completo de una CPU. Tenga en cuenta que además se inicializa la CPU, se interrumpen todas las conexiones existentes con la CPU y, siempre que exista insertada una Memory Card, el contenido de la Memory Card se copiará en la memoria interna de carga. Proceda de la siguiente forma:

1. Seleccione el comando de menú **Sistema de destino ▸ Estado operativo** y coloque la CPU en STOP.
2. Seleccione el comando de menú **Sistema de destino ▸ Borrado total**.
3. Confirme la acción en el cuadro de diálogo que aparece a continuación.

Carga en el sistema de destino

Es ventajoso cargar los bloques en estado operativo STOP, puesto que en caso de sobreescribirse un programa antiguo en estado operativo RUN pueden surgir errores. Proceda de la siguiente forma:

1. Seleccione el comando de menú **Sistema de destino ▸ Cargar.**
2. Si el bloque ya existe en la RAM de la CPU, cuando se solicite confirmación confirme si debe sobreescribirse o no el bloque.

5.7 Creación de un archivo de control de compilación

Resumen

Puede automatizar la compilación de varias fuentes SCL creando un archivo de control de compilación.

Archivo de control de compilación

Puede crear un archivo de control de compilación para su proyecto STEP 7. En él introducirá los nombres de las fuentes SCL que se encuentran en el proyecto. Estas fuentes SCL deben compilarse en un proceso por lotes.

Creación

Para crear el archivo, ejecute los siguientes pasos:

- Al generar un archivo con ”Nuevo” o en ”Abrir” debe crear el filtro para ”Archivo de control de compilación”.
- El archivo con el que trabaja en este momento tiene como identificador específico la extensión ”.inp”.
- Cuando compile este archivo se compilarán por orden los archivos indicados por su nombre.

Compilación

Al compilar, los bloques generados se depositan en el contenedor ”Bloques” del programa S7.

Test de un programa

6

Resumen

Las funciones de test de SCL ofrecen la posibilidad de controlar un programa durante su ejecución en el PLC para encontrar posibles errores.

Los errores de sintaxis son detectados e indicados por SCL en el proceso de compilación. Los errores de tiempo de ejecución en la ejecución del programa se indican mediante alarmas del sistema; los errores lógicos de programación puede detectarlos con las funciones de test de SCL.

¿Dónde puede encontrar más información?

En la ayuda online puede encontrar datos detallados sobre el test con SCL. La ayuda online le proporciona respuestas a preguntas concretas mientras trabaja con SCL.

Indice del capítulo

6.1 Resumen

Lenguaje de alto nivel

Con las funciones de test de SCL puede efectuar un test a los programas de usuario programados en SCL (lenguaje de alto nivel). Este tipo de test le permite:

- Descubrir errores de programación.
- Observar y controlar las repercusiones de un programa de usuario en la CPU al ejecutarse.

Requisitos

Antes de poder hacer un test a un programa SCL debe ejecutar con éxito los siguientes pasos:

1. Debe compilar sin errores el programa con las opciones de compilación ”Crear object code” y ”Crear debug information”. Los ajustes puede seleccionarlos en la ficha ”Compilador” del cuadro de diálogo **Herramientas ▸ Preferencias**.
2. Debe garantizar que entre la PG o el PC y la CPU existe comunicación online.
3. Además, debe cargar el programa en la CPU. Puede hacerlo con el comando de menú **Sistema de destino ▸ Cargar**.

Las funciones de test de SCL

La tabla 6-1 muestra los nombres y una caracterización abreviada de las funciones de test esenciales que pueden llamarse en SCL.

Tabla 6-1 Resumen de las funciones de test

Función	Caracterización
Observación continua (CPU SIMATIC S7-300/400)	Salida del nombre y los valores actuales de variables de un área de observación
Activar puntos de parada (sólo CPU SIMATIC S7-400)	Definir, borrar y editar puntos de parada: test paso a paso
Observar/forzar variables	Observar/definir valores actuales de datos globales
Crear datos de referencia	Crear un resumen a través del programa de usuario
Funciones de test del paquete básico STEP 7	Consultar/modificar estado operativo de la CPU

Nota

Un test efectuado con la instalación en funcionamiento puede provocar graves daños materiales y personales en caso de que existan averías de funcionamiento o errores del programa. Antes de ejecutar las funciones de test asegúrese de que no pueden producirse estados peligrosos.

6.2 Función de test ”Observación continua”

Resumen

Al observar continuamente un programa puede efectuar un test sobre un grupo de instrucciones. Este grupo de instrucciones se denomina también ”área de observación”.

Durante la ejecución del test se visualizan los valores de las variables y los parámetros de dicha área en orden cronológico se actualizan de manera cíclica. Si el área de observación se encuentra en una sección del programa que se recorre en cada uno de los ciclos, generalmente los valores de las variables no pueden registrarse a partir de ciclos consecutivos.

Los valores que se han modificado en el ciclo de ejecución actual se visualizan en negro; los valores que no se han modificado se representan en gris claro.

El alcance de las instrucciones que van a someterse al test está limitado por la capacidad de la(s) CPU(s) conectada(s). Dado que las instrucciones SCL del código fuente se reflejan en el programa de usuario en muchas instrucciones diferentes, la longitud del área de observación es variable. SCL determina y selecciona dicha longitud cuando usted selecciona la primera instrucción del área de observación deseada.

Tipo de test

Al realizar un test en modo ”Observación continua”, los valores actuales de los datos se consultan y visualizan en el área de observación. La consulta se realiza mientras se recorre el área de observación, y la mayoría de las veces provoca una prolongación de los tiempos de ciclo.

Para poder influir sobre esta prolongación, SCL ofrece dos tipos de test diferentes.

- **Tipo de test ”Proceso”:**

  En el tipo de test ”Proceso” el depurador SCL limita el área de observación máxima para que los tiempos de ciclo durante el proceso de test no sobrepasen los tiempos de ejecución reales del proceso o los sobrepasen insignificantemente.

- **Tipo de test ”Laboratorio”:**

  En el tipo de test ”Laboratorio” el área de observación está limitada únicamente por la capacidad de la CPU conectada. Sin embargo, los tiempos de ciclo pueden prolongarse con respecto al proceso real, por lo que el área máxima de observación es mayor que en el tipo de test ”Proceso”.

¿Cómo utilizar "Observación continua"?

Para ejecutar la función "Observación continua" proceda de la siguiente forma:

1. Cerciórese de que se cumplen los requisitos mencionados en el apartado 6.1.
2. Seleccione la ventana que contiene la fuente del programa que va a someterse a test.
3. Si desea cambiar el tipo de test preajustado (Proceso), seleccione el comando de menú **Test ▸ Tipo de test ▸ Laboratorio**.
4. Posicione la marca de inserción en la línea del texto fuente que contiene la primera instrucción del área que va a someterse a test.
5. Seleccione el comando de menú **Test ▸ Observación continua.**

   Resultado: Se determina el área de observación más grande posible, que se visualiza en el borde izquierdo de la ventana mediante una barra gris. La ventana se divide, y en la mitad derecha de la ventana se visualizan los nombres y los valores actuales de las variables que se encuentran en el área observada.
6. Seleccione el comando de menú **Test ▸ Observación continua** para interrumpir el proceso de test y continuarlo posteriormente.
7. Seleccione el comando de menú **Test ▸ Finalizar test** para concluir el proceso de test.

6.3 Función de test ”Activar puntos de parada”

Resumen

Al efectuar el test con la función ”Activar puntos de parada” el proceso de test se realiza paso a paso. Puede ejecutar el programa instrucción por instrucción y observar cómo cambian los valores de las variables editadas.

Después de definir los puntos de parada, primero puede ejecutar el programa hasta el primer punto de parada y desde ahí comenzar la observación paso a paso.

Puntos de parada

Puede definir puntos de parada en cualquier punto del área de instrucciones del texto fuente.

Los puntos de parada no se transmiten al PLC ni se activan hasta que no elija el comando de menú **Test ▸ Activar puntos de parada**. Entonces el programa se ejecuta hasta que se alcanza el primer punto de parada.

El número de puntos de parada activos depende de la CPU:

- CPU 416: máximo 4 puntos de parada activos.
- CPU 414: máximo 2 puntos de parada activos.
- CPU 314: sin posibilidad de puntos de parada activos.

Funciones de paso único

Cuando se ha arrancado la función de test **Activar puntos de parada** puede ejecutar las siguientes funciones:

- **Ejecutar instrucción siguiente**

  Avanza una instrucción, sirve para obtener los valores de las variables.

- **Continuar**

  Continuar hasta el siguiente punto de parada activo.

- **Ejecutar hasta punto de inserción**

  Continuar hasta un punto de inserción en la fuente que haya definido.

Nota

Asegúrese que no se exceda el número máximo de puntos de parada activos al utilizar los comandos de menú **Ejecutar instrucción siguiente** o **Ejecutar hasta punto de inserción**, puesto que dichos comandos definen y activan implícitamente un punto de parada.

¿Cómo usar "Activar puntos de parada"?

Antes de iniciar el test cerciórese de que la CPU se encuentra en estado operativo RUN o RUN-P. Entonces podrá efectuar un test de su programa en paso único con la función "Activar puntos de parada". La siguiente descripción y la representación de la figura 6-1 muestra el procedimiento que debe seguir:

1. Seleccione la ventana que contiene la fuente del bloque que va a someterse al test.
2. Defina puntos de parada posicionando el cursor en el punto deseado de la fuente del programa y seleccionando el comando de menú **Test ▸ Posicionar punto de parada**. Los puntos de parada se representan en el borde izquierdo de la ventana en forma de círculo rojo.
3. Arranque el proceso de paso único seleccionando el comando de menú **Test ▸ Activar puntos de parada**.

   Resultado: La ventana se divide verticalmente en dos mitades y se busca el siguiente punto de parada. Cuando se llega al mismo, la CPU pasa a estado operativo ALTO y el punto de parada alcanzado se marca con una flecha amarilla.
4. Como alternativa se dispone de las siguientes funciones de paso único:
   - Seleccione el comando de menú **Test ▸ Ejecutar instrucción siguiente** (4a). **Resultado:** La CPU pasa brevemente a estado operativo RUN. Cuando se llega a la siguiente instrucción se detiene de nuevo, y en la línea correcta de la mitad izquierda de la ventana muestra los valores de las variables que se han editado en la instrucción precedente.
   - Seleccione el comando de menú **Test ▸ Continuar** (4b). **Resultado:** La CPU pasa a estado operativo RUN. Cuando se llega al siguiente punto de parada activo se detiene de nuevo y marca el punto de parada en el borde izquierdo de la ventana. Para visualizar los contenidos de las variables debe seleccionar otra vez el comando de menú **Test ▸ Ejecutar instrucción siguiente**.
   - Seleccione el comando de menú **Test ▸ Ejecutar hasta selección** (4c). **Resultado:** En la posición actual del punto de inserción se define implícitamente y se activa un punto de parada. La CPU pasa a estado operativo RUN. Cuando se llega al punto de inserción se detiene de nuevo y se selecciona dicho punto de parada. Para visualizar los contenidos de las variables debe seleccionar de nuevo el comando de menú **Test ▸ Ejecutar instrucción siguiente**.
5. Retorne al punto 2. si desea continuar el test con los puntos de parada cambiados. En el punto 2. puede definir nuevos puntos de parada o borrar los puntos de parada existentes.
6. Seleccione el comando de menú **Test ▸ Activar puntos de parada** para salir del bucle de test.
7. Si no desea efectuar el test de ninguna instrucción más de la fuente, concluya el test seleccionando el comando de menú **Test ▸ Finalizar test**.

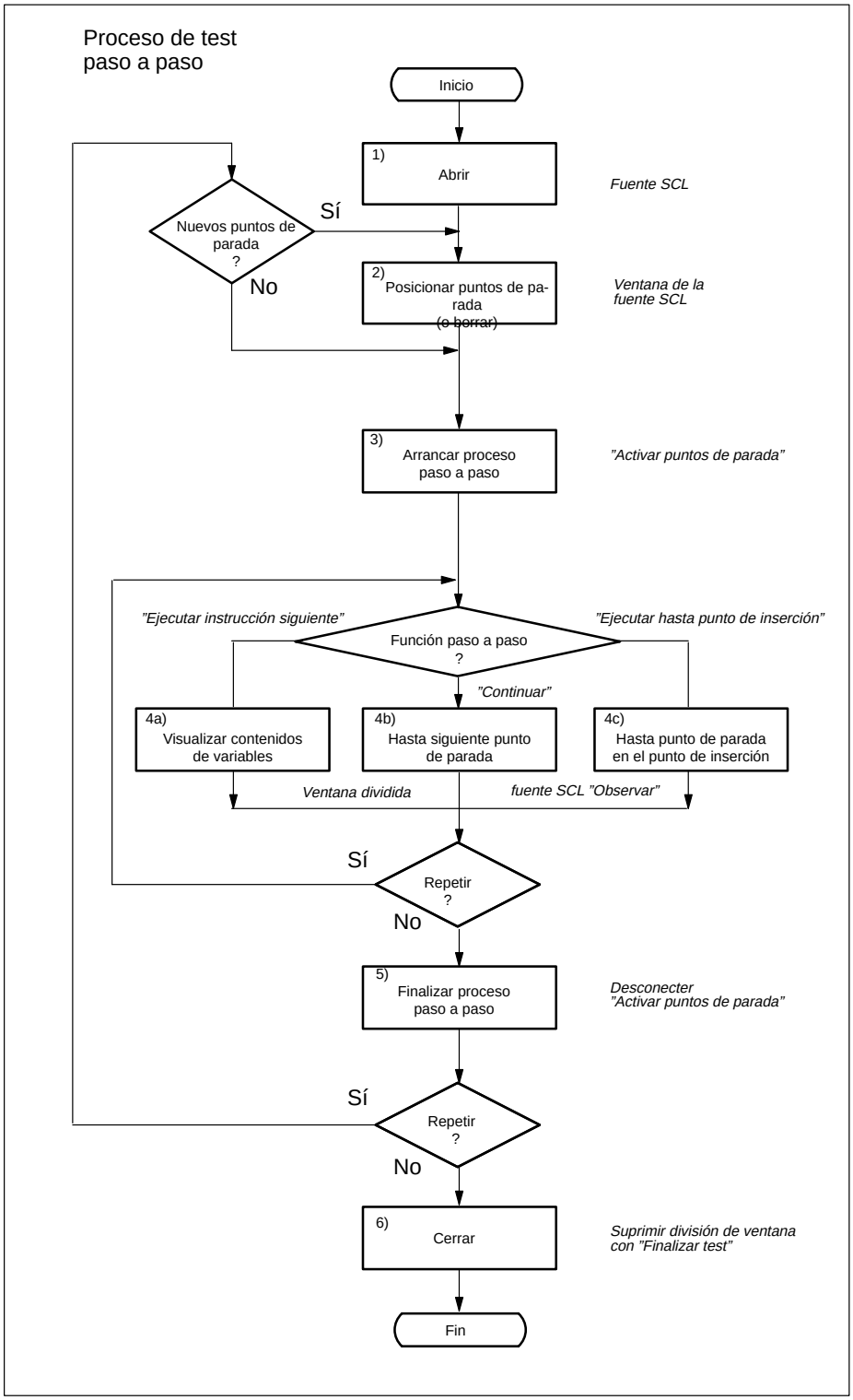


Figura 6-1 *Algoritmo del proceso de test*

6.4 Función de test ”Observar/forzar variables”

Resumen

Al efectuar un test con la función ”Observar/forzar variables” puede:

- visualizar (observar) los valores actuales de datos globales procedentes de su programa de usuario
- asignar valores fijos (forzar) a las variables de un programa de usuario.

Observar y forzar variables

Con el comando de menú **Sistema de destino ▸ Observar/forzar variables** puede:

- definir puntos y condiciones de disparo
- indicar valores para las variables de un programa de usuario.

En ambos casos debe crear una tabla de variables en la que especificará las variables que deben editarse. En el caso de ”Forzar” debe indicar además los valores deseados.

En el manual del usuario /**231**/ de STEP 7 encontrará una descripción detallada de la función de test.

6.5 Función de test ”Datos de referencia”

Resumen

Para facilitar la realización de tests y modificar su programa de usuario, puede generar y evaluar datos de referencia.

Los datos de referencia abarcan: estructura del programa, lista de referencias cruzadas, plano de ocupación, lista de operandos no utilizados, lista de operandos sin símbolo.

Los datos de referencia puede utilizarlos como:

- resumen de un programa de usuario completo,
- base para efectuar modificaciones y tests o
- complemento a la documentación del programa.

Generar datos de referencia

Para generar datos de referencia dispone de las siguientes posibilidades:

- Con el comando de menú **Herramientas ▸ Datos de referencia** puede generar, actualizar y visualizar los datos de referencia cuando lo necesite.
- Con el comando de menú **Herramientas ▸ Preferencias** puede determinar que los datos de referencia se generen automáticamente al compilar una fuente. Para ello, en la ficha ”Crear bloque” seleccione la entrada ”crear datos de referencia”. No obstante, la generación automática de datos de referencia prolonga el proceso de compilación.

En el manual del usuario /**231**/ de STEP 7 encontrará una descripción detallada de la función de test.

6.6 Utilización de las funciones de test STEP 7

Editor AWL

Se pueden abrir bloques en AWL que han sido compilados con SCL y probarlos en el editor AWL.

Consultar y modificar estado operativo de la CPU

Seleccione el comando de menú **Sistema de destino ▸ Estado operativo** para consultar o modificar el estado operativo actual de la CPU.

Visualizar características de la CPU

El comando de menú **Sistema de destino ▸ Información del módulo** abre un cuadro de diálogo en el que:

- puede determinar la causa del estado operativo STOP leyendo el búfer de diagnóstico
- puede consultar el contenido de las pilas de la CPU (la pila de interrupción es una ayuda importante para la búsqueda de errores)
- puede informarse sobre los datos técnicos de la CPU
- puede visualizar la fecha y la hora de la CPU
- puede determinar el tiempo de ciclo de la CPU
- puede informarse sobre los bloques contenidos en la CPU
- puede consultar informaciones sobre la comunicación de la CPU.

En todas estas funciones la CPU debe estar conectada online.

Parte 3:
Descripción del lenguaje

Conceptos básicos generales de SCL

7

Resumen

En este capítulo encontrará los medios de que dispone el lenguaje SCL y cómo puede manejarlos. Tenga en cuenta que aquí sólo se avanzarán conceptos fundamentales y las definiciones necesarias, sobre las que se profundizará en los próximos capítulos.

Indice del capítulo

7.1 Ayudas para la descripción del lenguaje

Descripción del lenguaje SCL

Para describir el lenguaje en cada uno de los capítulos se utilizan diagramas sintácticos. Dichos diagramas le darán una buena visión de la estructura sintáctica (es decir, gramatical) de SCL. En el anexo B encontrará una recompilación completa de todos los diagramas con los elementos de lenguaje utilizados.

¿Qué es un diagrama sintáctico?

El diagrama sintáctico es una representación gráfica de la estructura del lenguaje. La estructura se describe mediante una secuencia de reglas. Algunas reglas pueden basarse en reglas introducidas anteriormente.

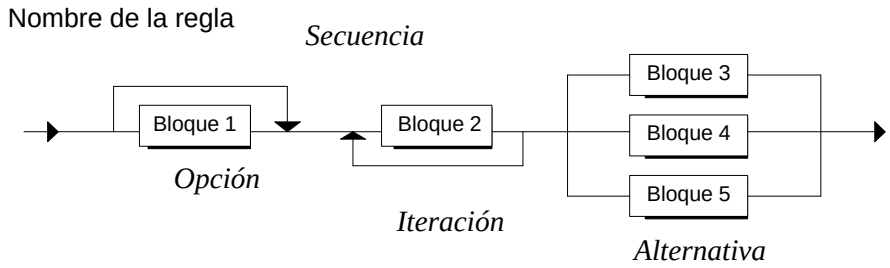


Figura 7-1 *Diagrama sintáctico*

El diagrama sintáctico se lee de izquierda a derecha. Deben tenerse en cuenta las siguientes estructuras de las reglas:

- Secuencia: Secuencia de bloques
- Opción : Rama que puede saltarse
- Iteración: Repetición de ramas
- Alternativa: Ramificación

¿Qué tipos de bloques hay?

Un bloque es un elemento fundamental o un elemento que a su vez puede estar compuesto por bloques. La figura siguiente indica los tipos de símbolo que corresponden a los bloques:

Elemento básico, que no precisa mayor explicación.

En este caso se trata de caracteres imprimibles y caracteres especiales, palabras clave e identificadores predefinidos.

Los datos referidos a estos bloques deben aceptarse sin cambios.

Elemento compuesto, descrito por otros diagramas sintácticos.

¿Qué significa libertad de formato?

Al introducir textos fuente deben respetarse tanto **reglas sintácticas** como **reglas léxicas**.

Ambos grupos de reglas están representados en los anexos B y C. Libertad de formato significa que entre los bloques de reglas puede insertar caracteres de formato, como blancos, tabuladores y saltos de página así como comentarios.

Reglas léxicas

En las reglas léxicas (por ejemplo según la figura 7-2), **no** tiene libertad de formato. Cuando utilice una regla léxica debe aceptar las indicaciones sin cambios.

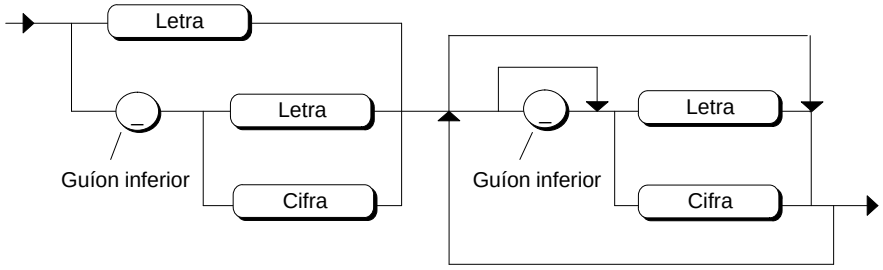


Figura 7-2 Ejemplo de una regla **léxica**

Algunos ejemplos válidos de la regla expuesta serían:

```
R_REGULADOR3
_A_ARRAY
_100_3_3_10
```

Por las razones que se han explicado más arriba, los siguientes serían ejemplos inválidos:

```
1_1AB
RR__20
*#AB
```

Reglas sintácticas

En las reglas sintácticas (p.ej. fig. 7-3) dispone de libertad de formato.

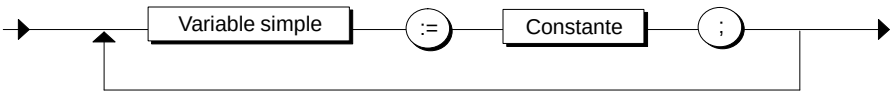


Figura 7-3 Ejemplo de una regla sintáctica

Algunos ejemplos válidos de las reglas expuestas:

```
VARIABLE_1  := 100;     INTERRUPTOR:=FALSE;
VARIABLE_2  := 3.2;
```

7.2 El juego de caracteres de SCL

Letras, cifras

Del juego de caracteres ASCII, SCL utiliza los siguientes:

- las letras (minúsculas y mayúsculas), de la A a la Z;
- las cifras arábigas del 0 al 9;
- blanco (valor ASCII 32) y todos los caracteres de control (ASCII 0-31), incluido el carácter de fin de línea (ASCII 13).

Otros caracteres

Los siguientes caracteres tienen un significado fijo en SCL:

```
+    -    *    /    =    <    >    [    ]    (    )
.    ,    :    ;    $    #    ”    ’    {    }
```

Información adicional

En el anexo A encontrará un listado detallado de todos los caracteres que pueden utilizarse, con la correspondiente interpretación en SCL.

7.3 Palabras reservadas

Significado

Las palabras reservadas son palabras clave que sólo se pueden utilizar tal como están predefinidas. No existe diferencia entre mayúscula y minúscula.

Palabras clave

AND
ANY
ARRAY
BEGIN
BLOCK_DB
BLOCK_FB
BLOCK_FC
BLOCK_SDB
BLOCK_SFB
BLOCK_SFC
BOOL
BY
BYTE

CASE
CHAR
CONST
CONTINUE
COUNTER
DATA_BLOCK
DATE
DATE_AND_TIME
DINT
DIV
DO
DT
DWORD
ELSE
ELSIF
END_CASE
END_CONST
END_DATA_BLOCK
END_FOR
END_FUNCTION
END_STRUCT
END_VAR
END_WHILE
EXIT
FOR
FUNCTION
FUNCTION_BLOCK
GOTO
IF
INT
LABEL
MOD
NIL
NOT
OF
OR
ORGANIZATION_BLOCK
POINTER
REAL
REPEAT
RETURN
S5TIME
STRING
STRUCT
THEN
TIME
TIMER
TIME_OF_DAY
TO
TOD
TYPE
VAR
VAR_TEMP
UNTIL

Palabras clave, continuación

END_FUNCTION_BLOCK
END_IF
END_LABEL
END_TYPE
END_ORGANIZATION_BLOCK
END_REPEAT
VOID
VAR_INPUT
VAR_IN_OUT
VAR_OUTPUT
WHILE
WORD
XOR

Otros nombres reservados

EN
ENO
OK
TRUE
FALSE
Nombres de las funciones estándar

7.4 Identificadores en SCL

Definición

Un identificador es un nombre que usted puede adjudicar a un objeto del lenguaje SCL, es decir, a una constante, a una variable, a una función o a un bloque.

Reglas

Los identificadores pueden estar compuestos por letras y cifras en un orden cualquiera; el primer carácter debe ser una letra o un guión inferior. Están permitidas las letras mayúsculas y minúsculas. Aquí tampoco se diferencia entre mayúsculas y minúsculas. (Por ejemplo, Anna y AnNa son idénticos).

Un identificador puede representarse formalmente por el siguiente diagrama sintáctico:

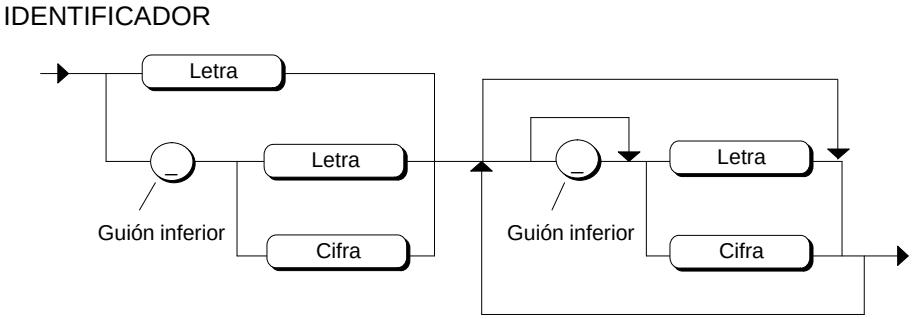


Figura 7-4 *Sintaxis: IDENTIFICADOR*

Tenga en cuenta las siguientes indicaciones:

- Al adjudicar nombres es mejor que seleccione nombres unívocos y expresivos que contribuyan a la inteligibilidad del programa.
- Debe considerar si el nombre ya está ocupado por palabras clave (p.ej. según la tabla 7-1) o identificadores estándar.
- La máxima longitud de un identificador es 24 caracteres.
- Los nombres simbólicos para bloques (es decir otros identificadores que los de la tabla 7-1) debe definirlos en las tablas de símbolos STEP 7 (encontrará información al respecto en **/231/**)

Ejemplos

Los siguientes nombres son identificadores válidos:

```
x          y12         Suma        Temperatura
Nombre     Superficie  Regulador   Tabla
```

Los nombres siguientes **no** son identificadores válidos por las razones indicadas:

`4ter`	El primer carácter debe ser una letra o un guión inferior.
`Array`	ARRAY es una palabra clave y, por lo tanto, no está permitida.
`Valor S`	Los blancos son caracteres y, por lo tanto, no están permitidos.

7.5 Identificadores estándar

Definición

SCL define una serie de identificadores, por lo que reciben el nombre de *identificadores estándar.* Estos identificadores estándar son:

- las palabras claves de bloques y
- los identificadores de operandos para referirse a las áreas de memoria de la CPU.

Palabras clave de bloque

Estos identificadores estándar se utilizan para el direccionamiento absoluto de bloques.

La tabla 7-1 está clasificada según la nemotécnica SIMATIC; también se indica la correspondiente nemotécnica internacional IEC.

Tabla 7-1 Palabras clave de bloque

Nemotécnica (SIMATIC)	Nemotécnica (IEC)	Caracteriza
DBx	DBx	Bloque de datos (Data Block)
FBx	FBx	Bloque de función (Function Block)
FCx	FCx	Función (Function)
OBx	OBx	Bloque de organización (Organization Block)
SDBx	SDBx	Bloque de datos del sistema (System Data Block)
SFCx	SFCx	Función del sistema (System Function)
SFBx	SFBx	Bloque de función del sistema (System Function Block)
Tx	Tx	Temporizador (Timer)
UDTx	UDTx	Tipo de datos de usuario (User defined Data Type)
Zx	Cx	Contador (Counter)
x = Número comprendido entre 0 y 65533 DB0 = ¡Está reservado!		

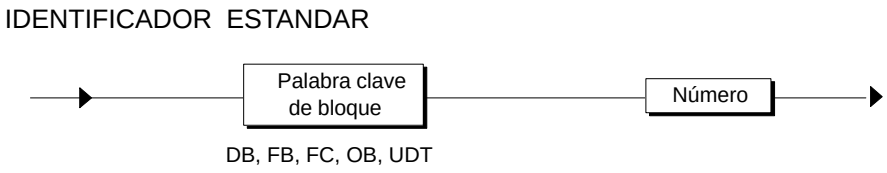


Figura 7-5 *Sintaxis: IDENTIFICADOR ESTANDAR*

Son identificaciones válidas las siguientes:

```
FB10
DB100
T141
```

Identificadores de operandos

Puede referirse a las áreas de memoria de una CPU desde cualquier punto del programa con su identificador de operando.

La tabla siguiente está ordenada según la nemotécnica SIMATIC; también se indica la nemotécnica internacional IEC correspondiente.

Nemotécnica (SIMATIC)	Nemotécnica (IEC)	... direcciona	Tipo de datos
Ax.y	Qx.y	Salida (mediante imagen del proceso)	Bit
ABx	QBx	Salida (mediante imagen del proceso)	Byte
ADx	QDx	Salida (mediante imagen del proceso)	Doble palabra
AWx	QWx	Salida (mediante imagen del proceso)	Palabra
AXx.y	QXx.y	Salida (mediante imagen del proceso)	Bit
Dx.y 1)	Dx.y 1)	Bloque de datos	Bit
DBx 1)	DBx 1)	Bloque de datos	Byte
DDx 1)	DDx 1)	Bloque de datos	Doble palabra
DWx 1)	DWx 1)	Bloque de datos	Palabra
DXx.y 1)	DXx.y1)	Bloque de datos	Bit
Ex.y	Ix.y	Entrada (mediante imagen del proceso)	Bit
EBx	IBx	Entrada (mediante imagen del proceso)	Byte
EDx	IDx	Entrada (mediante imagen del proceso)	Doble palabra
EWx	IWx	Entrada (mediante imagen del proceso)	Palabra
EXx.y	IXx.y	Entrada (mediante imagen del proceso)	Bit
Mx.y	Mx.y	Marcas	Bit
MBx	MBx	Marcas	Byte
MDx	MDx	Marcas	Doble palabra
MWx	MWx	Marcas	Palabra
MXx.y	MXx.y	Marcas	Bit
PABx	PQBx	Salida (periferia directa)	Byte
PADx	PQDx	Salida (periferia directa)	Doble palabra
PAWx	PQWx	Salida (periferia directa)	Palabra
PEBx	PIBx	Entrada (periferia directa)	Byte
PEDx	PIDx	Entrada (periferia directa)	Doble palabra
PEWx	PIWx	Entrada (periferia directa)	Palabra
PEWx	PIWx	Entrada (periferia directa)	Palabra
PEWx	PIWx	Entrada (periferia directa)	Palabra
x = número comprendido entre 0 y 65535 (dirección absoluta del byte) y = número comprendido entre 0 y 7 (número del bit)			

Son ejemplos válidos los siguientes:

```
E1.0        MW10        PAW5        DB20.DW3
```

1) *Estos identificadores de operando sólo valen junto con la indicación de bloque de datos.*

7.6 Números

Resumen

En SCL los números pueden escribirse de diferentes formas. Opcionalmente un número puede incluir un signo: por ejemplo, coma, coma decimal y exponente. Las afirmaciones siguientes son válidas para todos los números.

- Dentro de un número no pueden existir comas ni blancos.
- Para separar ópticamente está permitido el guión inferior (_).
- Al número puede anteponerse un signo más (+) o un signo menos (-). Si al número no se antepone ningún signo se considera positivo.
- Los números no pueden superar por exceso ni por defecto determinados valores máximos y mínimos.

Números enteros

Un número entero no contiene ni coma decimal, ni exponente. Por lo tanto un número entero es sólo una secuencia de cifras que puede estar precedida de un signo. En SCL hay dos tipos de enteros, cada uno de los cuales tiene diferentes rangos: `INT` y `DINT` (v. cap. 9).

Algunos números enteros válidos son:

```
0          1          +1         -1
743        -5280      600_00     -32_211
```

Los siguientes números enteros son **erróneos** por las razones indicadas:

`123,456`	La coma no está permitida.
`36.`	En un número entero no puede haber ningún punto decimal.
`10 20 30`	No están permitidos los blancos.

Números enteros como número binario, de base 8 o hexadecimal

En SCL puede representar los números enteros en otro sistema de numeración. Para hacerlo, se antepone una palabra clave correspondiente al sistema de numeración. 2# representa el sistema binario; 8# representa el sistema en base 8; 16# representa el sistema hexadecimal.

Algunos números enteros válidos para el decimal 15 son:

```
2#1111     8#17       16#F
```

Números reales

Un número real debe incluir o un punto decimal o un exponente (o ambos). Un punto decimal debe estar situado entre dos cifras. Por lo tanto, el número real no puede comenzar ni terminar con punto decimal.

Algunos números reales válidos son:

```
0.0        1.0        -0.2       827.602
50000.0    -0.000743  12.3       -315.0066
```

Los siguientes números reales son **erróneos** por las razones que se indican:

`1.`	A ambos lados del punto decimal debe haber una cifra.
`1,000.0`	Las comas no están permitidas.
`.3333`	A ambos lados del punto decimal debe haber una cifra.

Un exponente puede incluirse para determinar la posición del punto decimal. En caso de que no exista punto decimal se supone que se encuentra a la derecha de la cifra. El propio exponente puede ser un número entero positivo o negativo. La base 10 se sustituye por la letra E.

En SCL la magnitud 3×10^{10} puede representarse por los siguientes números reales:

```
3.0E+10   3.0E10        3e+10       3E10
0.3E+11   0.3e11        30.0E+9     30e9
```

Los siguientes números reales son **erróneos**:

`3.E+10`	A ambos lados del punto decimal debe haber una cifra.
`8e2.3`	El exponente debe ser un número entero.
`.333e-3`	A ambos lados del punto decimal debe haber una cifra.
`30 E10`	Los blancos no están permitidos.

Cadena de caracteres

Una cadena de caracteres es una secuencia de caracteres (es decir, letras, cifras y signos especiales) encerrados entre apóstrofos. Pueden utilizarse las letras mayúsculas y minúsculas.

Algunas cadenas de caracteres válidas son:

```
'ROJO'    '76181 Karlsruhe'         '270-32-3456'
'DM19.95' 'La respuesta correcta es:'
```

Los caracteres especiales de formateado, los apóstrofos (') o un signo $ puede introducirlos con el símbolo de escape $.

Texto fuente	Después de la compilación
`'SIGNAL$'ROT$''`	SEÑAL 'ROJO'
`'50.0$$'`	50.0$
`'WERT$P'`	VALOR *Salto de página*
`'REG-$L'`	REG *Salto de línea*
`'REGEL$R`	REGULADOR *Retorno de carro*
`'SCHRITT$T'`	PASO *Tabulador*

Para los caracteres no imprimibles introduzca la representación complementaria en código hexadecimal con el símbolo *$hh*, donde *hh* representa el valor del carácter ASCII expresado en sistema hexadecimal.

Para interrumpir una cadena de caracteres (para comentarios que no deben imprimirse o visualizarse), en SCL dispone de los caracteres $> y $<, que producen la interrupción de un string.

7.7 Tipos de datos

Resumen

Toda declaración de una variable debe indicar el tipo de esa variable. El tipo determina el rango de la variable y define las operaciones que pueden ejecutarse con ella.

Un tipo de dato concreto determina:

- el tipo y el significado de un elemento de datos,
- el rango admisible del elemento de datos,
- la cantidad admisible de operaciones que se pueden ejecutar con un operando de un tipo de datos,
- la forma en la que se escribe el dato de este tipo.

Tipos de datos

Se distingue entre los siguientes tipos de datos:

Tabla 7-2 Resumen de los tipos de datos existentes

Tipos de datos	Significado
Simples	En SCL puede disponer de ellos de serie.
Compuestos	Puede generarlos relacionando lógicamente tipos de datos simples.
De usuario	Puede definirlos especialmente para su aplicación y elegir libremente el nombre.
Tipos de parámetros	Sólo puede utilizarlos para declarar parámetros.

Tipos de datos simples

Los tipos de datos simples definen la estructura de datos que no pueden descomponerse en unidades menores. Cumplen la definición de la norma DIN EN 1131-3.

En **SCL** ha predefinido doce tipos de datos simples:

```
BOOL        CHAR        INT         TIME
BYTE                    DINT        DATE
WORD                    REAL        TIME_OF_DAY
DWORD                               S5TIME
```

Tipos de datos compuestos

Los tipos de datos compuestos definen estructuras de datos que están formados por otros tipos de dato. SCL admite los siguientes tipos de datos compuestos:

```
DATE_AND_TIME
STRING
ARRAY
STRUCT
```

Tipos de datos de usuario

Se trata de tipos de datos globales (UDT) que usted puede crear en SCL para su aplicación. Puede utilizar este tipo de dato en la tabla de declaración de un bloque o de un bloque de datos con su identificación UDT (UDTx, donde x representa un número) o con un nombre simbólico asignado.

Tipos de parámetros

Además de los tipo de datos simples, compuestos y de usuario, puede utilizar tipos de parámetro para definir parámetros. **SCL** le ofrece los siguientes tipos de parámetro:

```
TIMER       BLOCK_FB    POINTER     ANY
COUNTER     BLOCK_FC
            BLOCK_DB
            BLOCK_SDB
```

7.8 Variables

Declaración de variables

Un identificador cuyo valor puede cambiar durante la ejecución del programa se denomina *variable*. Cada variable debe declararse individualmente (es decir, antes de que pueda utilizarse dentro de un bloque lógico o de un bloque de datos. La declaración de variable determina que un identificador es una variable (y no una constante, etc.) y está especificado por la asignación al tipo de dato variable.

Dependiendo de la validez de las variables se distingue entre:

- datos locales,
- datos de usuario globales, y
- variables predefinidas admisibles (áreas de memoria de una CPU).

Datos locales

Los datos locales son datos que se declaran dentro de un bloque lógico (FC, FB, OB) y que sólo tienen validez para ese bloque lógico. En concreto son los siguientes:

Tabla 7-3 Los datos locales de un bloque

Variable	Significado
Variables estáticas	Una variable estática es una variable local cuyo valor se conserva a través de todos los recorridos de bloques (memoria de bloque). Sirve para guardar valores de un **bloque de función)**.
Variables temporales	Localmente las variables temporales se corresponden con un bloque lógico y **no** ocupan área de memoria estática. Su valor sólo se conserva durante un recorrido del bloque. **No** se puede acceder a las variables temporales fuera del bloque en el que se han declarado las variables.
Parámetros del bloque	Los parámetros del bloque son parámetros formales de un bloque de función o de una función. Son variables locales que sirven para transferir los parámetros actuales indicados en la llamada.

Datos de usuario globales

Los datos de usuario globales son datos o áreas de datos que puede utilizar desde cualquier punto del programa. Para hacerlo debe crear bloques de datos (DB).

Cuando usted crea un DB, define su estructura en una declaración de estructura. En lugar de una declaración de estructura también puede utilizarse un tipo de dato de usuario (UDT). El orden en el que usted introduce los componentes de la estructura determina el orden de los datos en el DB.

Areas de memoria de una CPU

Usted puede acceder a las áreas de memoria de una CPU a través de los identificadores de operandos (v. apt. 7.5) directamente desde cualquier punto del programa, sin necesidad de declarar dichas variables.

Además también tiene la posibilidad de recurrir a dichas áreas de datos simbólicamente. En este caso la asignación simbólica se realiza globalmente mediante la tabla de símbolos en STEP 7. Encontrará información al respecto en **/231/**.

7.9 Expresiones

Resumen

Una expresión representa un valor que se calcula durante la compilación o durante la ejecución del programa. Está compuesta por uno o varios operandos enlazados lógicamente por operadores. El orden de valoración de los operadores está predefinido por la prioridad de los mismos y puede controlarse mediante paréntesis.

- Expresiones aritméticas
- Expresiones lógicas
- Expresiones de comparación

Expresión aritmética

Una expresión típica es, por ejemplo, la siguiente:

```
(b*b-4*a*c)/(2*a)
```

Los identificadores a y b y los números 4 y 2 son los operandos; los símbolos *, - y / son los operadores correspondientes (multiplicación, sustracción y división). Toda la expresión representa un número.

Expresión de comparación

Una expresión de comparación es una expresión lógica que puede ser verdadera o falsa. He aquí un ejemplo de expresión de comparación:

```
Valor_teorico < 100.0
```

En esta expresión, `Valor_teorico` es una variable real; `100.0` es un número real, y el símbolo < es un operador de comparación. La expresión tiene el valor ”verdadero” cuando el valor teórico es un valor inferior a 100.0; en caso contrario la expresión tiene el valor **falso**.

Expresión lógica

Una típica expresión lógica es:

```
a AND NOT b
```

Los identificadores `a` y `b` son los operandos; las palabras clave `AND` y `NOT` son operadores lógicos. Toda la expresión representa una plantilla de bits.

7.10 Instrucciones

Resumen

Una instrucción SCL es una acción ejecutable en el área de instrucciones de un bloque lógico. En SCL hay tres instrucciones fundamentales:

1. Asignaciones de valor (asignación de una expresión a una variable)
2. Instrucciones de control (repetición o ramificación de instrucciones).
3. Procesamiento de subrutinas (llamada o ramificación de bloques lógicos).

Asignaciones de valor

Una asignación de valor típica es por ejemplo:

```
VALOR_TEORICO := 0.99*VALOR_TEORICO_ANTIGUO
```

En este ejemplo se presupone que `VALOR_TEORICO` y `VALOR_TEORICO_ANTIGUO` son variables reales. La orden de la instrucción multiplica el valor `VALOR_TEORICO_ANTIGUO` por 0.99 y asigna el producto a la variable `VALOR_TEORICO`. Tenga en cuenta que el símbolo de la asignación es :=.

Instrucciones de control

Una instrucción de control típica es:

```
FOR Contador :=1 TO 20 DO
            LISTA[Contador]  := VALOR+Contador;
END_FOR;
```

En este ejemplo se ejecuta la asignación 20 veces. Cada vez que se ejecuta, en el área `LISTA` el nuevo valor calculado se registra en un puesto inmediatamente superior de la lista.

Procesamiento de subrutinas

Al especificar un identificador de bloque para una función (FC) o un bloque de función (FB) se llama al bloque lógico declarado con dicho identificador. [1)] Cuando la declaración del bloque lógico incluye parámetros formales, al llamar a los parámetros formales pueden asignarse operandos actuales a los parámetros formales.

Todos los parámetros listados en el bloque de declaración

`VAR_INPUT, VAR_OUTPUT` y `VAR_IN_OUT`

de un bloque lógico se denominan como parámetros formales; por el contrario, los parámetros correspondientes en las llamadas que se hallan dentro de la parte de instrucciones se denominan parámetros actuales.

La transferencia de los parámetros actuales a los parámetros formales forma parte de la llamada.

Un procesamiento típico de subrutina es, por ejemplo:

```
FC31(X:=5, Q1:=Suma_horizontal);
```

1) *En las funciones la asignación de parámetros actuales es imprescindible; en los bloques de función es opcional.*

7.11 Bloques SCL

Resumen

En un archivo fuente SCL puede programar como texto fuente desde 1 hasta n bloques.

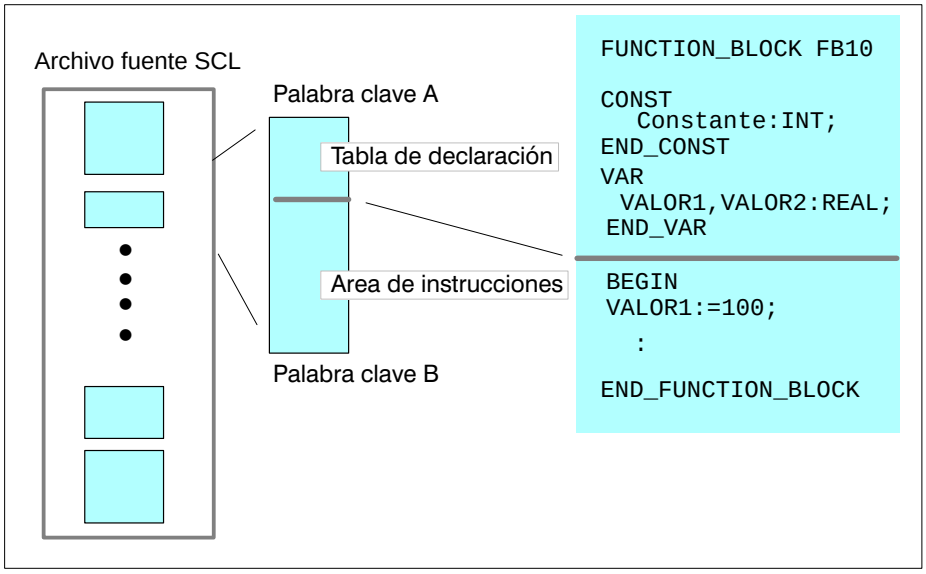
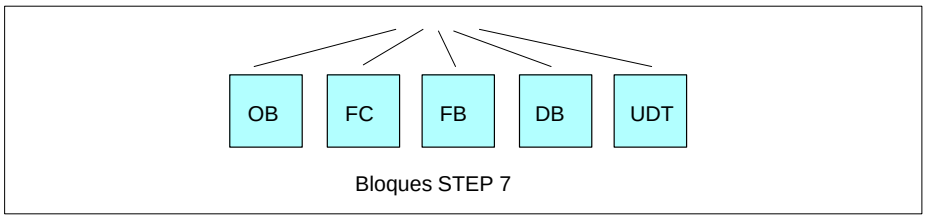


Figura 7-6 *Estructura de un archivo fuente SCL*

Tipos de bloques

Los bloques STEP 7 son partes de un programa de usuario delimitadas por su función, su estructura o su finalidad. Con SCL puede programar los siguientes bloques:



Bloques predefinidos

No necesita programar todas las funciones. También puede recurrir a bloques predefinidos, que se hallan en el sistema operativo de las unidades centrales de procesamiento o en librerías (*S7lib*) del paquete básico STEP7, y pueden utilizarse, por ejemplo, para la programación de funciones de comunicación.

Estructura de un bloque de SCL

Cada bloque consta de las siguientes partes:

- Inicio/Fin de encabezado de bloque (palabra clave correspondiente al tipo de bloque)
- Tabla de declaración
- Area de instrucciones (en los bloques de datos, tabla de asignación).

Tabla de declaración

En la tabla de declaración deben adoptarse todas las definiciones necesarias para formar la base del área de instrucciones: por ejemplo, definición de constantes y declaración de variables y parámetros.

Area de instrucciones

El área de instrucciones puede comenzar – si se desea – con la palabra clave `BEGIN` y finaliza siempre con la palabra clave de fin de bloque: `END_xxx` (v. apt. 8.2).

Todas las instrucciones terminan con punto y coma (";"). Delante de cada instrucción puede colocarse una meta de salto (Label). En el capítulo 13 encontrará la sintaxis de la tabla de cada una de las instrucciones.

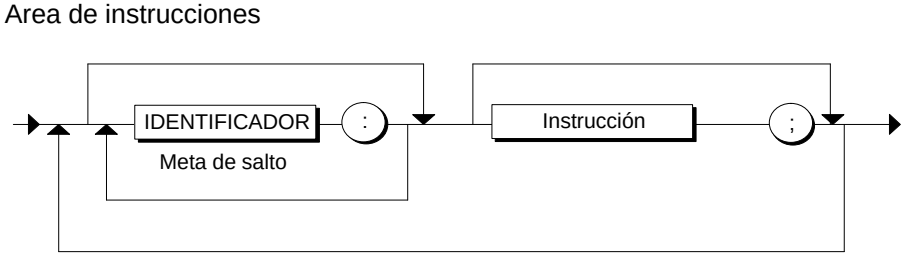


Figura 7-7 *Sintaxis: Area de instrucciones*

He aquí un ejemplo del área de instrucciones de un bloque de función (FB):

```
:                  //Fin de tabla de declaración
:
BEGIN              //Inicio del área de instrucciones
           X := X+1;
LABEL1 :   Y := Y+10;
           Z := X*Y;
           :
           GOTO LABEL1
LABELn :           //Fin del área de instrucciones
END_FUNCTION_BLOCK
```

En el área de instrucciones de un bloque de datos pueden predefinir valores mediante las asignaciones de valor de los datos del DB. Por ello, en los próximos capítulos el área de instrucciones de un DB se denominará **tabla de asignación**.

Programa S7

Después de la compilación los bloques generados se depositan en el contenedor "Bloques" del programa S7 correspondiente. Desde allí debe transferirlos a las CPU. Encontrará información en **/231/**.

7.12 Comentarios

Resumen

Los comentarios sirven de documentación y para comprender mejor un bloque SCL. Carecen de significado para la ejecución del programa después de la compilación. Hay dos tipos de comentarios:

- la línea de comentario, y
- el bloque de comentario.

Línea de comentario

Comentario que se inicia con '//' y que se extiende hasta el final de la línea. La longitud está limitada a un máximo de 253 caracteres, incluido el carácter de inicio '//'. Puede representarse formalmente por el siguiente diagrama sintáctico:

Línea de comentario

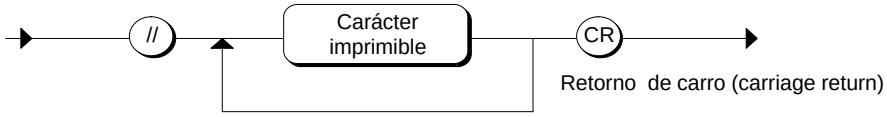


Figura 7-8 *Sintaxis: Línea de comentario*

Los caracteres imprimibles puede consultarlos en la tabla A-2 del anexo. Dentro de la línea de comentario las parejas de signos '(*' y '*)' carecen de significado.

Bloque de comentario

Comentario que puede abarcar varias líneas y que, por ser un bloque, se inicia con (* y concluye con *). Está permitido el anidamiento de bloques de forma estándar. Se puede cambiar este ajuste e impedir el anidamiento de bloques de comentario.

Bloque de comentario

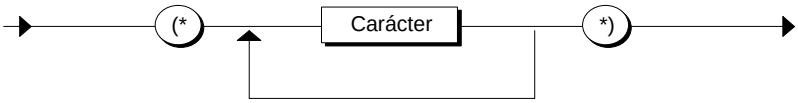


Figura 7-9 *Sintaxis: Bloque de comentario*

Los caracteres permitidos puede consultarlos en la tabla A-2 del anexo.

Reglas a tener en cuenta

Para la notación de comentarios tenga en cuenta que:

- Los bloques de comentario **dentro de bloques de datos** deben notarse de la misma manera que las líneas de comentario, es decir, también se introducen con '//'.
- El anidamiento de comentarios está permitido como ajuste estándar. No obstante, este ajuste del compilador puede cambiarse mediante la opción "Posibilidad de anidar comentarios ". Para ello, seleccione el comando de menú **Herramientas** ▸ **Preferencias**, y dentro de la ficha "Compilador" del siguiente cuadro de diálogo anule la opción seleccionada.
- Un comentario no puede interrumpir ni un nombre simbólico ni una constante. Sin embargo, está permitida la interrupción de cadenas (strings).

El siguiente comentario es **erróneo**:

```
FUNCTION_(* Adaptacion*)BLOCK FB10
```

Ejemplo de introducción de comentarios

En el ejemplo puede ver dos bloques de comentario y una línea de comentario.

```
FUNCTION_BLOCK FB15
(* Aquí hay un bloque de comentario
que puede ocupar varias líneas*)
VAR
   INTERRUPTOR: INT; // Línea de comentario
END_VAR;
BEGIN
  (* Asignar a la variable INTERRUPTOR un valor *)
  INTERRUPTOR:= 3;
END_FUNCTION_BLOCK
```

Figura 7-10 Ejemplo de comentario

Nota

Los comentarios de línea situados directamente detrás de la declaración de variables de un bloque se recuperan al decompilar a un programa AWL.

Estos comentarios los encontrará dentro de AWL en el área de interface, es decir, en la parte superior de la ventana (véase también **/231/**.)

Por lo tanto, en el ejemplo 7-10 se acepta la primera línea de comentario.

Estructura de un archivo fuente SCL

8

Resumen

En principio un archivo fuente SCL está formado por texto continuo. En dicho archivo fuente puede programar varios bloques. Dichos bloques pueden ser OBs, FBs, FCs, DBs o UDTs.

En este capítulo se va a describir la estructura externa de los bloques. Los capítulos siguientes le informarán sobre las estructuras internas.

Indice del capítulo

8.1 Estructura

Resumen

Un archivo fuente SCL está compuesto por el texto fuente, de 1 hasta n bloques (que pueden ser FBs, FCs, OBs, DBs o UDTs).

Para que su archivo fuente SCL pueda compilarse en cada uno de los bloques, debe tener en cuenta determinadas estructuras y normas sintácticas de dichos bloques.

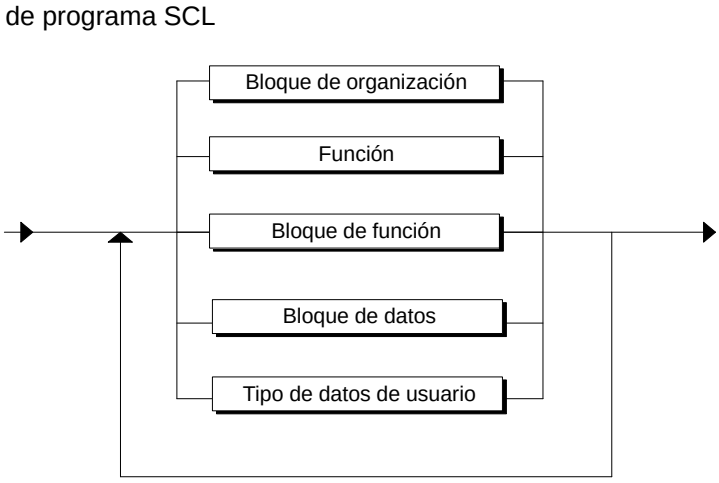


Figura 8-1 *Sintaxis: Unidad de programa SCL*

Secuencia de los bloques

Por cuanto respecta a la secuencia de los bloques, al crear el archivo fuente debe tener en cuenta lo siguiente:

Los bloques llamados preceden a los bloques invocantes.

Es decir:

- Los tipos de dato de usuario (UDTs) preceden a los bloques en los que se utilizan.
- Los bloques de datos que tienen asignado tipo de datos de usuario (UDT) se sitúan después del UDT.
- Los bloques de datos a los que se puede acceder desde todos los bloques lógicos preceden a los bloques desde los que usted efectúa la llamada.
- Los bloques de datos que tienen asignado bloque de función se sitúan después del bloque de función.
- El OB1 que llama a otros bloques va en último lugar. A su vez, los bloques llamados por los bloques llamados desde el OB1 anteceden a los anteriores.

Los bloques a los que usted llama en el archivo fuente pero que no programa en el mismo archivo fuente deben encontrarse en el programa de usuario correspondiente en el momento de compilar el archivo.

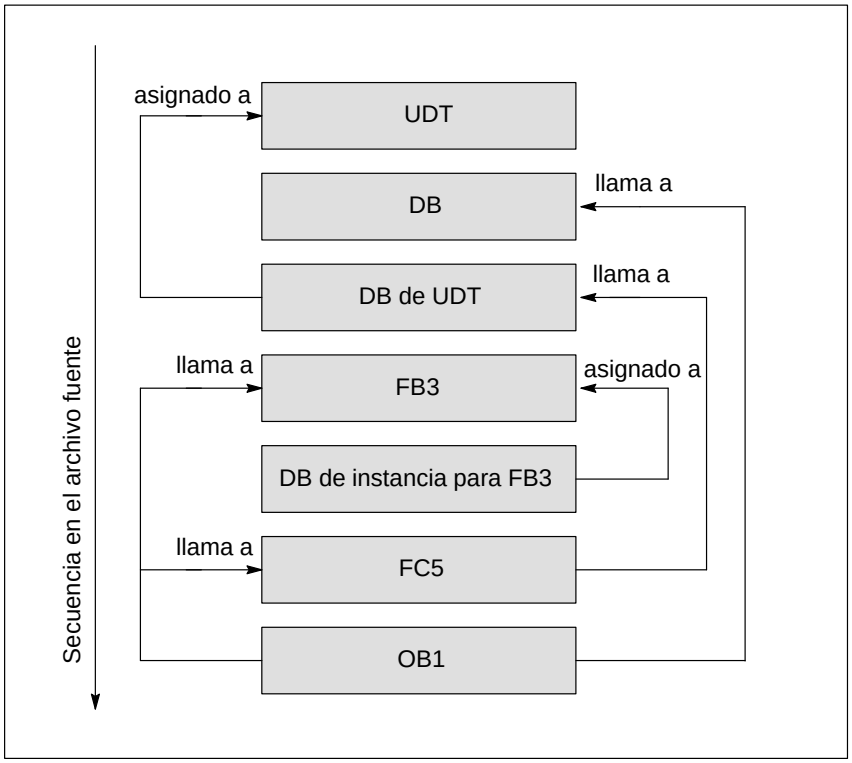


Figura 8-2 Estructura de bloques de un archivo fuente (ejemplo)

Estructura general de bloques

Básicamente el código fuente para un bloque está formado por los siguientes segmentos:

- Inicio de bloque, con indicación (absoluta o simbólica) del bloque
- Atributos del bloque (optativo)
- Tabla de declaración (varía dependiendo del tipo de bloque)
- Área de instrucciones en bloques lógicos o asignación de valores actuales en bloques de datos (optativo)
- Final de bloque

8.2 Inicio y final de bloque

Resumen

Cada uno de los textos fuente de un bloque viene precedido de un identificador estándar de inicio y final del bloque (v. tabla 8-1) que depende del tipo de bloque, así como de un identificador del bloque.

Sintaxis

Tabla 8-1 Identificadores estándar para inicio y final de bloque

Sintaxis	Tipo de bloque	Identificación

`ORGANIZATION_BLOCK` *`ob_name`*
`:`
`END_ORGANIZATION_BLOCK`
	OB	Bloque de organización

`FUNCTION` *`fc_name:function_type`*
`:`
`END_FUNCTION`
	FC	Función

`FUNCTION_BLOCK` *`fb_name`*
`:`
`END_FUNCTION_BLOCK`
	FB	Bloque de función

`DATA_BLOCK` *`db_name`*
`:`
`END_DATA_BLOCK`
	DB	Bloque de datos

`TYPE` *`name udt_name`*
`:`
`END_TYPE`
	UDT	Tipo de datos de usuario

Identificación de bloque

En la tabla 8-1 *xx_name* representa el identificador de bloque según la siguiente sintaxis:

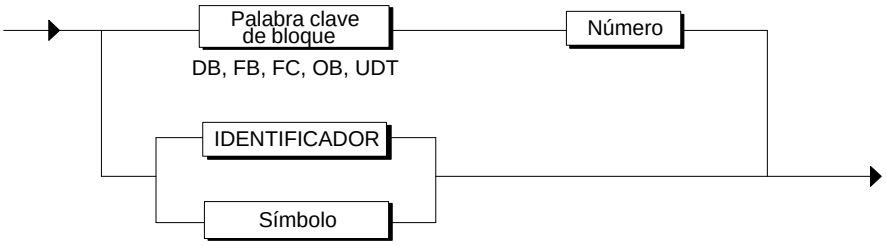


Figura 8-3 *Sintaxis: Identificación de bloque*

En el apartado 7.5 podrá consultar más información. Tenga también en cuenta que debe definir un identificador o un identificador de señal en la tabla de símbolos de STEP 7 (ver /**231**/).

Ejemplo

```
FUNCTION_BLOCK FB10
FUNCTION_BLOCK Bloque_regulador
FUNCTION_BLOCK "Regulador.B1&U2"
```

8.3 Atributos de bloque

Definición

Los atributos para bloques pueden ser:

- Atributos de bloque
- Atributos del sistema para bloques

Atributos de bloque

Puede indicar título, versión, protección de bloque, autor, nombre y familia de un bloque utilizando palabras clave.

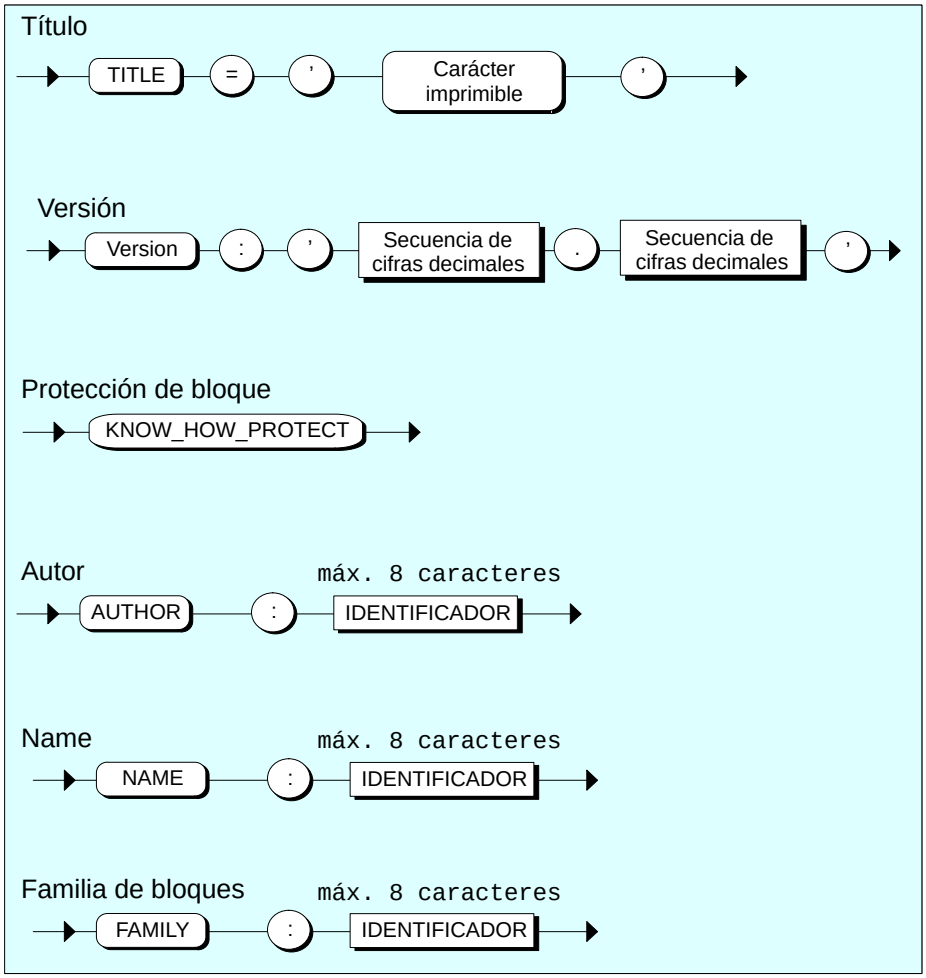


Figura 8-4 *Sintaxis: Atributos de bloque*

Atributos del sistema para bloques

Además puede asignar a los bloques atributos del sistema: p.ej., para configurar la técnica de mando.

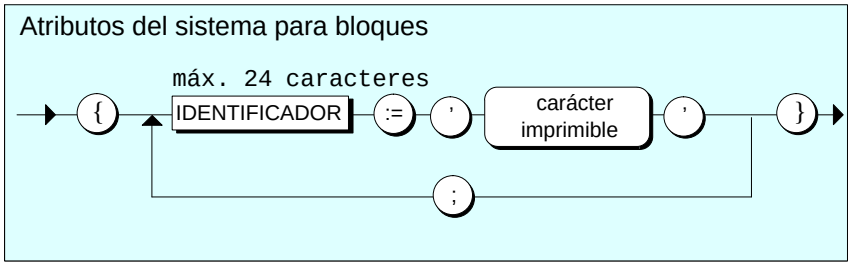


Figura 8-5 *Sintaxis: Atributos del sistema para bloques*

La tabla 8-2 muestra los atributos del sistema que puede asignar a bloques en SCL.

Tabla 8-2 Atributos del sistema para bloques

Atributo	Valor	Se asigna este atributo cuando	Tipo de bloque permitido
S7_m_c	true, false	el bloque debe manejarse o visualizarse desde un equipo de manejo y visualización.	FB
S7_tasklist	taskname1, taskname2, etc.	el bloque debe poder llamarse a otros OBs (p.ej., OBs de tratamiento de errores o de arranque) además de llamarse a bloques de organización cíclicos.	FB, FC
S7_block-view	big, small	el bloque debe poder representarse en formato grande o pequeño en un equipo de manejo y visualización.	FB, FC

Asignación de atributos

La asignación de atributos de bloque se realiza **después** de la identificación del bloque y **antes** de la tabla de declaración.

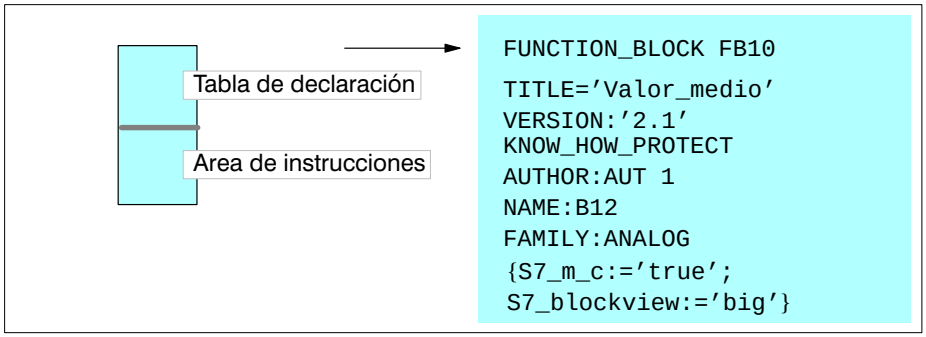


Figura 8-6 Asignación de atributo

8.4 Tabla de declaración

Resumen

La tabla de declaración sirve para definir las variables locales y globales, los parámetros, las constantes y las metas de salto.

- Las variables locales, los parámetros, las constantes y las metas de salto, que sólo deben tener validez dentro de un bloque lógico, puede definirlas en la tabla de declaración del bloque lógico.
- Los datos globales, a los que puede acceder cualquier bloque lógico, los definirá en la tabla de declaración del bloque de datos (DB).
- En la tabla de declaración de un UDT definirá un tipo de datos de usuario.

Estructura

Una tabla de declaración se estructura en diferentes bloques de declaración, cada uno de ellos identificado por su propia pareja de palabras clave. Cada bloque incluye una lista de declaración para los datos del mismo tipo como, por ejemplo, metas de salto, datos estáticos, datos temporales. Un tipo de bloque sólo puede presentarse una vez y no está permitido en todos los tipos de bloque, como muestra la siguiente tabla. Los bloques pueden encontrarse en cualquier orden.

Bloques de declaración

Datos	Sintaxis	FB	FC	OB	DB	UDT
Constantes	CONST *Lista de declaración* END_CONST	X	X	X		
Metas de salto	LABEL *Lista de declaración* END_LABEL	X	X	X		
Variables temporales	VAR_TEMP *Lista de declaración* END_VAR	X	X	X		
Variables estáticas	VAR *Lista de declaración* END_VAR	X	X[2)]		X[1)]	X[1)]
Parámetros de entrada	VAR_INPUT *Lista de declaración* END_VAR	X	X			
Parámetros de salida	VAR_OUTPUT *Lista de declaración* END_VAR	X	X			
Parámetros de entrada/ salida	VAR_IN_OUT *Lista de declaración* END_VAR	X	X			
Lista de declaración: es la lista de los identificadores del tipo que debe declararse						

1) *En los bloques de datos y los UDT las palabras clave VAR y END_VAR se reemplazan por STRUCT y END_STRUCT.*

2) *La declaración de variables dentro de la pareja de palabras clave VAR y END_VAR sí está permitida en funciones, pero durante la compilación las declaraciones se desplazan al área temporal.*

Atributos del sistema para parámetros

A los parámetros de entrada, de salida y de entrada/salida puede asignar también atributos del sistema: p.ej., para configurar mensajes o enlaces.

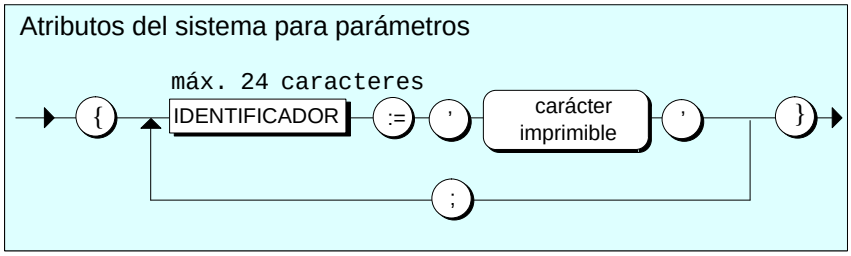


Figura 8-7 *Sintaxis: Atributos del sistema para parámetros*

La tabla 8-3 muestra los atributos del sistema que puede asignar a los parámetros.

Tabla 8-3 Atributos del sistema para parámetros

Atributo	Valor	Se asigna este atributo cuando	Tipo de declaración permitida
S7_server	connection, alarm_archiv	el parámetro es relevante para configurar enlaces o mensajes. Este parámetro incluye el número de enlace o de mensaje.	IN
S7_a_type	alarm, alarm_8, alarm_8p, alarm_s, notify, ar_send	usted define el tipo del bloque de mensaje llamado en el área de instrucciones (requisito: también debe estar asignado el atributo S7_server:=alarm_archiv).	IN, sólo en FB
S7_co	pbkl, pbk, ptpl, obkl, fdl, iso, pbks, obkv	usted define el tipo de enlace que debe configurarse (requisito: también debe estar asignado el atributo S7_server:=connection).	IN
S7_m_c	true, false	el parámetro debe poder manejarse y observarse desde un equipo de manejo y visualización.	IN/OUT/ IN_OUT, sólo en FB
S7_shortcut	2 caracteres cualesquiera; p.ej., W, Y	al parámetro debe asignarse una denominación abreviada para evaluar valores analógicos.	IN/OUT/ IN_OUT, sólo en FB
S7_unit	unidad; p. ej., litro	al parámetro deben asignarse unidades para evaluar valores analógicos.	IN/OUT/ IN_OUT, sólo en FB
S7_string_0	16 caracteres cualesquiera; p. ej., ABIERTO	al parámetro debe asignarse texto para evaluar valores binarios.	IN/OUT/ IN_OUT, sólo en FB, FC
S7_string_1	16 caracteres cualesquiera; p. ej., CERRADO	al parámetro debe asignarse texto para evaluar valores binarios.	IN/OUT/ IN_OUT, sólo en FB, FC
S7_visible	true, false	el parámetro debe poder visualizarse o no en CFC.	IN/OUT/IN_OUT, sólo en FB, FC

Tabla 8-3 Atributos del sistema para parámetros, continuación

Atributo	Valor	Se asigna este atributo cuando	Tipo de declaración permitida
S7_link	true, false	el parámetro debe poder interconectarse o no en CFC.	IN/OUT/IN_OUT, sólo en FB, FC
S7_dyna-mic	true, false	el parámetro debe poder dinamizarse o no en CFC.	IN/OUT/IN_OUT, sólo en FB, FC
S7_param	true, false	el parámetro debe poder parametrizarse o no en CFC.	IN/IN_OUT, sólo en FB, FC

Asignación de atributos

Los atributos del sistema para parámetros debe asignarlos en los bloques de declaración Parámetros de entrada, Parámetros de salida o Parámetros de entrada/salida.

Ejemplo:

```
VAR_INPUT
  in1  {S7_server:='alarm_archiv';
        S7_a_type:='ar_send'}:DWORD;
END_VAR
```

8.5 Area de instrucciones

Resumen

El área de instrucciones contiene **instrucciones**:[1)]

- que se ejecutan después de llamar a un bloque lógico (estas instrucciones sirven para procesar datos y direcciones) o
- para predefinir valores concretos en bloques de datos.

Sintaxis

La figura 8-8 muestra la sintaxis del área de instrucciones. Está compuesta por una repetición de las instrucciones individuales, por lo que antes de cada instrucción puede existir una meta de salto (v. apt. 11.6) que es la meta de una instrucción de salto.

Area de instrucciones

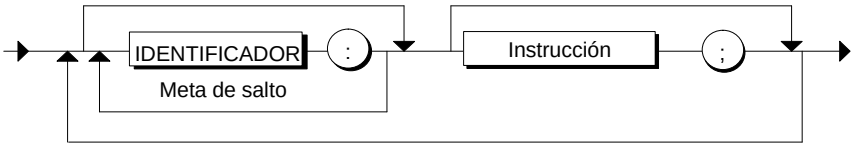


Figura 8-8 *Sintaxis: Area de instrucciones*

Algunas instrucciones válidas son, por ejemplo:

```
BEGIN
            VALOR_INICIAL           :=0;
            VALOR_FINAL             :=200;
:
GUARDAR:    RESULTADO   :=VALOR_TEORICO;
:
```

Notas importantes

Al formular instrucciones debe tener en cuenta que:

- el área de instrucciones comienza con la palabra clave `BEGIN`,
- el área de instrucciones termina con la palabra clave de final de bloque,
- cada instrucción termina con punto y coma, y
- se declaran todos los identificadores utilizados en el área de instrucciones.

1) *En el presente manual utilizamos el término "instrucción" para todos los entes que declaran una función ejecutable.*

8.6 Instrucción

Resumen

Cada una de las instrucciones está compuesta por:

- **Asignaciones de valor**, que sirven para asignar a una variable un valor, el resultado de una expresión o el valor de otra variable.
- **Instrucciones de control**, que sirven para repetir instrucciones o grupos de instrucciones o para ramificarlas dentro de un programa.
- **Procesamiento de subrutinas**, que sirven para llamar a funciones y bloques de función.

Instrucción

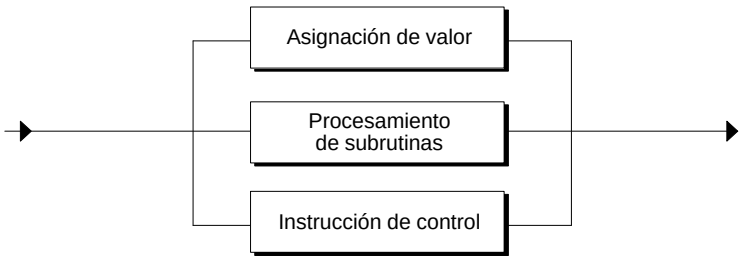


Figura 8-9 *Sintaxis: Instrucción*

Los elementos necesarios para formular estas instrucciones son expresiones, operadores y operandos. Se tratarán en posteriores capítulos.

Ejemplos

Los ejemplos que siguen pretenden ilustrar las diferentes variantes de instrucciones que existen:

```
// Ejemplo de asignación de valor
      VALOR_MEDIDO:= 0 ;

// Ejemplo de procesamiento de subrutina
      FB1.DB11(TRANSFERENCIA:= 10) ;

// Ejemplo de instrucción de control
      WHILE CONTADOR < 10 DO..
      :
      END_WHILE;
```

Ejemplo 8-1 Instrucciones

8.7 Estructura de un bloque de función (FB)

Resumen

Un FB (bloque de función) es un bloque lógico que contiene una sección de un programa y que dispone de un área de memoria asignada. Cada vez que se llama a un FB debe asignársele un DB de instancia (v. cap. 10). La estructura de este DB de instancia puede definirla especificando la tabla de declaración de FB.

Bloque de función

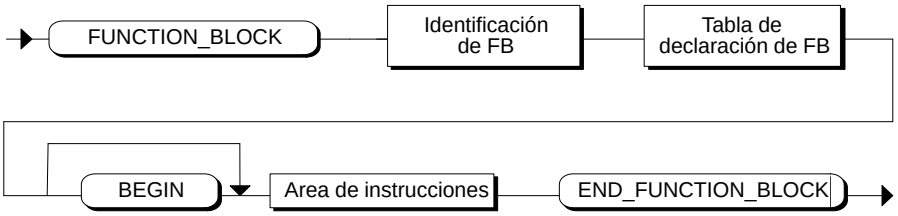


Figura 8-10 *Sntaxis: Bloque de función (FB)*

Identificación de FB

Después de la palabra clave

```
FUNCTION_BLOCK
```

de identificación de FB, introduzca la palabra clave FB y después el número de bloque o el nombre simbólico del FB.

Ejemplos:

```
FUNCTION_BLOCK FB10
```

```
FUNCTION_BLOCK MOTOR_1
```

Area de declaración de FB

La tabla de declaración de FB sirve para declarar los datos específicos del bloque. Puede consultar los bloques de declaración permitidos en el capítulo 8.4. Tenga en cuenta que la tabla de declaración también define la estructura del DB de instancia asignado.

Ejemplos:

```
CONST
   CONSTANTE:=5;
END_CONST


VAR
   VALOR1,VALOR2,VALOR3:INT;
END_VAR
```

Ejemplo

Ejemplo 8-2 muestra el código fuente para un bloque de función. Los parámetros de entrada y de salida (aquí V1, V2) están predefinidos con valores iniciales.

```
FUNCTION_BLOCK FB11
      VAR_INPUT
            V1: INT:= 7;
      END_VAR
      VAR_OUTPUT
            V2: REAL;
      END_VAR
      VAR
            RECORRIDO_1:INT;
      END_VAR
      BEGIN
      IF V1 = 7 THEN
      RECORRIDO_1:= V1;
      V2:= FC2 (VALOR_TEST:= RECORRIDO_1);
      //Llamada a la función FC2 suministrando
      //parámetros mediante la variable
      //estática RECORRIDO_1
      END_IF;
END_FUNCTION_BLOCK
```

Ejemplo 8-2 *Ejemplo de un bloque de función*

8.8 Estructura de una función (FC)

Resumen

Una FC (función) es un bloque lógico al que no se le asigna un área de memoria propia. Por ello una FC no necesita ningún DB de instancia. A diferencia de un FB, una función puede devolver al punto de llamada un resultado de la función (**valor de respuesta**). Por ello, la función puede utilizarse como una variable en una expresión. Las funciones del tipo VOID no tienen valor de respuesta.

Función

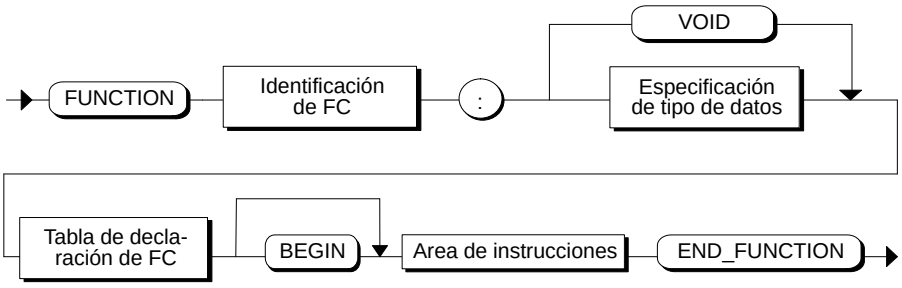


Figura 8-11 *Sintaxis: Función (FC)*

Identificación de FC

Después de la palabra clave

```
FUNCTION
```

de identificación de FC introduzca la palabra clave FC seguida del número de bloque o del nombre simbólico de la FC.

Ejemplos:

```
FUNCTION FC100
FUNCTION NUM_REVOLUCIONES
```

Especificación de tipo de dato

Aquí indique el tipo de dato del valor de respuesta. Están permitidos todos los tipo de dato que se describen en el capítulo 9 a excepción de STRUCT y ARRAY. No es necesario indicar un tipo de dato cuando se prescinde del valor de respuesta con **VOID**.

Tabla de declaración de FC

Los bloques de declaración permitidos puede consultarlos en el apartado 8.4.

Area de instrucciones

Dentro del área de instrucciones debe asignarse el **resultado de la función** al nombre de la función. Una instrucción válida dentro de una función con identificador FC31 es, por ejemplo:

```
FC31:= VALOR;
```

Ejemplo

El ejemplo muestra una función con los parámetros de entrada formales x1, x2, y1, y2, un parámetro formal de salida Q2 y un valor de respuesta FC11.

El significado de los parámetros formales lo encontrará en el capítulo 10.

```
FUNCTION FC11: REAL
      VAR_INPUT
            x1: REAL;
            x2: REAL;
            y1: REAL;
            y2: REAL;
      END_VAR
      VAR_OUTPUT
            Q2: REAL;
      END_VAR
      BEGIN       // Tabla de instrucciones
      FC11:= RAIZ // Valor de respuesta de la
                  // función
      ( (x2 - x1)**2 + (y2 - y1) **2 );
      Q2:= x1;
END_FUNCTION
```

Ejemplo 8-3 *Ejemplo de una función*

8.9 Estructura de un bloque de organización (OB)

Resumen

Un OB (bloque de organización) es, al igual que un FB o una FC, una sección del programa de usuario, que el sistema operativo llama cíclicamente o cuando se producen determinados sucesos. Constituye el interface entre el programa de usuario y el sistema operativo.

Bloque de organización

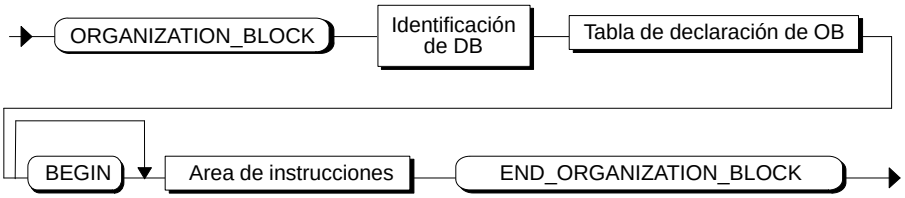


Figura 8-12 *Sintaxis: Bloque de organización (OB)*

Identificación de OB

Después de la palabra clave

```
ORGANIZATION_BLOCK
```

de identificación de OB introduzca la palabra clave OB seguida del número de bloque o, simplemente, del nombre simbólico del OB.

Ejemplo:

```
ORGANIZATION_BLOCK OB14
```

```
ORGANIZATION_BLOCK ALARMA_HORA
```

Tabla de declaración de OB

Para ejecutarse, cada OB necesita básicamente **datos locales de 20 bytes** para su información de arranque. Dependiendo de los requerimientos del programa, puede declarar variables temporales complementarias en el OB. La descripción de los datos locales de 20 bytes puede consultarla en **/235/**.

Ejemplo:

```
ORGANIZATION_BLOCK OB14
//ALARMA_HORA

VAR_TEMP
HEADER:ARRAY [1..20] OF BYTE;//20 byte para BESY
:
:
END_VAR
```

Los restantes bloques de declaración permitidos para los OB puede consultarlos en el apartado 8.4.

8.10 Estructura de un bloque de datos (DB)

Resumen

DB (bloque de datos): el bloque de datos incluye datos de usuario globales a los que acudirán **todos** los bloques del programa. Todos los FB, FC u OB pueden leer o escribir estos bloques de datos. En el capítulo 12 encontrará la estructura de los bloques de datos que sólo están asignados a determinados FB (DB de instancia).

Bloque de datos

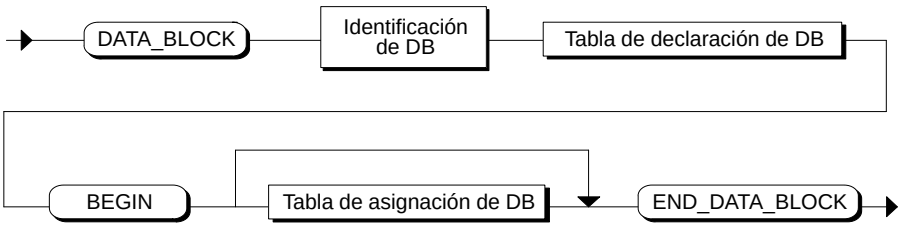


Figura 8-13 *Sintaxis: Bloque de datos (DB)*

Identificación de DB

Después de la palabra clave

`DATA_BLOCK`

de identificación de DB introduzca la palabra clave DB seguida del número de bloque o simplemente del nombre simbólico del DB.

Ejemplo

```
DATA_BLOCK DB20
DATA_BLOCK RANGO_VAL_MED
```

Tabla de declaración de DB

En la tabla de declaración de DB puede definir la estructura de datos del DB. A una variable de DB puede asignarse un tipo de datos estructurado (STRUCT) o un tipo de datos de usuario (UDT).

Tabla de declaración de DB

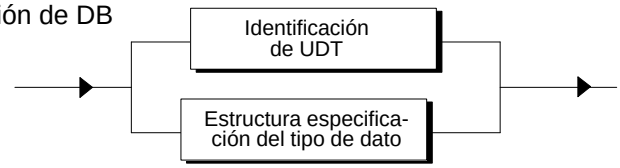


Figura 8-14 *Sintaxis: Tabla de declaración de DB*

Ejemplo:

```
DATA_BLOCK DB20
           STRUCT      // Tabla de declaración
             VALOR:ARRAY [1..100] OF INT;
           END_STRUCT

BEGIN                  // Inicio de tabla de asignación
:
END_DATA_BLOCK         // Final de bloque de datos
```

Tabla de asignación de DB

Usted puede adaptar los datos que ha declarado en la tabla de declaración para su aplicación específica utilizando datos concretos específicos de DB. La tabla de asignación comienza con la palabra clave:

```
BEGIN
```

y está compuesta de una secuencia de asignaciones de valor con la siguiente sintaxis:

Tabla de asignación de DB

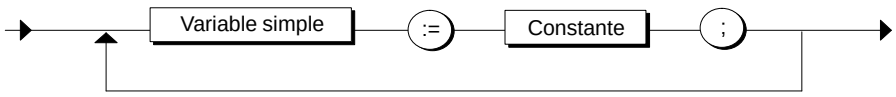


Figura 8-15 *Sintaxis: Tabla de asignación de DB*

Nota

Al adjudicar valores iniciales (inicialización), al indicar atributos y al indicar comentarios dentro de un DB, se aplica la sintaxis de AWL. En el manual de usuario **/231/** o en el manual **/232/** podrá consultar informaciones sobre las formas de escribir constantes, atributos y comentarios.

Ejemplo

El ejemplo siguiente le muestra cómo puede formularse la tabla de asignación, cuando los valores del array [1] y [5] deben tener el valor entero 5 y –1 en lugar del valor predefinido 1.

```
DATA_BLOCK  DB20
STRUCT            //Declaración de datos
                  //predefinidos
      VALOR : ARRAY [ 1..100] OF INT := 100 (1);
      MARCA: BOOL := TRUE;
      S_WORD: WORD   := W#16#FFAA;
      S_BYTE: BYTE   := B#16#FF;
      S_TIME: S5TIME := S5T#1h30m30s;
END_STRUCT

BEGIN             //Tabla de asignación
  //Asignación de valor para determinados
  //elementos del ARRAY
  VALOR [1] := 5;
  VALOR [5] :=–1;
END_DATA_BLOCK
```

Ejemplo 8-4 Tabla de asignación de DB

8.11 Estructura de un tipo de datos de usuario (UDT)

Resumen

Los tipos de datos de usuario (UDT) son estructuras de datos especialmente generadas por usted. Dado que los tipos de datos de usuario tienen un nombre, pueden utilizarse múltiples veces. Por definición pueden utilizarse en la totalidad del programa de la CPU, por lo que son tipos de datos globales. En consecuencia, estos tipos de datos pueden:

- utilizarse en bloques como tipos de datos simples o compuestos, o
- utilizarse como plantilla para la creación de bloques de datos de igual estructura de datos.

Tipo de datos de usuario

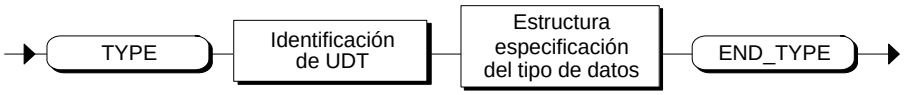


Figura 8-16 *Sintaxis: Tipo de datos de usuario (UDT)*

Identificación de UDT

Después de la palabra clave

`TYPE`

introduzca la palabra clave UDT seguida de un número o del nombre simbólico del UDT

Ejemplos:

```
TYPE UDT10
TYPE BLOQUE_SUMINISTRO
```

Especificación de tipos de datos

La especificación de tipos de datos se realiza siempre con ayuda de una **especificación de tipos de datos STRUCT**. El tipo de datos UDT puede utilizarse en los bloques de declaración de bloques lógicos o en bloques de datos, o asignarse a DBs. En el capítulo 9 puede consultar los bloques de declaración permitidos y otras informaciones.

Tipos de datos

9

Resumen

Un tipo de datos es una reunión de rangos y operaciones para formar una unidad. Como la mayoría de los demás lenguajes de programación, SCL posee tipos de datos predefinidos (es decir, integrados en el lenguaje). Además, el programador puede crear tipos de datos compuestos o de usuario.

Indice del capítulo

9.1 Resumen

Resumen

Dentro de SCL se distinguen los siguientes tipos de datos, como muestra la tabla 9-1.

Tabla 9-1 Los tipos de datos de SCL

Tipos de datos simples			
BOOL	CHAR	INT	TIME
BYTE		DINT	DATE
WORD		REAL	TIME_OF_DAY
DWORD			S5TIME
Tipos de datos compuestos			
DATE_AND_TIME	STRING	ARRAY	STRUCT
Tipos de datos de usuario			
UDT			
Tipos de parámetros			
TIMER	BLOCK_FB	POINTER	ANY
COUNTER	BLOCK_FC		
	BLOCK_DB		
	BLOCK_SDB		

Estos tipos de datos determinan:

- el tipo y el significado de los elementos de datos,
- los rangos permitidos para los elementos de datos,
- la cantidad de operaciones permitidas que se pueden efectuar con un operando de un tipo de datos, y·
- la forma de escribir los datos de este tipo.

9.2 Tipos de datos simples

Resumen

Los tipos de datos simples definen la estructura de datos que no puede descomponerse en unidades menores. Cumplen la definición de la norma DIN EN 1131-3. Un tipo de datos simple describe un área de memoria de longitud fija y representa una magnitud de bits, un entero, un número real, un tiempo, una hora del día o un carácter. Estos tipos de datos están predefinidos en SCL.

Tabla 9-2 Bits de anchura y rangos de los tipos de datos simples

Tipos de datos	Palabra clave	Bits de anchura	Rango
Bits	Los datos de este tipo ocupan 1 bit (tipo de datos BOOL), 8 bits, 16 bits o 32 bits.		
Bit	BOOL	1	`0, 1 o FALSE, TRUE`
Byte	BYTE	8	No puede indicarse un rango de valores numérico. Se trata de combinaciones de bits con las que no pueden formarse expresiones numéricas.
Palabra	WORD	16	
Doble palabra	DWORD	32	
Caracteres	Los datos de este tipo ocupan exactamente 1 carácter del juego de caracteres ASCII		
Carácter individual	CHAR	8	Juego de caracteres ASCII ampliado
Datos numéricos	Se dispone de ellos para procesar valores numéricos		
Entero de 16 bits	INT	16	`-32_768` hasta `32_767`
Entero de 32 bitse	DINT	32	`-2_147_483_648` hasta `2_147_483_647`
Número en coma flotante (IEE)	REAL	32	-3.402822E+38 hasta -1.175495E-38, 0.0, +1.175495E-38 hasta 3.402822E+38
Temporizadores	Los datos de este tipo representan los diferentes valores de temporización y fecha dentro de STEP 7.		
Tiempo S5	S5TIME	16	`T#0H_0M_0S_10MS` hasta `T#2H_46M_30S`
Datos de tiempo: Tiempo IEC en intervalos de 1 ms	TIME (=DURATION)	32	`-T#24D_20H_31M_23S_647MS` hasta `T#24D_20H_31M_23S_647MS`
Fecha: Fecha IEC en intervalos de 1 día	DATE	16	`D#1990-01-01` hasta `D#2168-12-31`
Hora del día: Hora en intervalos de 1 ms	TIME_OF_DAY (=TOD)	32	`TOD#0:0:0` hasta `TOD#23:59:59.999`

Nota respecto al tiempo S5: Dependiendo de la base de tiempo utilizada (0.01S, 0.1S, 1S o 10S), la resolución del valor de temporización está limitada. El compilador redondea adecuadamente los valores.

9.3 Tipos de datos compuestos

Resumen

SCL soporta los siguientes tipos de datos compuestos:

Tabla 9-3 Tipos de datos compuestos

Tipo de dato	Descripción
DATE_AND_TIME DT	Define un área con 64 bits (8 Bytes). Este tipo de datos memoriza (en formato decimal con codificación binaria) la fecha y la hora del día, y está predefinido en SCL.
STRING	Define un área para una secuencia de caracteres de un máximo de 254 caracteres (tipo de datos CHAR).
ARRAY	Define un área de elementos de un tipo de datos (o simples o compuestos).
STRUCT	Define una agrupación de tipos de datos combinados a voluntad. Usted puede definir un ARRAY formado por estructuras o una estructura formada por estructuras y ARRAYS.

9.3.1 Tipo de datos: DATE_AND_TIME

Resumen

Este tipo de datos consta de los dos tipos de datos `DATE` y `TIME` y define un área de 64 bits (8 Bytes) para indicar fecha y hora del día. El área de datos guarda las siguientes informaciones: `año-mes-día-horas:minutos:segundos.milisegundos`.

DATE_AND_TIME

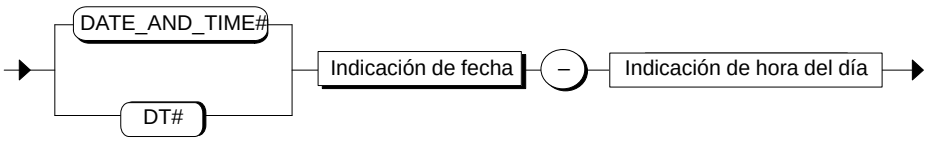


Figura 9-1 *Sintaxis: DATE_AND_TIME*

Rango

Tabla 9-4 Bits de anchura y rango

Tipo	Palabra clave	Bits de anchura	Rango
Fecha y hora del día	DATE_AND_TIME (=DT)	64	`DT#1990-01-01-0:0:0.0` hasta `DT#2089-12-31-23:59:59.999`

En el capítulo 11 de la presente descripción encontrará la sintaxis exacta para indicar la fecha y la hora del día. Una definición válida para las 12 horas, 20 minutos, 30 segundos y 10 milisegundos del día 20.10.1995 es:

```
DATE_AND_TIME#1995-10-20-12:20:30.10
```

```
DT#1995-10-20-12:20:30.10
```

Nota

Se dispone de funciones estándar para acceder dirigidamente a los componentes DATE o TIME.

9.3.2 Tipo de datos STRING

Resumen

El tipo de datos STRING define una cadena de caracteres de una longitud máxima de 254 caracteres individuales.

El área estándar reservada para una cadena de caracteres está compuesta por 256 bytes, que es el espacio necesario para memorizar 254 caracteres y un encabezado de 2 bytes.

Puede reducir el espacio de memoria destinado a una cadena de caracteres definiendo el número máximo de caracteres que pueden guardarse en una cadena de caracteres. Una *Cadena cero* (es decir, una cadena sin contenido) representa el valor más pequeño posible.

Especificación de tipo de datos STRING

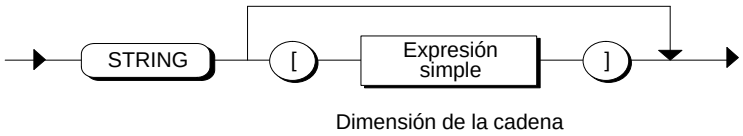


Figura 9-2 *Sintaxis: Especificación de tipo de datos STRING*

La expresión simple (dimensión de la cadena) representa el número máximo de caracteres de la cadena.

Algunos tipos de string válidos son:

```
STRING[10]
STRING[3+4]
STRING[3+4*5]
STRING      Rango máximo (predefinido ≙ 254 caracteres)
```

Rango

En una cadena de caracteres están permitidos todos los caracteres del código ASCII. En el capítulo 11 se describe cómo se manejan los caracteres de control y los caracteres no imprimibles.

Nota

Para aprovechar mejor los recursos de su CPU se puede reducir la longitud estándar de 254 caracteres del tipo de datos STRING a cualquier cantidad de caracteres en valores de retorno de una FC, así como en parámetros de salida y de entrada/salida. Seleccione el comando de menú **Preferencias** del menú **Herramientas**, y en el siguiente cuadro de diálogo seleccione la ficha ”Compilador”. A continuación introduzca el número de caracteres deseado dentro de la opción ”Número máx. de caracteres”.

9.3.3 Tipo de datos ARRAY

Resumen

El tipo de datos ARRAY consta de un número predeterminado de componentes de un único tipo de datos. En el diagrama sintáctico 9-3, correspondiente a los arrays, se especifica con más precisión este tipo de datos después de la palabra reservada OF. SCL distingue:

- El tipo de array unidimensional
  Es una lista de elementos de datos organizada en orden ascendente.
- El tipo de array bidimensional
  Es una tabla de datos compuesta por filas y columnas. La primera dimensión se refiere al número de línea y la segunda al número de columna.
- El siguiente tipo de array de dimensión superior a dos
  Es la ampliación del tipo de array bidimensional añadiéndole otras dimensiones. El número máximo de dimensiones permitido es 6.

Especificación de tipo de datos ARRAY

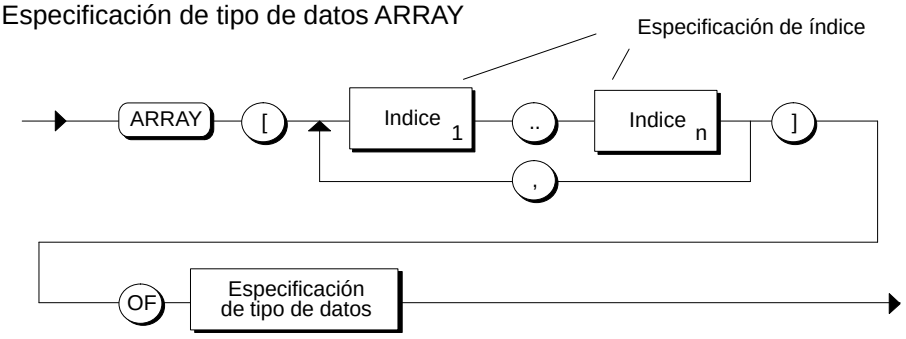


Figura 9-3 *Sintaxis: Especificación de tipo de datos ARRAY*

Especificación de índice

Describe la dimensión del array con:

- El índice mínimo y máximo posible (rango de índice) para cada dimensión. El índice puede ser cualquier valor numérico entero (-32768 hasta 32767).
- Los límites deben indicarse separados por dos puntos.
- Cada uno de los rangos de índice se separan entre sí por comas, y la especificación completa de índices se encierra entre corchetes.

Especificación de tipo de datos

Con la especificación de tipo de datos puede declarar el tipo de datos de los componentes de array. Como tipos de datos están permitidas todas las posibilidades mencionadas en este capítulo. El tipo de datos de un array puede ser también una estructura.

Las siguientes especificaciones son posibles tipos de array:

```
ARRAY[1..10] OF REAL
ARRAY[1..10] OF STRUCT..END_STRUCT
ARRAY[1..100, 1..10] OF REAL
```

9.3.4 Tipo de datos STRUCT

Resumen

Un tipo de datos STRUCT describe un área compuesta por un número fijo de componentes, cuyos tipos de datos respectivos pueden ser diferentes. En el diagrama sintáctico de la fig. 9-4 se indican estos elementos de datos inmediatamente después de la palabra clave STRUCT dentro de la declaración de componentes.

En particular, un elemento de tipo de datos STRUCT puede estar a su vez estructurado. Es decir, está permitido el anidamiento de estructuras.

En el capítulo 10 verá cómo puede acceder a los datos de una estructura.

Especificación de tipo de datos STRUCT

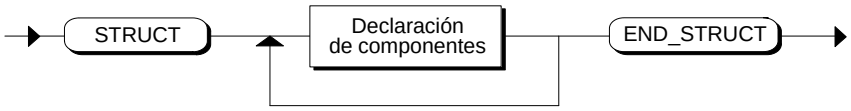


Figura 9-4 *Sintaxis: Especificación de tipo de datos STRUCT*

Declaración de componentes

Es un listado de los diferentes componentes que forman una estructura. Según el diagrama sintáctico 9-5, este listado está compuesto por:

- 1 a n identificadores con
- el tipo de dato asignado y
- una predefinición opcional de los valores iniciales.

Declaración de componentes

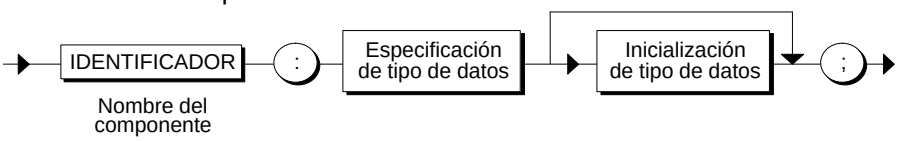


Figura 9-5 *Sintaxis: Declaración de componentes*

Identificador

El identificador es el nombre de un elemento de estructura al que se aplica la especificación del tipo de datos que le sigue.

Inicialización de tipo de datos

Ocasionalmente, después de la especificación de tipo de datos puede definir un elemento de estructura concreto con un valor inicial. Esta asignación se realiza mediante una asignación de valor y se describe en el capítulo 10.

Ejemplo

El siguiente ejemplo le muestra una posible definición de un tipo de datos STRUCT.

```
STRUCT
//INICIO Declaración de componentes
  A1       :INT;
  A2       :STRING[254];
  A3       :ARRAY [1..12] OF REAL;
//FIN Declaración de componentes
END_STRUCT
```

Nombres de componentes

Especificaciones de tipo de dato

Ejemplo 9-1 Definición de un tipo de datos STRUCT

9.4 Tipo de datos de usuario (UDT)

Resumen

Un tipo de datos UDT se define como un bloque. De acuerdo a su definición, este tipo de datos puede utilizarse en todo el programa de usuario de la CPU, por lo que es un tipo de datos global. Puede utilizar estos tipos de datos con su identificación UDT (UDTx, donde x representa números) o con un nombre simbólico asignado en la tabla de declaración de un bloque o de un bloque de datos.

Tipo de datos de usuario

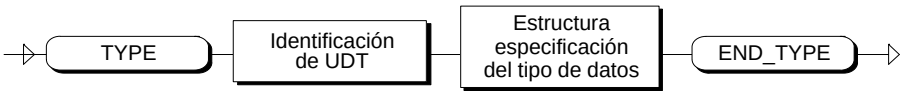


Figura 9-6 *Sintaxis: Tipo de datos de usuario (UDT)*

Identificación de UDT

La declaración de un UDT se inicia con la palabra clave TYPE, seguida del nombre del UDT (identificador de UDT). De acuerdo a lo expuesto en el capítulo 8, el nombre del UDT puede indicarse de forma absoluta, es decir, mediante un nombre estándar de forma UDTx (donde x es un número), o puede utilizarse un nombre simbólico.

Especificación del tipo de datos

Después de la identificación de UDT viene la especificación del tipo de datos. Aquí sólo está permitida la especificación del tipo de datos STRUCT (v. apt. 9.3.4):

```
STRUCT
:
END_STRUCT
```

A continuación, la declaración completa de un UDT se cierra con la palabra clave

```
END_TYPE
```

Utilización de un UDT

El tipo de datos así definido puede utilizarse para usar variables o parámetros y para la declaración de DB. También pueden declararse componentes de estructuras o arrays, incluso dentro de otros UDT, con ayuda del UDT.

Nota

Al adjudicar valores iniciales (inicialización) dentro de un UDT se aplica la sintaxis de AWL. En el manual de usuario **/231/** o en el manual **/232/** podrá consultar informaciones sobre las formas de escribir las constantes.

Ejemplo

El ejemplo le muestra una definición de UDT y el uso de este tipo de datos dentro de una declaración de variables. Se presupone que en la lista de símbolos se ha declarado el nombre "VALORES_MEDIDOS" para UDT50.

```
TYPE VALORES_MEDIDOS // Definición UDT
 STRUCT
  BIPOL_1 : INT;
  BIPOL_2 : WORD   := W#16#AFA1;
  BIPOL_3 : BYTE   := B#16#FF;
  BIPOL_4 : WORD   := B#(25,25);
  BIPOL_5 : INT    := 25;
  S_TIME  : S5TIME := S5T#1h20m10s;
  MEDICION: STRUCT
            BIPOLAR_10V    : REAL;
            UNIPOLAR_4_20MA : REAL;
           END_STRUCT;
 END_STRUCT
END_TYPE
```

```
FUNCTION_BLOCK FB11
VAR
   RANGO_VAL_MED: VALORES_MEDIDOS;
END_VAR
BEGIN
//...
   RANGO_VAL_MED.BIPOL:= -4;
   RANGO_VAL_MED.MEDICION.UNIPOLAR_4_20MA:= 2.7;
//...
END_FUNCTION_BLOCK
```

Ejemplo 9-2 Declaración de tipos de datos de usuario

9.5 Tipos de parámetros

Resumen

Además de los tipos de datos simples, compuestos y de usuario, para definir los parámetros formales de bloques de función y en funciones puede utilizar los denominados **tipos de parámetros**. Estos sirven para:

- poder declarar como parámetros funciones de temporizador y contador (`TIMER/ COUNTER`),
- poder declarar como parámetros FC, FB, DB y SDB (BLOCK_xx),
- admitir como parámetro un operando de cualquier tipo de datos (`ANY`).
- admitir como parámetro un área de memoria (POINTER).

Tabla 9-5 Tipos de parámetros

Parámetro	Tamaño	Descripción
TIMER	2 bytes	Identifica un determinado temporizador que será utilizado por el programa en el bloque lógico llamado. Parámetro actual: p.ej.: `T1`
COUNTER	2 bytes	Identifica un determinado contador que será utilizado por el programa en el bloque lógico llamado. Parámetro actual: p.ej.: `Z10`
BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	2 bytes	Identifica un determinado bloque que será utilizado por el programa en el bloque lógico llamado. Parámetro actual: p.ej.: `FC101` `DB42`
ANY	10 bytes	Se utiliza cuando como tipo de datos del parámetro actual debe permitirse cualquier tipo de datos, a excepción de ANY.
POINTER	6 bytes	Identifica una determinada área de memoria que será utilizada por el programa. Parámetro actual: p.ej.: `M50.0`

Temporizador (TIMER) y contador (COUNTER)

Definen un determinado temporizador o un determinado contador que se utilizará en el procesamiento de un bloque. Los tipos de datos `TIMER` y `COUNTER` sólo están permitidos para parámetros de entrada (`VAR_INPUT`).

Tipos de bloques

Definen un determinado bloque que será utilizado como parámetro de entrada. La declaración del parámetro de entrada determina el tipo de bloque (FB, FC, DB). Para especificar parámetros, indique el identificador absoluto del bloque, bien de forma absoluta (p. ej.: FB20) o mediante un nombre simbólico.

SCL no dispone de ninguna operación con estos tipos de datos. Unicamente pueden especificarse parámetros de este tipo al llamar a bloques. En las funciones no es posible especificar un parámetro de entrada.

En SCL puede asignar como parámetros actuales operandos de los siguientes tipos de datos:

- Bloques de función sin parámetros formales
- Funciones sin parámetro formal y sin valor de respuesta (VOID)
- Bloques de datos y bloques de datos de sistema.

ANY

En SCL existe la posibilidad de declarar parámetros de bloque del tipo de datos ANY; cuando se llama a uno de estos bloques, estos parámetros pueden especificarse con operandos de cualquier tipo de datos. Sin embargo, SCL ofrece sólo una posibilidad de procesar el tipo de datos ANY: seguir hasta bloques subordinados.

En SCL puede asignar como parámetros actuales operandos de los siguientes tipos de datos:

- Tipos de datos simples:
  indique la dirección absoluta o el nombre simbólico del parámetro actual.
- Tipos de datos compuestos:
  indique el nombre simbólico del dato de tipo compuesto (p.ej.:array y estructura).
- Tipo de datos `ANY:`
  Sólo es posible cuando el operando es un parámetro formal de tipo compatible.
- Tipo de datos `NIL:`
  Debe indicar un puntero cero.
- Temporizadores, contadores y bloques:
  indique el número (p.ej.: T1, Z20 o FB6).

El tipo de dato ANY está permitido para parámetros de entrada formales, para parámetros de entrada/salida de FB y FC y para parámetros de salida de FC.

Nota

Si al llamar a un FB o un FC asigna a un parámetro formal del tipo ANY una variable temporal, en el bloque llamado no podrá trasladar dicho parámetro a otro bloque. Las direcciones de las variables temporales pierden su validez al trasladarse.

PUNTERO

En SCL existe la posibilidad de declarar parámetros de bloque del tipo de datos `POINTER`; al llamar a uno de dichos bloques, a los parámetros pueden asignarse operandos de cualquier tipo de datos. Sin embargo, SCL ofrece sólo una posibilidad de procesar el tipo de datos `POINTER`: seguir hasta bloques subordinados.

En SCL puede asignar como parámetros actuales operandos de los siguientes tipos de datos:

- Tipos de datos simples:
  Debe indicar la dirección absoluta o el nombre simbólico del parámetro actual.
- Tipos de datos compuestos:
  Debe indicar el nombre simbólico de los datos de tipo compuesto (p.ej., arrays y estructuras).
- Tipo de datos `POINTER:`
  Sólo es posible cuando el operando es un parámetro formal de tipo compatible.
- Tipo de datos `NIL:`
  Debe indicar un puntero cero.

El tipo de datos `POINTER` está permitido para parámetros formales de entrada, de entrada/salida de FBs y FCs, y para parámetros de salida de FCs.

Nota

Si al llamar a un FB o un FC asigna a un parámetro formal del tipo `POINTER` una variable temporal, en el bloque llamado no podrá trasladar dicho parámetro a otro bloque. Las direcciones de las variables temporales pierden su validez al trasladarse.

Ejemplos

```
FUNCTION DISTANCIA: REAL
      VAR_INPUT
            MiDB:BLOCK_DB;
            Tempo : TIMER;
      END_VAR
      VAR
            INDICE: ENTERO;
      END_VAR
      BEGIN
      MiDB.DB5:=5;
      DISTANCIA:=....   // VALOR_RESPUESTA

END_FUNCTION
```

Ejemplo 9-3 Tipos de datos BLOCK_DB y TIMER

```
FUNCTION FC100: VOID
      VAR_IN_OUT
            in, out:ANY;
      END_VAR
      VAR_TEMP
            ret: INT;
      END_VAR
      BEGIN
      //...
      ret:=SFC20(DSTBLK:=out,SCRBLK:=in);
      //...
END_FUNCTION

FUNCTION_BLOCK FB100
      VAR
            ii:INT;
            aa, bb:ARRAY[1..1000] OF REAL;
      END_VAR
      BEGIN
      //...
      FC100(in:=aa, out:=bb);
      //...
END_FUNCTION_BLOCK
```

Ejemplo 9-4 Tipo de datos ANY

Declaración de variables locales y parámetros de bloque

10

Resumen

Las variables locales y los parámetros de bloque son datos que se declaran dentro de un bloque lógico (FC, FB, OB) y que sólo tienen validez para dicho bloque lógico. El presente capítulo le informa sobre la declaración y la inicialización de estos datos.

Indice del capítulo

10.1 Resumen

Clasificación de las variables

Las variables locales pueden clasificarse en las siguientes categorías, como muestra la tabla 10-1:

Tabla 10-1 Variables locales

Variables locales	Significado
Variables estáticas	Las variables estáticas son variables locales cuyo valor se conserva a lo largo de todos los recorridos del bloque (memoria de bloque). Sirven para guardar valores de un bloque de función y se depositan en el bloque de datos de instancia.
Variables temporales	Las variables temporales se corresponden localmente con un bloque lógico y **no** ocupan área de memoria estática, puesto que se depositan en la pila de la CPU. Su valor sólo se conserva durante el recorrido de un bloque. A las variables temporales **no** se puede acceder fuera del bloque en el que se han declarado las variables.

Clasificación de los parámetros de bloque

Los parámetros de bloque son comodines que sólo pueden definirse cada vez que se utiliza de forma concreta (llamada) el bloque. Los comodines existentes en el bloque se denominan parámetros **formales**, y los valores asignados cuando se llama al bloque se denominan parámetros **actuales**. Los parámetros formales de un bloque pueden considerarse como variables locales.

Los parámetros de bloque pueden clasificarse en las siguientes categorías, como muestra la tabla 10-2:

Tabla 10-2 Parámetros de bloque

Parámetros de bloque	Significado
Parámetros de entrada	Los parámetros de entrada asumen los valores de entrada actuales cuando se llama al bloque. Sólo admiten lectura.
Parámetros de salida	Los parámetros de salida transmiten los valores de salida actuales al bloque invocante. Pueden escribirse y leerse.
Parámetros de entrada/salida	Cuando se produce una llamada, los parámetros de entrada/salida asumen el valor actual de una variable, lo procesan y a continuación devuelven los resultados a la misma variable.

Marcas OK (OK flag)

El compilador SCL ofrece una marca OK que sirve para detectar errores durante la ejecución de programas en la CPU. Es una variable local de tipo BOOL con el nombre predefinido ”OK”.

Declaración de variables y parámetros

Como muestra la tabla 10-3, cada categoría de variables locales o parámetros tiene asignado un bloque de declaración propio, identificado por su propia pareja de palabras clave.

Cada bloque contiene las declaraciones que están permitidas para dicho bloque de declaración. Sólo puede presentarse una vez en la tabla de declaración del bloque, y los bloques pueden tener un orden cualquiera.

En la tabla 10-3, los bloques de declaración permitidos en un bloque están marcados con una ”x”.

Tabla 10-3 Bloques de declaración para variables locales y parámetros

Datos	Sintaxis	FB	FC	OB
Variable estática	VAR : END_VAR	X	X[1)]	
Variable temporal	VAR_TEMP : END_VAR	X	X	X
Parámetro de bloque como: Parámetro de entrada	VAR_INPUT : END_VAR	X	X	
Parámetro de salida	VAR_OUTPUT : END_VAR	X	X	
Parámetro de entrada/salida	VAR_IN_OUT : END_VAR	X	X	

1) La declaración de variables dentro de la pareja de palabras clave VAR y END_VAR sí está permitida en funciones, pero durante la compilación las declaraciones se desplazan al área temporal.

Inicialización

Al realizar la declaración debe asignar a las variables y parámetros un tipo de datos que determina la estructura y, por lo tanto, también la memoria necesaria. Además, también puede asignar valores iniciales a las variables estáticas y a los parámetros de un bloque de función. La tabla 10-4 le ofrece un resumen de los casos en que es posible una inicialización.

Tabla 10-4 Inicialización de datos locales

Categoría de datos	Inicialización
Variables estáticas	posible
Variables temporales	no posible
Parámetros de bloque	sólo posible con parámetros de entrada y con parámetros de salida de un bloque de función

10.2 Declaración de variables y parámetros

Resumen

Una declaración de variables y parámetros está compuesta por un identificador que puede elegirse a voluntad para el nombre de la variable, y de una especificación del tipo de datos. La forma general se muestra en el diagrama sintáctico. La asignación de atributos del sistema para parámetros se describe con detalle en el apartado 8.4.

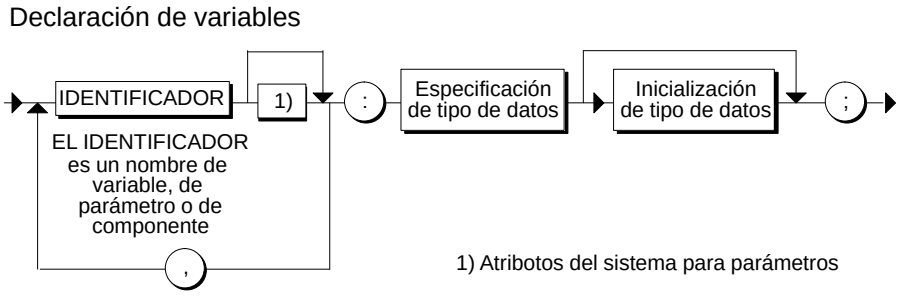


Figura 10-1*Sintaxis: Declaración de variable*

Ejemplos de declaraciones válidas son los siguientes:

```
VALOR1 : REAL;
```

o, en caso de varias variables del mismo tipo:

```
VALOR2, VALOR3,VALOR4,....: INT;

ARRAY   : ARRAY[1..100, 1..10] OF REAL;

CONJUNTO: STRUCT
            ARRAY_MEDICION:ARRAY[1..20] OF REAL;
            INTERRUPTOR:BOOL;
          END_STRUCT
```

Especificación de tipo de datos

Están permitidos todos los tipos de datos que se han tratado en el capítulo 9.

Nota

Las palabras reservadas que sólo tienen validez en SCL puede declararlas como identificadores anteponiéndoles el carácter ”#” (p.ej., #FOR). Esto puede ser útil cuando desee transferir parámetros actuales a bloques que han sido creados en otro lenguaje (p.ej., AWL).

10.3 Inicialización

Principio

En la declaración de variables estáticas, de parámetros de entrada y de parámetros de salida de un FB puede predefinirse un valor. La predefinición se realiza asignando un valor (:=) después de la indicación del tipo de datos. De acuerdo con el diagrama sintáctico 10-2, es posible:

- asignar una constante a una variable simple, o
- asignar una lista de inicialización a un array.

Inicialización

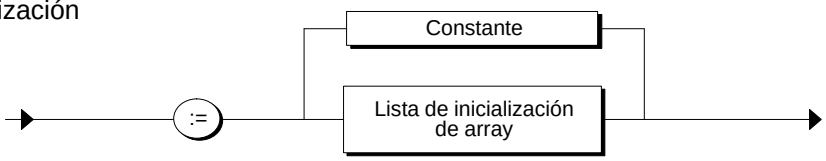


Figura 10-2 *Sintaxis: Inicialización*

Ejemplo:

```
VALOR   :REAL   := 20.25;
```

Tenga en cuenta que no es posible inicializar una lista de variables (A1, A2, A3,...:INT:=...). En este caso debe inicializar las variables independientemente. Pueden predefinirse valores iniciales para los arrays de acuerdo con la figura 10-3.

Lista de inicialización de array

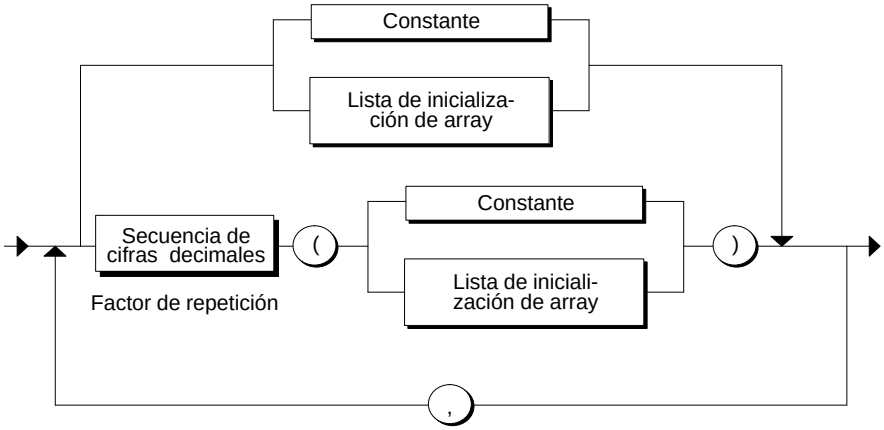


Figura 10-3 *Sintaxis: Lista de inicialización de array*

```
ARRAY :     ARRAY[1..10, 1..100] OF INT:=10(100(0));
```

Factor de repetición (Número de columnas)

Factor de repetición (Número de líneas)

Valor

Ejemplos

El ejemplo 10-1 muestra la inicialización de una variable estática:

```
VAR
      INDICE1: INT:= 3;
END_VAR
```

Ejemplo 10-1 Inicialización de variables estáticas

La inicialización de un array bidimensional se muestra en el ejemplo 10-2. Si desea declarar la siguiente estructura de datos en SCL con el nombre REGULADOR, escriba:

-54	736	-83	77
-1289	10362	385	2
60	-37	-7	103
60	60	60	60

```
VAR
REGULADOR:
ARRAY [1..4, 1..4] OF INT:=  -54,  736, -83, 77,
                            -1289, 10362, 385,  2,
                               60,  -37,  -7, 103,
                            4(60);

END_VAR
```

Ejemplo 10-2 Inicialización de array

El ejemplo 10-3 le muestra una inicialización de estructura:

```
VAR
GENERADOR:STRUCT
  DATOS: REAL     := 100.5;
  A1:    INT      := 10;
  A2:    STRING[6]:= 'FACTOR';
  A3:    ARRAY[1..12] OF REAL:= 12(100.0);
END_STRUCT;
END_VAR
```

Ejemplo 10-3 Inicialización de estructura

10.4 Declaración de instancias

Resumen

Además de las variables de tipo de datos simples, compuestos o de usuario ya conocidas, en la tabla de declaración de bloques de función puede declarar también variables del tipo FB o SFB. Estas variables se denominan **instancias locales** del FB o del SFB.

Los datos de instancia locales se guardan en el bloque de datos de instancia del bloque de función invocante.

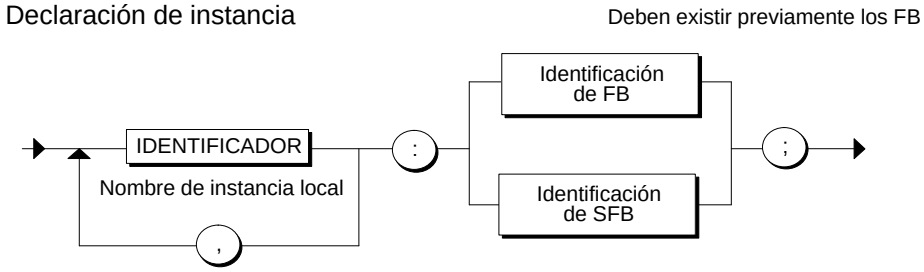


Figura 10-4 *Sintaxis: Declaración de instancias*

Ejemplos: Algunos ejemplos sintácticamente válidos según la figura 10-4 son:

```
Suministro1 : FB10;
Suministro2,Suministro3,Suministro4: FB100;
Motor1      : Motor ;
```

`// Motor` es un símbolo que se ha introducido en la tabla de símbolos.

Símbolo	Dirección	Tipo de dato
MOTOR	FB20	FB20

Figura 10-5 Tabla de símbolos correspondiente en STEP 7

Inicialización

No es posible una inicialización local específica de instancia.

10.5 Variables estáticas

Resumen

Una variable estática es una variable local cuyo valor se mantiene a lo largo de todos los recorridos del bloque (memoria de bloque). Sirve para guardar valores de un **bloque de función**. Las variables se guardan en el bloque de datos de instancia de su bloque de función.

Bloque de variables estáticas

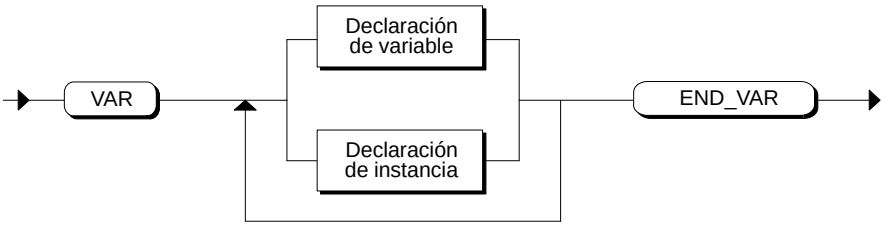


Figura 10-6 *Sintaxis: Bloque de variables estáticas*

Bloque de declaración
VAR
END_VAR

Este bloque de declaración forma parte de la tabla de declaración de FB. En este bloque podrá:

- declarar nombres de variables y tipos de datos con la declaración de variables (v. apt. 10.2) disponiendo opcionalmente de la posibilidad de inicialización, o
- insertar otras declaraciones de variables ya existentes con la declaración de instancia (v. apt. 10.4).

Después de la compilación, este bloque constituye, junto con los bloques para parámetros de bloque, la estructura del bloque de datos de instancia asignado.

Ejemplo

El ejemplo 10-4 muestra la declaración de variables estáticas:

```
VAR
   RECORRIDO       :INT;
   ARRAY_MEDICION  :ARRAY[1..10] OF REAL;
   INTERRUPTOR     :BOOL;
   MOTOR_1,Motor_2 :FB100;   //Declaración de instancia

END_VAR
```

Ejemplo 10-4 Declaración de variables estáticas

Acceso

El acceso a las variables se realiza en el área de instrucciones:

- **Acceso desde el interior**: es decir, en el área de instrucciones del bloque de función en cuya tabla de declaración se ha declarado la **variable**. Este acceso se ha explicado en el capítulo 14 ”Asignación de valores”.
- **Acceso desde el exterior a través de bloques de datos de instancia:** A través de la variable indicada *DBx variable* (DBx es la identificación de bloque de datos).

10.6 Variables temporales

Resumen

Las variables temporales pertenecen localmente a un bloque lógico y **no** ocupan ningún área de memoria estática. Su valor sólo se conserva durante el recorrido de un bloque. En calidad de variables temporales, **no** puede accederse a ellas fuera del bloque en el que se han declarado las variables.

Deberá declarar como datos temporales aquellos datos que sólo los necesite para guardar resultados temporales durante el procesamiento de su OB, FB o FC.

Bloque de variables temporales

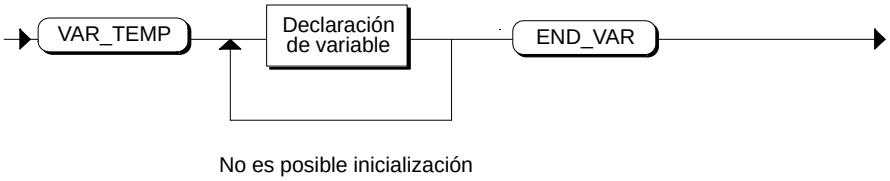


Figura 10-7 *Sintaxis: Bloque de variables temporales*

Bloque de declaración
VAR_TEMP
END_VAR

Este bloque de declaración es parte de un FB, FC u OB. Dentro de la declaración de variables, los nombres de las variables y los tipos de datos se indican como se ha explicado en el apartado 10.2.

Al iniciarse la ejecución de un OB, FB o FC, el valor de los datos temporales no está definido. No es posible la inicialización.

Ejemplo

El ejemplo 10-5 muestra la declaración de variables temporales de bloque:

```
VAR_TEMP
   BUFER_1        :ARRAY [1..10] OF INT;
   AYUDA1,AYUDA2  :REAL;
END_VAR
```

Ejemplo 10-5 Declaración de variables temporales de bloque

Acceso

El acceso a las variables se realiza siempre en el área de instrucciones del bloque lógico, en cuya tabla de declaración se ha declarado la **variable** (**acceso interno**). Información más detallada al respecto encontrará en el capítulo 14 ”Asignación de valores”.

10.7 Parámetros del bloque

Resumen

Los parámetros de bloques son parámetros formales de un bloque de función o de una función. Cuando se llama al bloque de función o a la función, los parámetros actuales sustituyen a los parámetros formales y constituyen así un mecanismo de intercambio de información entre ambos bloques.

- Los parámetros formales de entrada adoptan los valores de entrada actuales (flujo de datos de fuera hacia dentro).
- Los parámetros formales de salida sirven para transferir valores de salida (flujo de datos de fuera adentro).
- Los parámetros formales de entrada/salida tienen también la función de un parámetro de entrada y de un parámetro de salida.

En el capítulo 16 obtendrá más información sobre el uso de parámetros y el intercambio de información entre los bloques.

Bloque de parámetros

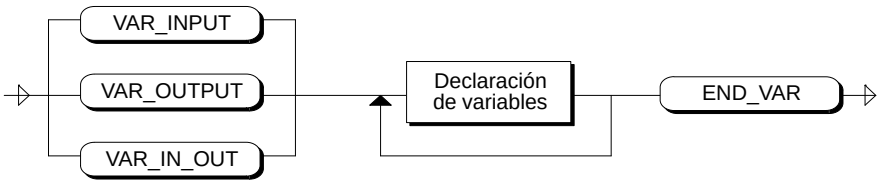


Figura 10-8 *Sintaxis: Bloque de parámetros*

Bloque de declaración
VAR_INPUT
VAR_OUTPUT
VAR_IN_OUT

Este bloque de declaración es un componente del FB o de la FC. De acuerdo con el apartado 10.2, el nombre de variable y el tipo de dato asignado se indican dentro de la declaración de variable.

Después de compilar un FB, estos bloques determinan, junto con el bloque VAR y END_VAR, la estructura del bloque de datos de instancia asignado.

Ejemplo

El ejemplo 10-6 muestra la declaración de un parámetro:

```
VAR_INPUT              //Parámetro de entrada
  REGULADOR       :DWORD;
  HORA_DEL_DIA    :TIME_OF_DAY;
END_VAR
```

```
VAR_OUTPUT             //Parámetro de salida
  VALORES_TEORICOS: ARRAY [1..10] OF INT;
END_VAR
```

```
VAR_IN_OUT             //Parámetro de entrada/salida
  AJUSTE: INT;
END_VAR
```

Ejemplo 10-6 Declaración para parámetros

Acceso

El acceso a los parámetros de bloque se realiza en el área de instrucciones de un bloque lógico:

- **Acceso interno**: es decir, en el área de instrucciones del bloque en cuya tabla de declaración se han declarado los **parámetros**. Se explica en el capítulo 14 ”Asignación de valores” y en el capítulo 13 ”Expresiones, operadores y operandos”.
- **Acceso desde exterior a través de DB de instancia**: A los parámetros de FB se puede acceder a través del DB de instancia asociado (v. apt. 14.8).

10.8 Marcas OK (OK flag)

Descripción

La marca OK sirve para indicar la ejecución correcta o incorrecta de un bloque. Es una variable global del tipo BOOL y palabra clave ”OK”.

Si durante la ejecución de una instrucción del bloque se produce un error (p.ej.: un desbordamiento en una multiplicación), la marca OK pasa a posición FALSE. Al abandonar el bloque el valor de la marca OK se guarda en el parámetro de salida ENO definido implícitamente (v. apt. 16.4), de forma que puede ser valorado por el bloque invocante.

Al iniciarse el recorrido del bloque la marca OK tiene el valor TRUE. Mediante instrucciones SCL puede consultarse en cualquier punto del bloque si la marca está en posición TRUE/FALSE.

Declaración

La marca OK es una variable de declaración del sistema. No es necesario realizar una declaración. Pero antes de la compilación deberá seleccionar la opción de compilador ”Activar OK flag” si desea utilizar la marca OK en su programa de usuario.

Ejemplo

El ejemplo 10-7 ilustra la utilización de la marca OK:

```
      // Colocar variable OK en TRUE,
      // para que pueda verificarse si la
      // acción siguiente se ejecuta
      // correctamente.
OK: = TRUE;
SUM: = SUM + IN;
IF OK THEN
      // La adición ha transcurrido
      // de forma correcta.
      //:
ELSE  // La adición ha transcurrido
      // con error.
END_IF;
```

Ejemplo 10-7 Utilización de la marca OK

Declaración de constantes y metas de salto

11

Resumen

Las constantes son datos que poseen un valor fijo y que no puede cambiarse durante el tiempo de ejecución del programa. Cuando el valor de las constantes se expresa mediante su escritura, se habla de **constantes literales**.

Las constantes no necesitan ninguna declaración de variables. Sin embargo, puede adjudicar nombres simbólicos a las constantes mediante el bloque de declaración.

Las metas de salto representan los nombres de las metas del salto dentro del área de instrucciones del bloque lógico.

Tanto los nombres simbólicos de las constantes como las metas de salto se declaran por separado en sus propios bloques de declaración.

Indice del capítulo

11.1 Constantes

Utilización de constantes

En las asignaciones de valor y expresiones, además de variables y parámetros de bloque se utilizan constantes. Las constantes pueden utilizarse como constantes literales o con un nombre simbólico.

Declaración de nombres simbólicos

La declaración de nombres simbólicos para constantes se realiza dentro del bloque de declaración CONST en la tabla de declaración de su bloque lógico (ver apartado 8.4).

Bloque de constantes

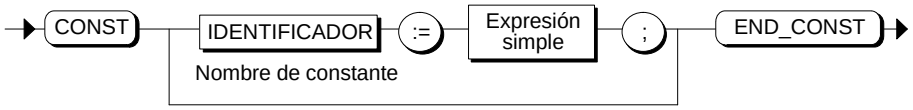


Figura 11-1 *Sintaxis: Bloque de constantes*

En este caso la expresión simple representa expresiones aritméticas en las que puede utilizar las operaciones básicas +, -, *, /, DIV y MOD.

Ejemplo

El ejemplo 11-1 ilustra la declaración de nombres simbólicos:

```
CONST
 Numero        := 10 ;
 HORA_DEL_DIA1 := TIME#1D_1H_10M_22S.2MS ;
 NOMBRE        := 'SIEMENS' ;
 NUMERO2       := 2 * 5 + 10 * 4 ;
 NUMERO3       := 3 + NUMERO2 ;
 END_CONST
```

Ejemplo 11-1 Declaración de constantes simbólicas

Escrituras

SCL ofrece diversas formas de escritura (formatos) para introducir o visualizar constantes. Estas formas de escritura se denominan literales. La información que sigue a continuación se ocupa de cada uno de los literales individualmente.

11.2 Literales

Definición

El literal es una escritura formal para definir el valor y el tipo de una constante. Existen los siguientes grupos de literales:

- literal numérico
- literal de carácter
- indicación de tiempos.

Dependiendo del tipo de datos y del formato existe una determinada forma de escritura para el valor de una constante:

```
15          VALOR_15    como número entero en representación decimal.
2#1111      VALOR_15    como número entero en representación binaria.
16#F        VALOR_15    como número entero en representación hexadecimal.
```

Literal con varias formas de escritura para el valor 15

Asignación de tipos de datos a las constantes

A una constante se asigna el tipo de datos cuyo rango es mínimamente suficiente para aceptar la constante sin pérdida de valor. El tipo de datos asignado debe ser acorde con el tipo de datos final existente cuando se utilizan las constantes para una asignación a una variable o a una expresión. Por ejemplo, si se define un literal entero cuyo valor excede del rango del entero, se supone que se trata de un entero doble. El compilador emite un mensaje de error cuando usted asigna ese valor a una variable del tipo entero.

11.3 Escrituras para literales de números enteros y números reales

Resumen

Para escribir valores numéricos SCL ofrece la posibilidad de utilizar:

- literales enteros para valores numéricos enteros, y
- literales numéricos reales para números de coma flotante.

En ambos literales utilice una secuencia de cifras que tenga la estructura de la fig. 11-2. En los siguientes diagramas sintácticos esta frecuencia de cifras se designará de forma simplificada como secuencia de cifras decimales.

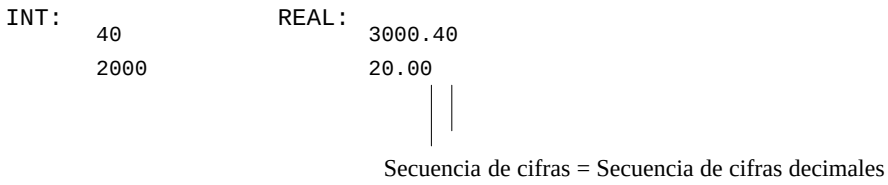


Figura 11-2 Secuencia de cifras en un literal

El número decimal dentro de literales está formado por una secuencia de cifras que opcionalmente pueden separarse mediante guiones inferiores. Los guiones inferiores ayudan para mejorar la legibilidad cuando los números son grandes.

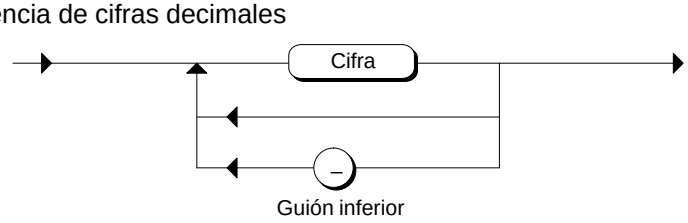


Figura 11-3 *Sintaxis: Secuencia de cifras decimales*

Los siguientes ejemplos son escrituras permitidas para las secuencias de cifras decimales dentro de literales:

```
1000
1_120_200
666_999_400_311
```

Literales enteros

Los literales enteros son valores numéricos enteros. En el programa SCL pueden asignarse a estos literales, dependiendo de su longitud, variables de los tipos de datos:

```
BOOL, BYTE, INT, DINT, WORD, DWORD
```

La figura 11-4 muestra la sintaxis de un literal entero:

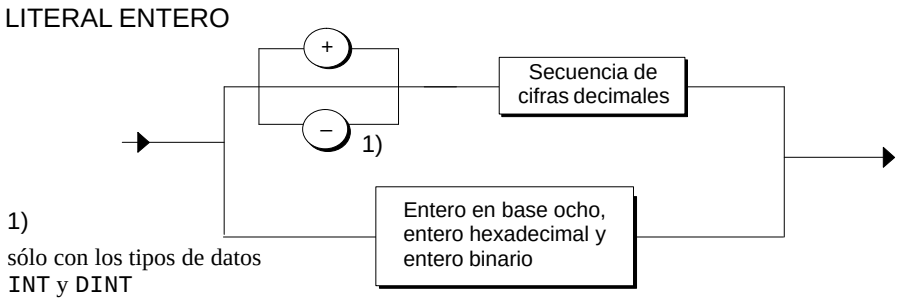


Figura 11-4 *Sintaxis: Literal entero*

Los siguientes ejemplos son escrituras permitidas para la secuencia de cifras decimales dentro de literales enteros:

```
1000
+1_120_200
-666_999_400_311
```

Valores binarios/ en base ocho/ hexadecimales

Un literal entero puede expresarse en otro sistema de numeración diferente del decimal; esto puede hacerse anteponiendo los prefijos 2#, 8# ó 16# seguidos del número, expresado en el sistema correspondiente. El guión inferior puede insertarse opcionalmente entre las cifras para mejorar la legibilidad de los números grandes.

La escritura general de un literal entero se explica en la figura 11-5 con el ejemplo de una secuencia de cifras para un número en base ocho:

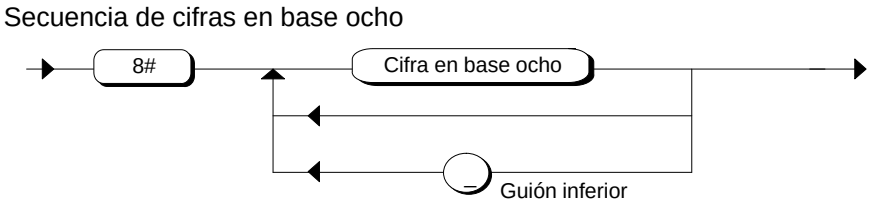


Figura 11-5 *Sintaxis: Secuencia de cifras en base ocho*

Los siguientes ejemplos son posibles escrituras de literales enteros:

```
Valor_2:=2#0101; // Número binario, valor decimal 5
Valor_3:=8#17;   // Número en base ocho, valor decimal 14
Valor_4:=16#F;   // Número hexadecimal, valor decimal 15
```

Literales numéricos reales

Los literales numéricos reales son valores con decimales. Pueden asignarse variables con tipo de dato REAL. La indicación del signo es opcional. Si no se indica ningún signo el número se considera positivo. La figura 11-6 muestra la sintaxis de un número real:

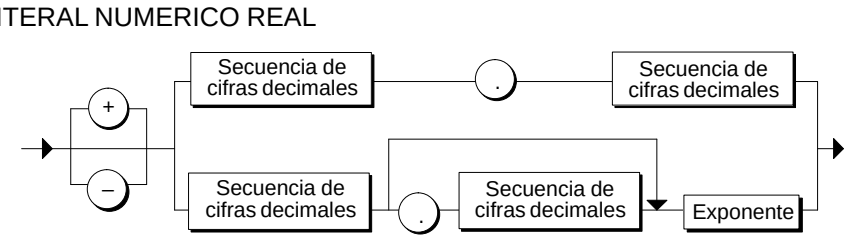


Figura 11-6 *Sintaxis: Literal numérico real*

En la escritura exponencial puede utilizar un exponente para expresar números de coma flotante. El exponente se expresa anteponiendo la letra ”E” o ”e” seguida de un número decimal. La figura 11-7 muestra la sintaxis para expresar un exponente.

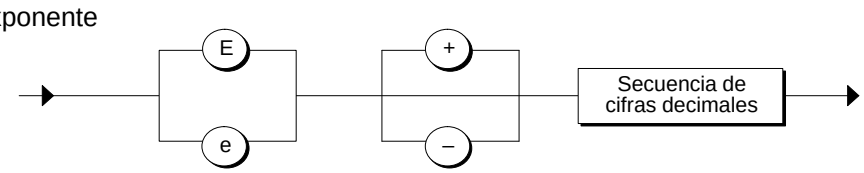


Figura 11-7 *Sintaxis: Exponente*

Ejemplo:

En SCL la magnitud 3 x 10 ^{10} puede representarse por los siguientes números reales:

```
3.0E+10   3.0E10        3e+10       3E10
0.3E+11   0.3e11        30.0E+9     30e9
```

Ejemplos

El ejemplo final que viene a continuación resume una vez más todas las posibilidades:

```
// Literales enteros
NUMERO1:= 10 ;
NUMERO2:= 2#1010 ;
NUMERO3:= 16#1A2B ;
// Literales numéricos reales
NUMERO4:= -3.4 ;
NUMERO5:= 4e2 ;
NUMERO6:= 40_123E10;
```

Ejemplo 11-2 Literales numéricos

11.4 Escrituras para literales de caracteres y literales de cadenas

Resumen

SCL también ofrece la posibilidad de introducir y procesar informaciones textuales; por ejemplo, una cadena de caracteres que debe visualizarse como mensaje.

Con los literales de caracteres no es posible efectuar cálculos: es decir, que los literales de caracteres **no** pueden utilizarse en expresiones. Se distingue entre:

- literal de carácter, que son caracteres aislados, y un
- literal de cadena (string), que es una secuencia de caracteres compuesta por un máximo de 254 caracteres individuales.

Literal de carácter (carácter individual)

El literal de carácter tiene exactamente un carácter (v. fig. 11-8). El carácter está encerrado entre ”apóstrofos” (’).

LITERAL DE CARACTER

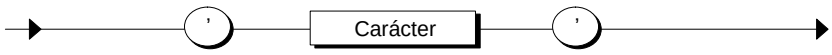


Figura 11-8 *Sintaxis: Literal de carácter*

Ejemplo:

```
CARACTER_1 := ’B’;       // Carácter B
```

Literal de cadena (string)

Un literal ’string’ es una cadena de caracteres compuesta por un máximo de 254 caracteres (letras, cifras y signos especiales) encerrados entre apóstrofos (’). Pueden utilizarse letras mayúsculas y minúsculas.

LITERAL DE STRING

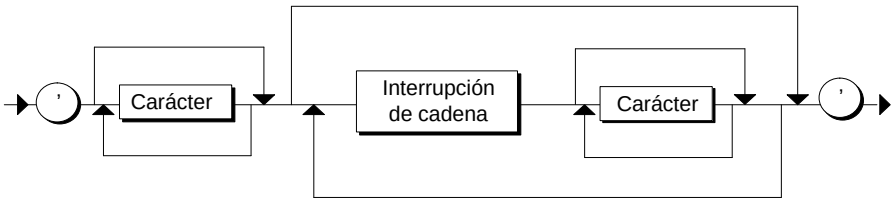


Figura 11-9 *Sintaxis: Literal de string*

Algunos literales string válidos son los siguientes:

```
’ROJO’    ’7500 Karlsruhe’          ’270–32–3456’
’DM19.95’ ’La respuesta correcta es:’
```

Tenga en cuenta que cuando se asigna un literal string a una variable string el número de caracteres está limitado a < 254.

La asignación de valor:

`TEXT:STRING[20]:='SIEMENS KARLSRUHE Rheinbrückenstr.'`

provoca un mensaje de error, y en la variable '`TEXT`' se deposita:

'`SIEMENS KARLSRUHE Rh`'

Los signos de formateado especiales, el apóstrofo (`'`) o un símbolo `$` puede introducirlos con el símbolo de escape $. Puede **interrumpir** y continuar varias veces un literal string.

Interrupción de cadena (string)

Un 'string' se encuentra situado en una línea de un bloque SCL o dividido en varias líneas mediante identificadores especiales. Para interrumpir un string se utiliza la identificación '`$>`'; y para continuarlo en una línea subsiguiente se utiliza la identificación '`$<`'.

```
TEXT:STRING[20]:='FB$>//Versión anterior
$<convertido';
```

El espacio entre el identificador de interrupción y el identificador de continuación puede ocupar varias líneas e incluir únicamente comentario y blancos. De esta forma, puede interrumpir y continuar varias veces un literal string (v. fig. 11-10).

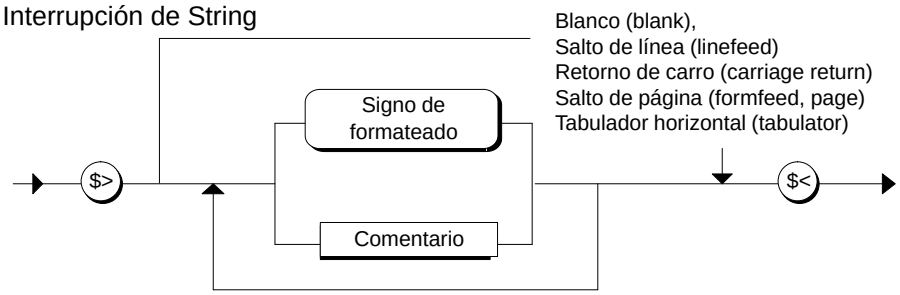


Figura 11-10 *Sintaxis: Interrupción de string*

Caracteres imprimibles

Los caracteres de un literal de carácter o de cadena (string) pueden ser cualquiera de los del juego de caracteres ampliado completo de ASCII. Los caracteres de formateado especiales y los caracteres no representables directamente (' y $) debe introducirlos con ayuda del símbolo de escape $.

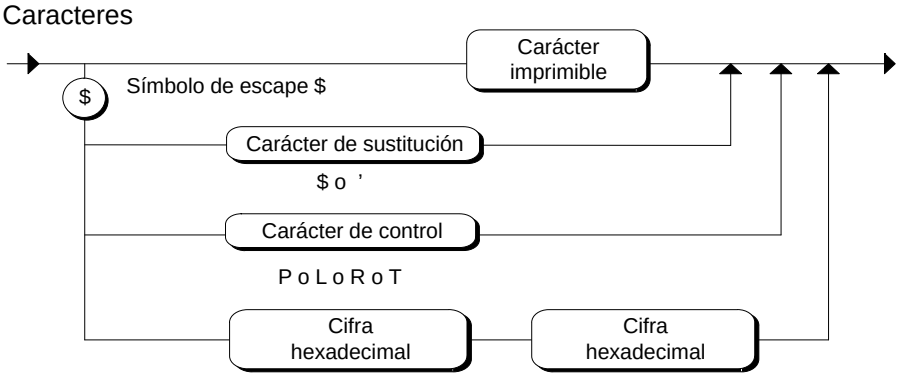


Figura 11-11 *Sintaxis: Caracteres*

Caracteres no imprimibles

En un literal de carácter o de string puede introducir caracteres no imprimibles incluidos en el juego de caracteres ampliado completo de ASCII. Para ello debe introducirlos en su representación en código hexadecimal.

Introduzca un carácter ASCII mediante **$hh**, donde **hh** representa el valor del carácter ASCII expresado en el sistema hexadecimal.

Ejemplo:

```
CARACTER_A  :='$65'; //corresponde al carácter 'A'
Blan#co     :='$32'; //corresponde al carácter ␣
```

En el anexo A encontrará más información sobre los caracteres de sustitución y control.

Ejemplos

Los siguientes ejemplos ilustran la formulación de literales de carácter:

```
// Literal de carácter
Carácter:= 'S' ;

// Literal string:
NOMBRE:= 'SIEMENS' ;

// Interrupción de un literal string:
MENSAJE1:= 'MOTOR- $>
$< Steuerung' ;

 // String en representación hexadecimal:
 MENSAJE1:= '$41$4E' (*Secuencia de caracteres AN*);
```

Ejemplo 11-3 Literales de carácter

11.5 Escrituras de datos de temporizador

Escritura de tipos de temporizador

SCL ofrece diferentes formatos para introducir valores de hora y de fecha. Son posibles las siguientes indicaciones de tiempo:

```
Fecha
Tiempo
Hora del día
Fecha y hora
```

Fecha

Una fecha está precedida por el prefijo `DATE#` o `D#` (v. fig. 11-12).

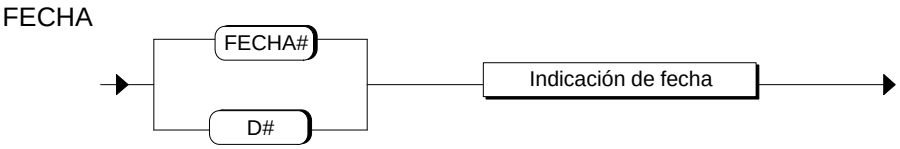


Figura 11-12 *Sintaxis: Fecha*

La **indicación de fecha** se realiza con números enteros para el número del año (4 caracteres), y para introducir el mes y el día, separando los números con guión.

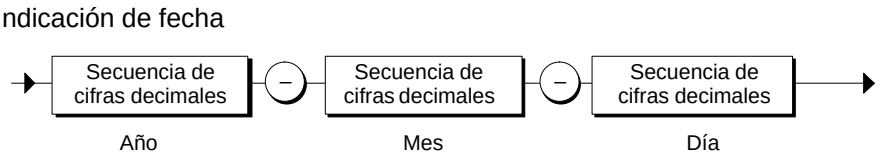


Figura 11-13 *Sintaxis: Indicación de fecha*

Son indicaciones de fecha válidas las siguientes:

```
// Indicación de fecha
VARIABLE_TEMP1:= DATE#1995-11-11;
VARIABLE_TEMP2:= D#1995-05-05;
```

Tiempo

Un tiempo se inicia con el prefijo `TIME#` o `T#` (v. fig. 11-14). La indicación de tiempo puede hacerse de dos formas:

- Representación decimal
- Representación escalonada.

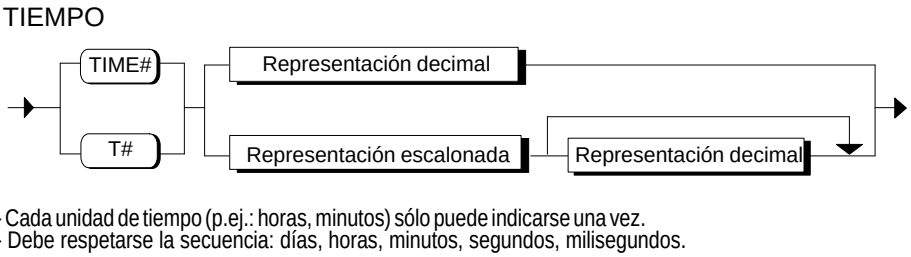


Figura 11-14 *Sintaxis: Tiempo*

Utilice la **representación decimal** cuando tenga que expresar su tiempo alternativamente en días, horas, minutos, segundos o milisegundos:

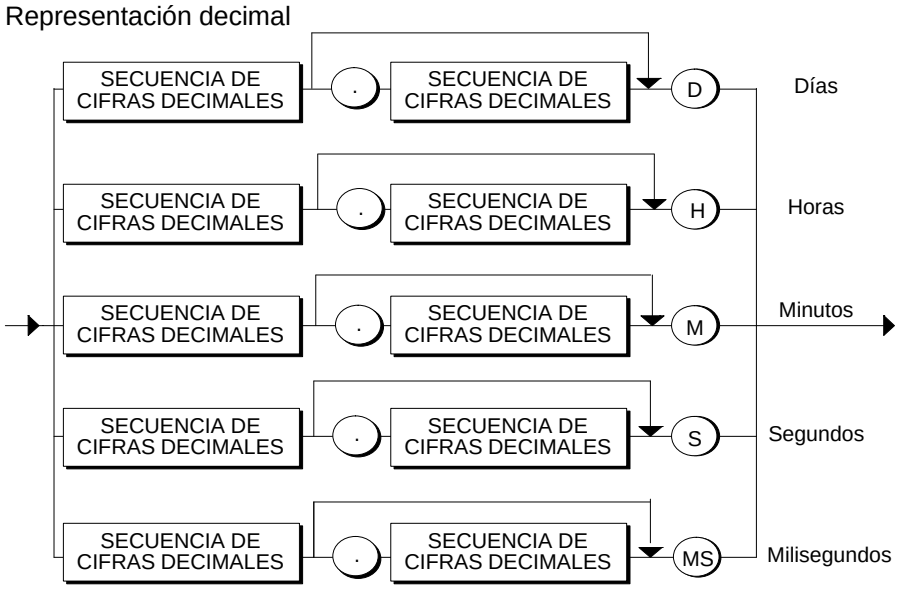


Figura 11-15 *Sintaxis: Representación decimal*

Ejemplos

Son indicaciones válidas las siguientes:

```
TIME#20.5D          para 20,5  días
TIME#45.12M         para 45,12 minutos
T#300MS             para 300 milisegundos
```

Utilice la representación escalonada cuando tenga que indicar su tiempo como una secuencia de días, horas, minutos, segundos o milisegundos. Está permitido omitir componentes individuales (v. fig. 11-16). Pero como mínimo debe indicarse un componente.

Representación escalonada

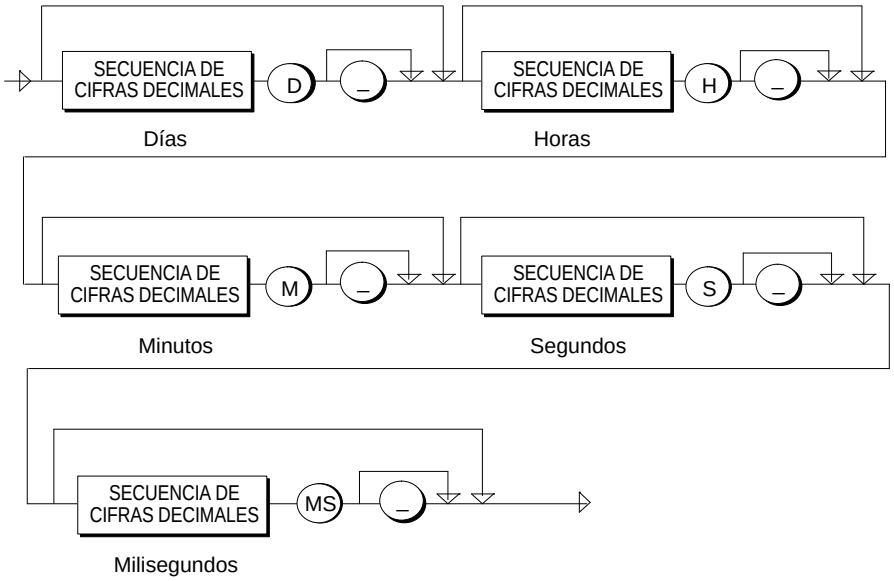


Figura 11-16 *Sintaxis: Representación escalonada*

Son indicaciones válidas las siguientes:

```
TIME#20D_12H
TIME#20D_10H_25M_10s
TIME#200S_20MS
```

Hora del día

Como indica la figura 11-17, una hora del día se inicia con el prefijo TIME_OF_DAY# o TOD#.

HORA DEL DIA

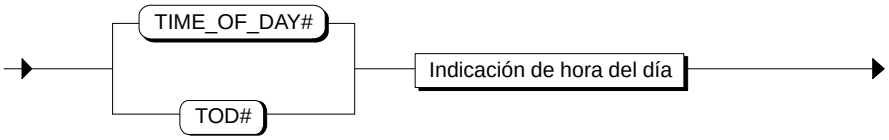


Figura 11-17 *Sintaxis: Hora del día*

Para la **indicación de** la **hora del día** se declaran las horas, minutos y segundos separados por dos puntos. La indicación de los milisegundos es opcional. En caso de utilizarse, se separa de los demás datos mediante un punto. La figura 11-18 muestra la sintaxis de indicación de hora del día:

Indicación de hora del día

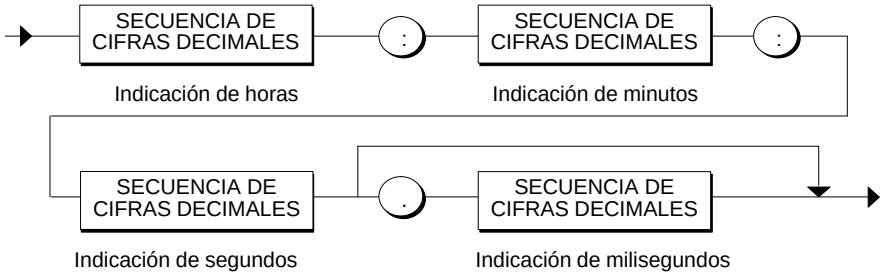


Figura 11-18 *Sintaxis: Indicación de hora del día*

Son indicaciones válidas las siguientes:

```
// Indicación de hora del día
HORA_DEL_DIA1:= TIME_OF_DAY#12:12:12.2;
HORA_DEL_DIA2:= TOD#11:11:11.200;
```

Fecha y hora

Una fecha y hora se indica anteponiendo el prefijo `DATE_AND_TIME#` o DT# (v. fig. 11-19). Es un literal compuesto de un dato de fecha y un dato de hora del día.

FECHA Y HORA

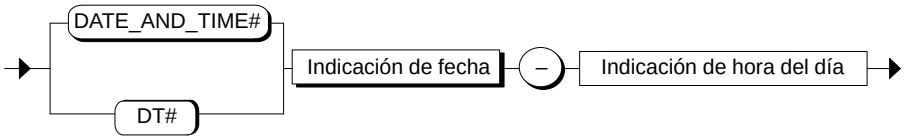


Figura 11-19 *Sintaxis: Fecha y hora*

El siguiente ejemplo ilustra la indicación de fecha y hora del día:

```
// Indicación de hora del día
HORA_DEL_DIA1:= DATE_AND_TIME#1995-01-01-12:12:12.2;
HORA_DEL_DIA2:= DT#1995-02-02-11:11:11;
```

11.6 Metas de salto

Descripción

Las metas de salto (labels) sirven para definir la meta de una instrucción GOTO (v. apt. 11.4).

Descripción de metas de salto

Las metas de salto se declaran en la tabla de declaración de un bloque lógico utilizando su nombre simbólico (v. apt. 8.4):

Bloque de metas de salto

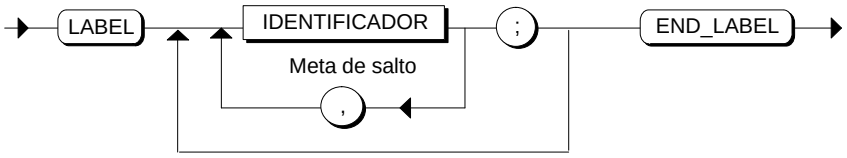


Figura 11-20 *Sintaxis: Bloque de meta de salto*

Ejemplo

El ejemplo 11-4 ilustra la declaración de meta de salto:

```
LABEL
        MARCA1, MARCA2, MARCA3;
END_LABEL;
```

Ejemplo 11-4 Metas de salto

Declaración de datos globales

12

Resumen

Los datos globales son datos que pueden utilizarse desde cualquier bloque lógico (FC, FB, OB). A estos datos puede tenerse acceso absoluto o simbólico. En este capítulo vamos a presentarle cada una de las áreas de datos, describiendo cómo puede acceder a estos datos.

Indice del capítulo

12.1 Resumen

Datos globales

En SCL, dispone de la posibilidad de acceder a datos globales. Existen dos tipos de datos globales:

- **Areas de memoria de la CPU:**

  Estas áreas de memoria son datos declarados por el sistema: p. ej.: entradas, salidas y marcas (v. apt. 7.5). El número de ellas de que dispone está determinado por su PLC.

- **Datos de usuario globales como bloques de datos susceptibles de ser cargadas:**

  Estas áreas de datos se encuentran dentro de los bloques de datos. Para poder utilizarlas, previamente debe crear los bloques de datos y declarar los datos dentro de ellos. En caso de tratarse de bloques de datos de instancia, se deducirán de los bloques de función y se generarán automáticamente.

Tipos de acceso

A los datos globales puede accederse de las siguientes formas:

- **acceso absoluto:** mediante identificador de operando y dirección absoluta.
- **acceso simbólico:** mediante un símbolo que previamente ha definido en la tabla de símbolos (v. **/231/**).
- **acceso indizado:** mediante identificador de operando e índice de array.
- **acceso estructurado:** mediante una variable.

Tabla 12-1 Utilización de los tipos de acceso a datos globales

Tipo de acceso	Areas de memoria de la CPU	Datos de usuario globales
absoluto	sí	sí
simbólico	sí	sí
indizado	sí	sí
estructurado	no	sí

12.2 Areas de memoria de la CPU

Definición

Las áreas de memoria de la CPU son áreas declaradas por el sistema. Por ello, no es necesario que declare en su bloque lógico estas áreas.

Diferentes áreas de memoria

Cada CPU pone a su disposición las siguientes áreas de memoria con su propio espacio de direccionamiento:

- Entradas/salidas en imagen del proceso
- Entradas/salidas de periferia
- Marcas
- Temporizadores, contadores (v. cap. 17)

Sintaxis para el acceso

A un área de memoria de la CPU se accede en una asignación de valor dentro, del área de instrucciones de un bloque lógico (v. apt. 14.3):

- mediante acceso simple que puede especificar absoluta o simbólicamente, o
- mediante acceso indizado.

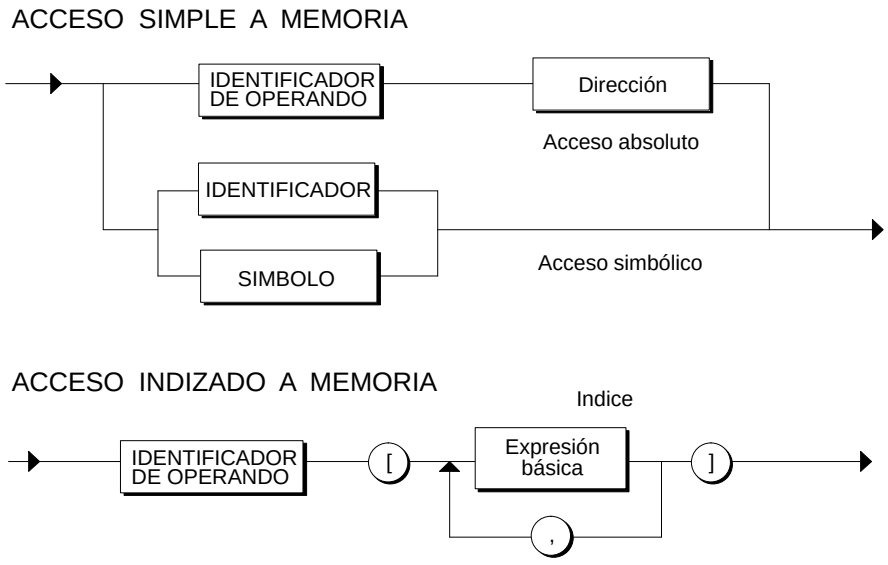


Figura 12-1 *Sintaxis: Acceso simple y acceso indizado a memoria*

12.3 Acceso absoluto a áreas de memoria

Principio

El acceso absoluto a un área de memoria se realiza, p.ej., mediante una asignación de valor a una variable del mismo tipo:

```
ESTADO_2:= EB10;
```

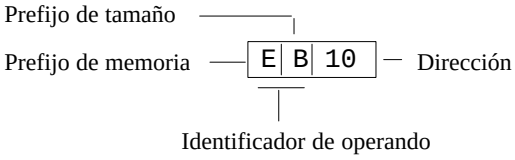
Acceso absoluto

Variable del mismo tipo

El identificador absoluto remite a un área de memoria dentro de la CPU. Puede especificar este área indicando el identificador de operando (en este caso `EB`) seguido de la dirección (en este caso `10`).

Identificador absoluto

El identificador absoluto se compone del identificador de operando con prefijo de memoria y tamaño y una dirección.



Identificador de operando

La combinación de prefijo de memoria y prefijo de tamaño compone el identificador de operando.

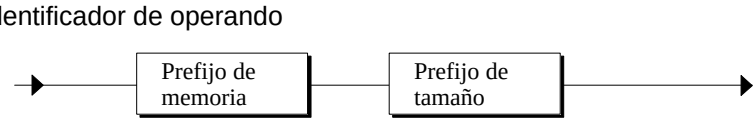


Figura 12-2 *Sintaxis: Identificador de operando*

Prefijo de memoria

Con el prefijo de memoria puede determinar el tipo de áreas de memoria. Como muestra la figura 12-3, dispone de los siguientes tipos: [1)]

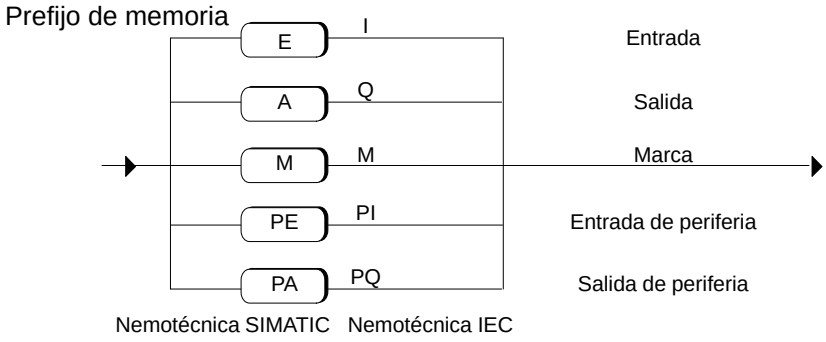


Figura 12-3 *Sintaxis: Prefijo de memoria*

1) *Dependiendo del lenguaje seleccionado (ajuste del compilador), los identificadores de operando en alemán, o en código internacional tienen un significado reservado.*

Prefijo de tamaño

Al introducir el prefijo de tamaño especifica el tamaño o el tipo del área de memoria que debe leerse desde la periferia; p.ej., un byte o una palabra. La especificación del prefijo de tamaño es opcional si está especificando un byte. La figura 12-4 muestra la sintaxis:

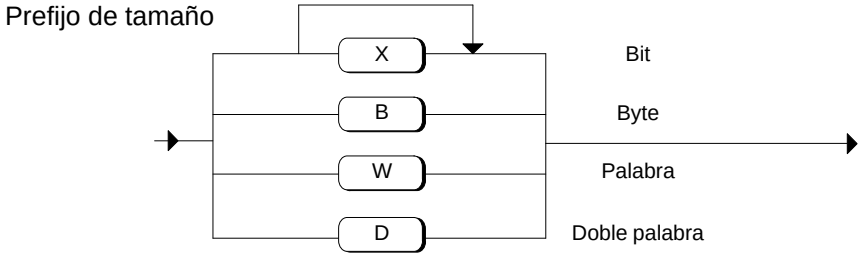


Figura 12-4 *Sintaxis: Prefijo de tamaño*

Dirección

Para especificar la dirección como muestra la figura 12–5, introduzca, dependiendo del prefijo de tamaño que haya utilizado, una dirección absoluta cuya meta puede ser un bit, un byte, una palabra o una doble palabra. Sólo si ha especificado ”bit” puede indicar una dirección de bit complementaria (v. fig. 12-5). El primer número significa la dirección del byte, y el segundo la dirección del bit.

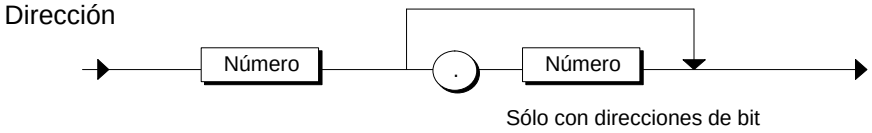


Figura 12-5 *Sintaxis: Dirección*

Ejemplos

Algunos ejemplos de acceso absoluto:

```
BYTE_ESTADO       := EB10;
ESTADO3           := E1.1;
Valor_medido      := EW20;
```

Ejemplo 12-1 Acceso absoluto

12.4 Acceso simbólico a áreas de memoria de la CPU

Principio

En la programación simbólica, en lugar del acceso absoluto con identificadores de operando y dirección, utilice un nombre simbólico para el direccionamiento de un área de memoria de la CPU:

Símbolo	Operando	Tipo de dato	Comentario
Contacto_motor	E 1.7	BOOL	Contactor 1 para Motor A 1
Entrada	EW 10	INT	Palabra de estado de entrada
Byte_entrada1	EB 1	BYTE	Byte de entrada
″Entrada 1.1″	E 1.1	BOOL	Barrera óptica
Canales_medicion	MW 2	WORD	Búfer de valores de medida
	MD4	REAL	Búfer

Para asignar el nombre simbólico a cada uno de los operandos de su programa de usuario, debe crear una tabla de símbolos.

Pueden especificarse todos los tipos de datos simples siempre que puedan recibir el ancho del acceso.

Acceso

Se realiza, por ejemplo, mediante una asignación de valor a una variable del mismo tipo, utilizando los símbolos declarados.

VALOR_MEDIDO1 := Contacto_motor;

Si se ha declarado un tipo de datos en la tabla de símbolos sin asignarle ningún nombre simbólico, este tipo de datos tiene efecto de todos modos al utilizar el operando en el programa.

Valor_Real := %MD4;

Creación de la tabla de símbolos

La tabla de símbolos se crea con STEP 7, al igual que la introducción de valores en la tabla de símbolos.

Puede arrancar la tabla de símbolos con el Administrador SIMATIC o directamente con SCL mediante el comando de menú **Herramientas ▸ Tabla de símbolos**.

Además, es posible exportar y procesar las tablas de símbolos que existan en forma de ficheros de texto, y que puedan crearse con cualquier editor de texto (para información al respecto, ver /**231**/).

Ejemplos

Algunos ejemplos de acceso simbólico:

```
BYTE_ESTADO       := Byte de entrada;
ESTADO3           := ″Entrada 1.1″;
Valor_medido      := Canales de medición;
```

Ejemplo 12-2 Acceso simbólico

12.5 Acceso indizado a áreas de memoria de la CPU

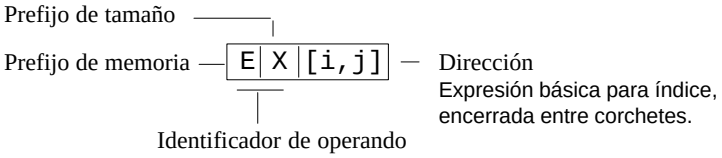
Principio

También tiene la posibilidad de acceder a áreas de memoria de una CPU con acceso **indizado**. Con respecto al acceso absoluto, tiene la ventaja de que puede direccionar dinámicamente los accesos utilizando índices variables. Por ejemplo, puede utilizar la variable en ejecución de un bucle FOR para el indizado.

El acceso indizado a un área de memoria se realiza de forma similar al acceso absoluto. Sólo se diferencia en la forma de especificar la dirección. En lugar de la dirección se especifica un índice que puede ser una constante, una variable o una expresión aritmética.

Identificador absoluto

En el acceso indizado el identificador absoluto se compone del identificador de operando (v. apt. 12.3) y de una expresión básica para el indizado.



Reglas para el acceso indizado

El indizado debe seguir las siguientes reglas:

- En un acceso a tipo de dato BYTE, WORD o DWORD, debe utilizar exactamente un índice. El índice se interpreta como la dirección del byte. La anchura del acceso viene determinada por el prefijo de tamaño.
- En un acceso con tipo de datos BOOL debe utilizar dos índices. El primer índice especifica la dirección del byte, y el segundo índice la posición del bit dentro del byte.
- Cada índice debe ser una expresión aritmética del tipo de dato `INT`.

```
VALOR_MEDIDO1     := EW[CONTADOR];

MARCA_SALIDA      := E[NUM_BYTE, NUM_BIT];
```

Ejemplo 12-3 Acceso indizado

12.6 Bloques de datos

Resumen

Dentro de los bloques de datos puede guardar y procesar para su aplicación todos los datos cuyo rango de validez se aplica a todo el programa o a todo el proyecto. Todos los bloques lógicos pueden acceder a ellos con derecho a lectura o escritura.

Declaración

En el capítulo 8 puede consultar la sintaxis para la estructura de los bloques de datos. Debe distinguir entre los dos tipos de bloques de datos siguientes:

- bloques de datos,
- bloques de datos de instancia.

Acceso a bloques de datos

A los datos de un bloque de datos cualquiera puede accederse siempre de las formas siguientes:

- acceso simple o absoluto,
- acceso indizado,
- acceso estructurado.

La figura 12-6 muestra un resumen de los diferentes tipos de acceso:

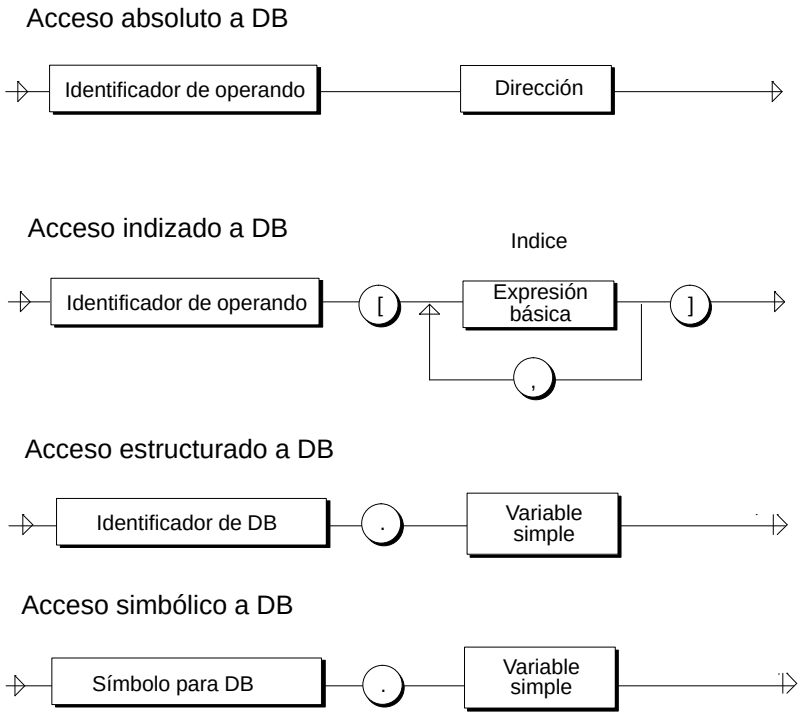
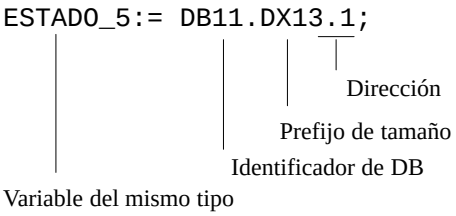


Figura 12-6 *Sintaxis: Acceso absoluto, acceso indizado y acceso estructurado a DB*

12.7 Acceso absoluto a bloques de datos

Principio

Al igual que el acceso a las áreas de memoria de la CPU, el acceso absoluto se realiza asignando un valor a una variable del mismo tipo. Después de especificar el identificador de DB sigue la palabra clave `'D'` junto con especificación del prefijo de tamaño (p.ej.: `W` para `WORD`) y la dirección del byte (p.ej.: `13.1`).



Acceso

Puede especificar el acceso según la figura 12-7, indicando el identificador de DB junto con el prefijo de tamaño y la dirección.

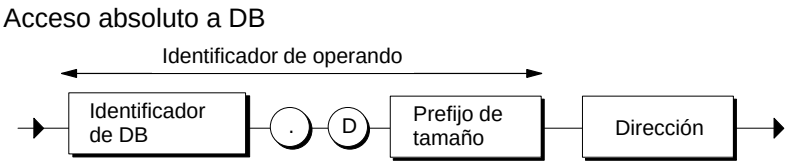


Figura 12-7 *Sintaxis: Acceso absoluto a DB*

Prefijo de tamaño

Indica la longitud del área de memoria dentro del bloque de datos al que debe accederse (p.ej.: un byte o una palabra). La especificación del prefijo de tamaño es opcional si desea especificar un bit. La figura 12-8 muestra la sintaxis para el prefijo de tamaño.

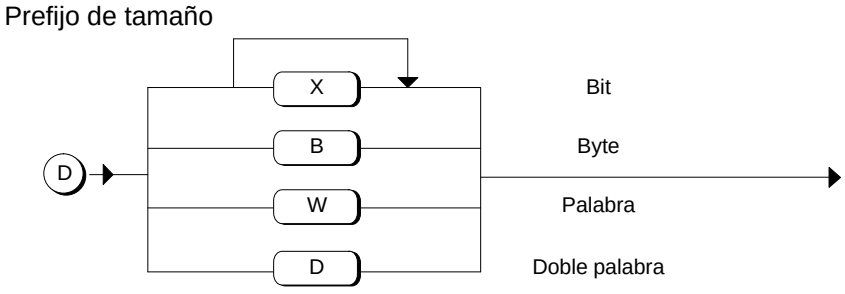


Figura 12-8 *Sintaxis: Prefijo de tamaño*

Dirección

Para especificar la dirección conforme a la figura 12-9, indique una dirección absoluta cuyo destino sea un bit, un byte, una palabra o una doble palabra, dependiendo del prefijo de tamaño que haya utilizado. Sólo en el caso de que haya especificado ”bit” puede indicar una dirección de bit complementaria. El primer número significa la dirección del byte, y el segundo la dirección del bit.

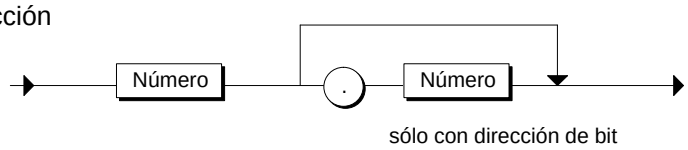


Figura 12-9 *Sintaxis: Dirección*

Ejemplos

Algunos ejemplos de acceso absoluto a un bloque de datos. El mismo bloque de datos se especifica de modo absoluto en la primera parte, y simbólicamente en la segunda:

```
BYTE_ESTADO       := DB101.DB10;
ESTADO_3          := DB30.D1.1;
Valor_medido      := DB25.DW20;

BYTE_ESTADO       := Datos de estado.DB10;
ESTADO_3          := ”Datos nuevos”.D1.1;
Valor_medido      := Datos medidos.DW20;

ESTADO_1          := WORD_TO_BLOCK_DB(INDICE).DW10;
```

Ejemplo 12-4 Acceso absoluto

12.8 Acceso indizado a bloques de datos

Acceso indizado

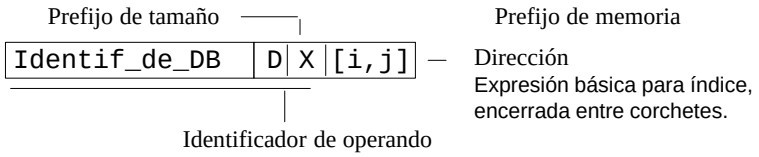
También tiene la posibilidad de acceder a bloques de datos globales de forma indizada. Frente al direccionamiento absoluto tiene la ventaja de que puede efectuar un direccionamiento dinámico utilizando índices variables. Por ejemplo, puede utilizar la variable en ejecución de un bucle FOR para el indizado.

El acceso indizado a un bloque de datos se realiza de forma similar al acceso absoluto. Sólo se diferencia en la forma de especificar la dirección.

En lugar de la dirección se especifica un índice que puede ser una constante, una variable o una expresión aritmética.

Identificador absoluto

En el acceso indizado el identificador absoluto se compone del identificador de operando (v. apt. 12.7) y de una expresión básica para el indizado.



Reglas para el acceso indizado

El indizado debe seguir las siguientes reglas:

- Cada índice debe ser una expresión aritmética del tipo de dato `INT`.
- En un acceso a tipo de dato `BYTE`, `WORD` o `DWORD`, debe utilizar exactamente un índice. El índice se interpreta como la dirección del byte. La anchura del acceso viene determinada por el prefijo de tamaño.
- En un acceso con tipo de dato `BOOL` debe utilizar dos índices. El primer índice especifica la dirección del byte, y el segundo índice la posición del bit dentro del byte.

```
ESTADO_1:= DB11.DW[CONTADOR];
ESTADO_2:= DB12.DX[WNR, NUM_BIT];

ESTADO_1:= Base de datos1.DW[CONTADOR];
ESTADO_2:= Base de datos2.DX[WNR, NUM_BIT];

ESTADO_1:= WORD_TO_BLOCK_DB(INDICE).DW[CONTADOR];
```

Ejemplo 12-5 Acceso indizado

12.9 Acceso estructurado a bloque de datos

Principio

El acceso estructurado se realiza asignando valores a una variable del mismo tipo.

```
TIME_1:= DB11.HORA_DEL_DIA ;
```

Variable simple

Identificación de DB

Variable del mismo tipo

Referencie la variable del bloque de datos indicando el nombre del DB y, separado por un punto, el nombre de la variable simple. Las sintaxis respectiva se describe en la figura 12-6.

La variable simple representa una variable (sencilla) que usted ha asignado en la declaración a un tipo de datos simple o compuesto.

Ejemplos

```
En la tabla de declaración del FB10:
VAR
Resultado:  STRUCT RES1 : INT;
                   RES2 : WORD;
            END_STRUCT
END_VAR
```

```
Tipo de datos de usuario UDT1:
TYPE UDT1   STRUCT RES1 : INT;
                   RES2 : WORD;
            END_STRUCT
```

```
DB20 con tipo de datos de usuario:
DB20
UDT1
BEGIN ...
```

```
DB30 sin tipo de datos de usuario:
DB30        STRUCT RES1 : INT;
                   RES2 : WORD;
            END_STRUCT
BEGIN ...
```

Ejemplo 12-6 Declaración de datos para los bloques de datos

```
Bloque de función con los accesos:
..
FB10.DB10();
PAL_RES_A   :=    DB10.Resultado.RES2;
PAL_RES_B   :=    DB20.RES2;
PAL_RES_C   :=    DB30.RES2;
```

Ejemplo 12-7 Acceso a los datos de los bloques de datos

Expresiones, operadores y operandos

13

Resumen

Una expresión representa un valor que se calcula durante el tiempo de compilación o durante el tiempo de ejecución del programa. Está compuesta de operandos (p.ej., constantes, variables o valores de función) y operadores (p.ej.: *, /, +, -).

Los tipos de datos de los operandos y de los operadores que intervienen determinan el tipo de la expresión. SCL distingue los siguientes:

- expresiones aritméticas
- expresiones de potencias
- expresiones de comparación
- expresiones lógicas

Indice del capítulo

13.1 Operadores

Resumen

Las expresiones están formadas por operadores y operandos. La mayoría de los operadores de SCL relacionan dos operandos y, por lo tanto, se denominan *binarios*. Los restantes trabajan con un solo operando, y por ello se llaman *unarios*.

Los operadores binarios se escriben entre los operandos (p.ej., A+B). Un operador unario se sitúa siempre inmediatamente delante del operando (p.ej., -B).

La prioridad entre los operadores que se muestra en la tabla 13-1 regula el orden de cálculo. En la tabla la cifra ”1” corresponde a la máxima prioridad.

Clases de operadores

Tabla 13-1 Resumen de operadores

Clase	Operador	Representación	Prioridad
Operador de asignación *Este operador deposita un valor en una variable*	Asignación	: =	11
Operadores aritméticos *Estos operadores se necesitan para cálculos matemáticos*	Potencia	**	2
	Operadores unarios		
	Más unario	+	3
	Menos unario	-	3
	Operadores aritméticos básicos		
	Multiplicación	*	4
	Función de módulo	MOD	4
	División por núm. enteros	DIV	4
	Adición	+	5
	Sustracción	-	5
Operadores de comparación *Estos operadores se necesitan para poder formular condiciones.*	Menor que	<	6
	Mayor que	>	6
	Menor o igual que	<=	6
	Mayor o igual que	>=	6
	Igualdad	=	7
	Desigualdad	<>	7
Operadores lógicos *Estos operadores se necesitan para las expresiones lógicas.*	Negación (unaria)	NOT	3
	Operadores lógicos básicos		
	Y	AND o &	8
	O exclusiva	XOR	9
	O	OR	10
Paréntesis	(**Expresión**)	( )	1

13.2 Sintaxis de expresiones

Resumen

Las expresiones pueden representarse por el diagrama sintáctico de la figura 13-1. Las expresiones aritméticas lógicas y de comparación y la expresión de potencias presentan algunas particularidades, por lo que se tratarán por separado en los apartados 13.3 a 13.6.

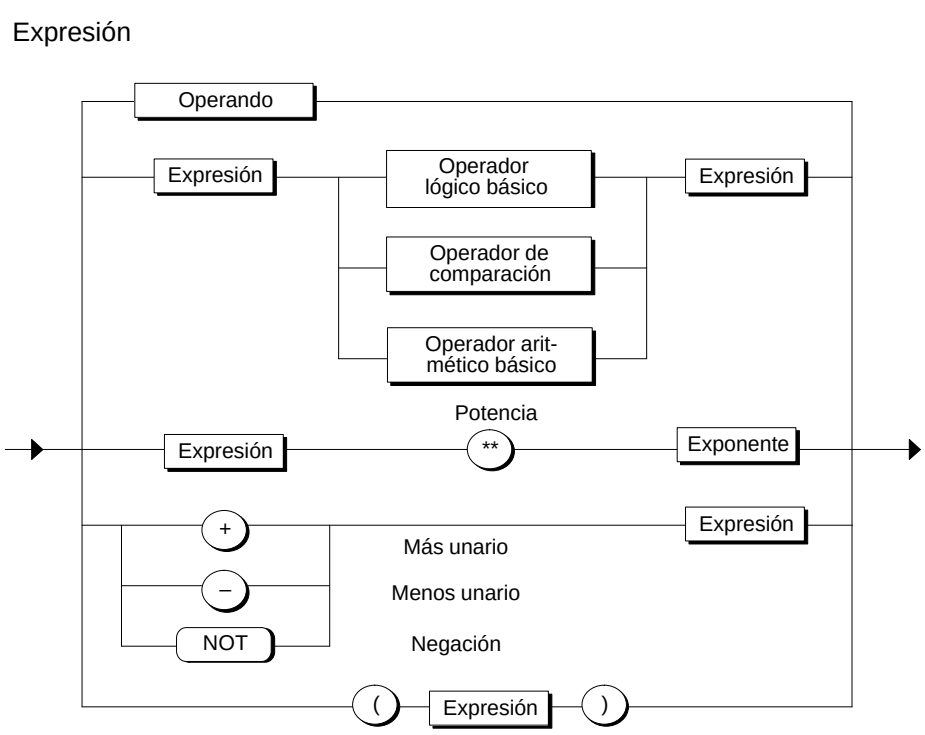


Figura 13-1 *Sintaxis: Expresión*

Resultado de una expresión

El resultado de una expresión puede:

- asignarlo a una variable,
- utilizarlo como condición para una instrucción de control,
- utilizarlo como parámetro para llamar a una función o a un bloque de función.

Orden de valoración

El orden de valoración de una expresión es determinado por:

- la prioridad de los operadores involucrados,
- el orden de izquierda a derecha,
- los paréntesis utilizados (tratándose de operadores de igual prioridad).

Reglas

Las expresiones se procesan según determinadas reglas:

- Los operadores se procesan siguiendo su orden jerárquico (prioridad).
- Los operadores de igual prioridad se procesan de izquierda a derecha.
- Anteponer un signo menos a un identificador equivale a multiplicar por -1.
- No puede haber dos operadores aritméticos consecutivos. Por esta razón no está permitida la expresión a* -b, pero es válida la expresión a* (-b).
- La colocación de parejas de paréntesis puede anular la prioridad de los operadores; es decir, la colocación de paréntesis tiene prioridad máxima.
- Las expresiones dentro de paréntesis se consideran operadores únicos y se valoran siempre en primer lugar.
- El número de paréntesis situados a la izquierda debe coincidir con el de situados a la derecha.
- Las operaciones aritméticas no pueden aplicarse a caracteres o datos lógicos. Por ello son erróneas las expresiones 'A'+'B' y (n<=0) + (n<0).

Ejemplos

Las diferentes expresiones podrían tener la siguiente estructura:

```
EB10                      // Operando
A1 AND (A2)               // Expresión lógica
(A3) < (A4)               // Expresión de comparación
3+3*4/2                   // Expresión aritmética

VALOR_MEDIDO**2           // Expresión de potencia
(DIFERENCIA)**DB10.EXPONENTE // Expresión de potencia
(SUMA)**FC100(..)         // Expresión de potencia
```

Ejemplo 13-1 Expresiones diversas

13.2.1 Operandos

Resumen

Los operandos son objetos que permiten formar expresiones. Los operandos pueden representarse por el diagrama sintáctico de la figura 13-2.

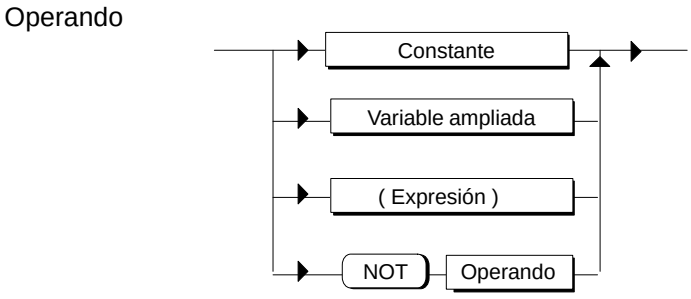


Figura 13-2 *Sintaxis: Operando*

Constantes

Pueden utilizarse constantes con su valor numérico o con un nombre simbólico o una secuencia de caracteres.

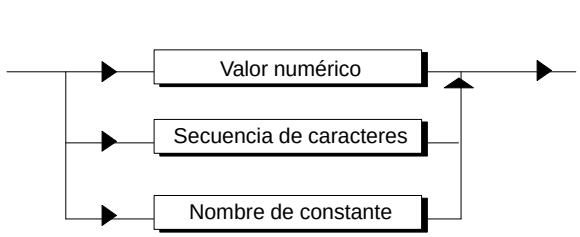


Figura 13-3 *Sintaxis: Constante*

Los siguientes ejemplos son ejemplos de constantes válidas:

```
4_711
4711
30.0
'CARACTER'
FACTOR
```

Variables ampliadas

La variable ampliada es un concepto genérico que engloba a una serie variables que se tratarán con más exactitud en el capítulo 14.

Variable ampliada

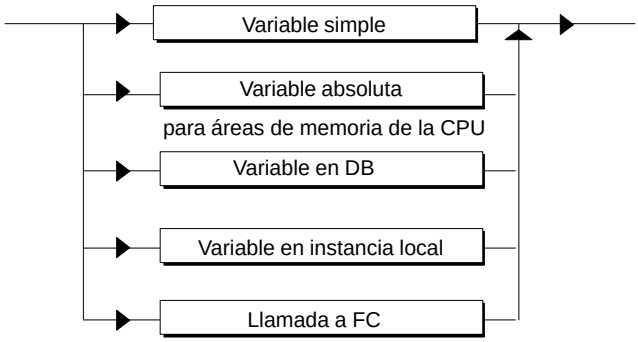


Figura 13-4 *Sintaxis: Variable ampliada*

Ejemplos de variables ampliadas

Algunos ejemplos de variables ampliadas son los siguientes:

```
VALOR_TEORICO        Variable simple
EW10                 Variable absoluta
E100.5               Variable absoluta
DB100.DW[INDICE]     Variable en DB
MOTOR.NUM_REV        Variable en instancia local
SQR(20)              Función estándar
FC192(VALOR_TEORICO) Llamada a función
```

Ejemplo 13-2 Variables ampliadas en expresiones

Nota

En una llamada a función el resultado calculado (valor de respuesta) se inserta en la expresión en lugar del nombre de la función. Por esta razón **no** están permitidos en una expresión las funciones VOID actuando como operandos, puesto que no tienen valor de respuesta.

13.3 Expresiones aritméticas

Definición

Expresiones aritméticas son expresiones formadas con operadores aritméticos. Dichas expresiones permiten el procesamiento de tipos de datos numéricos.

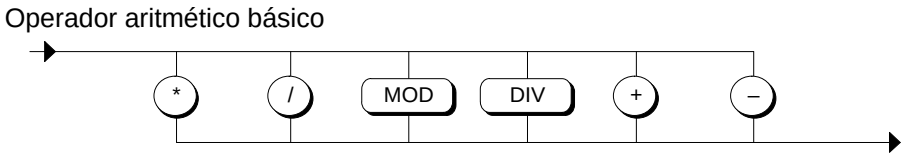


Figura 13-5 *Sintaxis: Operador aritmético básico*

Operaciones aritméticas

La tabla 13-2 resume las posibles operaciones y muestra el tipo que debe asignarse al resultado en función de los operandos.
Se utilizan las abreviaturas

`ANY_INT` para los tipos de datos `INT, DINT`
`ANY_NUM` para los tipos de datos `ANY_INT` y `REAL`

Tabla 13-2 Operadores aritméticos

Operación	Operador	1º operando	2º operando	Resultado [1]	Prioridad
Potencia	**	ANY_NUM	INT	REAL	2
Más unario	+	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
Menos unario	-	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
Multiplicación	*	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	
División	/	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	
División por enteros	DIV	ANY_INT	ANY_INT	ANY_INT	4
		TIME	ANY_INT	TIME	
División por módulo	MOD	ANY_INT	ANY_INT	ANY_INT	4
Adición	+	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DT	TIME	DT	
Sustración	-	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DATE	DATE	TIME	
		TOD	TOD	TIME	
		DT	TIME	DT	
		DT	DT	TIME	

[1] *Tenga en cuenta que el tipo correspondiente al resultado viene determinado por el tipo del operando de mayor rango.*

Reglas

Dentro de una expresión aritmética los operadores se procesan siguiendo su orden de prioridades, como se indica en la tabla 13-2.

- Para facilitar la lectura de los paréntesis se recomienda poner los signos de los números negativos incluso donde no sea necesario.
- En divisiones con dos operandos enteros del tipo de datos INT, los operadores ”DIV” y ”/” arrojan el mismo resultado (véase ejemplo 13-3).
- Los operadores de división (’/’, ’MOD’ y ’DIV’) exigen que el segundo operando sea distinto de cero.
- Cuando un operando es del tipo INT (entero) y otro del tipo REAL (número de coma flotante), el resultado siempre será del tipo REAL.

Ejemplos

Los siguientes ejemplos ilustran la formación de expresiones aritméticas.

Supongamos que i y j son variables enteras cuyos valores son 11 o -3. En el ejemplo 13-3 se muestran algunas expresiones enteras y sus correspondientes valores.

Expresión	Valor
i + j	8
i – j	14
i * j	–33
i DIV j	–3
i MOD j	2
i/j	–3

Ejemplo 13-3 Expresiones aritméticas

Supongamos que i y j son variables enteras cuyos valores son 3 o -5. Entonces, según el ejemplo 13-4, el resultado de la expresión aritmética (el valor entero 7) se asignará a la variable VALOR.

```
VALOR:= i + i * 4 / 2 - (7+i) / (-j) ;
```

Ejemplo 13-4 Expresión aritmética

13.4 Exponentes

Resumen

La figura 13-6 pretende ilustrar la formación del exponente en una expresión de potencia (ver apt. 13.2). Tenga en cuenta que el exponente, como caso especial, también puede formarse con variables ampliadas.

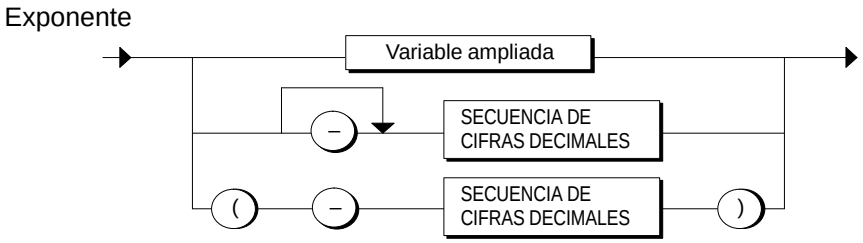


Figura 13-6 *Sintaxis: Exponente*

```
VALOR_MEDIDO**2         // Expresión de potencia
(DIFERENCIA)**DB10.EXPONENTE  // Expresión de potencia
(SUMA)**FC100(..)       // Expresión de potencia
```

Ejemplo 13-5 Expresiones de potencia con diversos exponentes

13.5 Expresiones de comparación

Definición

Una expresión de comparación es una expresión de tipo BOOL formada con operadores de comparación. Estas expresiones se forman combinando operadores del mismo tipo (de cualquier tipo, excepto las lógicas) con los operadores relacionados en la figura 13-7.

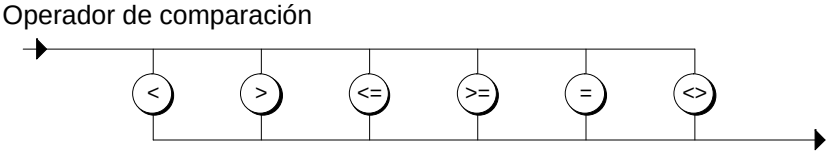


Figura 13-7 *Sintaxis: Operador de comparación*

Operaciones de comparación

Los operadores de comparación comparan sus dos operandos con su valor numérico.

```
1º Operando Operador 2º Operando ⇒ Valor booleano
```

Como resultado se obtiene un valor que puede ser verdadero o falso. El valor es ”verdadero” (TRUE) en caso de que la comparación se cumpla; y ”falso” (FALSE) cuando no se cumple.

Reglas

Al formar expresiones de comparación deben tenerse en cuenta las siguientes reglas:

- Para evitar falta de claridad en el orden en que deben efectuarse las operaciones lógicas, los operandos lógicos deben estar encerrados entre paréntesis.
- Las expresiones lógicas pueden enlazarse aplicando la ley de la lógica booleana, para ejecutar consultas como ”si a<b **y** b<c, entonces...”. Como expresión pueden utilizarse variables o constantes del tipo BOOL y expresiones de comparación.
- Los siguientes tiempos sólo permiten comparar variables de todo tipo:
  - `INT, DINT, REAL`
  - `BOOL, BYTE, WORD, DWORD`
  - `CHAR, STRING`
- Los siguientes tiempos permiten comparar variables del mismo tipo:
  - `DATE, TIME, TOD, DT`
- Al comparar caracteres (tipo `CHAR`) la valoración se efectúa siguiendo el orden de los caracteres ASCII.
- Las variables de tiempo S5 no son comparables.
- Cuando los dos operandos son del tipo DT o `STRING` deben compararse con las correspondientes funciones IEC.

Ejemplos

Los siguientes ejemplos ilustran la formación de las expresiones de comparación:

```
// Se niega el resultado de la expresión de
// comparación.

      IF NOT (CONTADOR > 5) THEN
            //...
            //...
      END_IF;


// Se niega el resultado de la primera expresión
// de comparación y se conjuga con el resultado
// de la segunda.

      A:= NOT (CONTADOR1 = 4) AND (CONTADOR2 = 10);


// Disyunción de dos expresiones de comparación
      WHILE (A >= 9) OR (CONSULTA <> 'n') DO
            //...
            //...
      END_WHILE;
```

Ejemplo 13-6 Expresiones lógicas

13.6 Expresiones lógicas

Definición

Una expresión lógica es una expresión formada con operadores lógicos. Los operadores lógicos (AND, &, XOR, OR) pueden enlazar operandos lógicos (tipo BOOL) o variables del tipo de dato BYTE, WORD o DWORD, para formar expresiones lógicas. Para negar el valor de un operando lógico (es decir, invertirlo) se utiliza el operador NOT.

Operador lógico básico

¡NOT no es un operador básico!
El operador actúa como un signo.

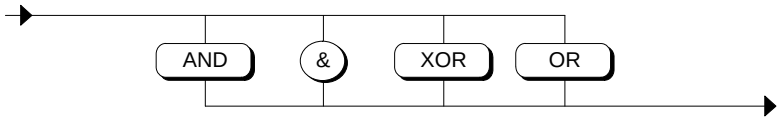


Figura 13-8 *Sintaxis: Operador lógico básico*

Operaciones lógicas

La tabla 13-3 ofrece información sobre las expresiones lógicas disponibles y los tipos de datos para los resultados y operandos. Se utilizan las abreviaturas:

ANY_BIT para los tipos de datos BOOL, BYTE, WORD, DWORD

Tabla 13-3 Operadores lógicos

Operación	Operador	Primer operando	Segundo operando	Resultado	Prioridad
Negación	NOT	ANY_BIT	-	ANY_BIT	3
Conjunción	AND	ANY_BIT	ANY_BIT	ANY_BIT	8
Disyunción exclusiva	XOR	ANY_BIT	ANY_BIT	ANY_BIT	9
Disyunción	OR	ANY_BIT	ANY_BIT	ANY_BIT	10

Resultados

El resultado de una expresión lógica es:

- 1 (*true*) o = 0 (*false*) al relacionar lógicamente operaciones booleanas, o
- una plantilla de bits, después de relacionar bit a bit los dos operandos.

Ejemplos

Supongamos que `'n'` es una variable entera con el valor 10 y `'s'` una variable de caracteres que representa el carácter A. Algunas expresiones lógicas con estas variables son las siguientes:

```
Expresión                    Valor

( n>0 )   AND  ( n<20)       verdadero
( n>0 )   AND  ( n<5 )       falso
( n>0 )   OR   ( n<5 )       verdadero
( n>0 )   XOR  ( n<20)       falso
( n=10 )  AND  ( s='A')      verdadero
( n<>5 )  OR   ( s>='A')     verdadero
```

Ejemplo 13-7 Expresiones lógicas

Asignación de valores

14

Resumen

Una asignación de valor sirve para asignar a una variable el valor de una expresión, sobreescribiéndose el valor anterior.

Indice del capítulo

Información adicional

En SCL existen instrucciones simples y estructuradas. Entre las instrucciones simples se encuentran la asignación de valor, la instrucción de llamada y la instrucción GOTO (Ir a). Más información al respecto encontrará en los capítulos 15 y 16.

Las instrucciones de control para una ramificación del programa y para un procesamiento de bucles se cuentan entre las instrucciones estructuradas. Encontrará explicaciones en el capítulo 15.

14.1 Resumen

Principio

Las asignaciones de valores sustituyen el valor instantáneo de una variable por un nuevo valor especificado mediante una expresión. Esta expresión también puede incluir identificadores de funciones (FC) que se activan y proporcionan como respuesta los valores correspondientes (valor de respuesta).

Conforme al diagrama sintáctico siguiente, la expresión del lado derecho de la asignación de valor es valorada y el resultado se deposita en la variable cuyo nombre figura en el lado izquierdo del símbolo de asignación. La figura 14-1 muestra la totalidad de las variables permitidas.

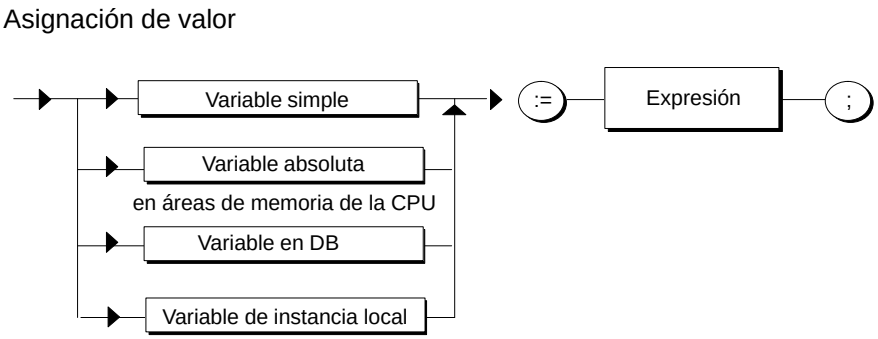


Figura 14-1 *Sintaxis: Asignación de valor*

Resultado

El resultado de una expresión de asignación es del mismo tipo que el operando de la izquierda.

14.2 Asignación de valores con variables de un tipo simple

Asignación

Toda expresión y toda variable de tipo de datos simple puede asignarse a otra variable del mismo tipo.

```
Identificador: = Expresión;
Identificador: = Variable de tipo de datos simple;
```

Ejemplos

Los siguientes ejemplos son asignaciones de valor válidas:

```
FUNCTION_BLOCK FB10
VAR
   INTERRUPTOR_1   : INT;
   INTERRUPTOR_2   : INT;
   VALOR_TEORICO_1 : REAL;
   VALOR_TEORICO_2 : REAL;
   CONSULTA_1      : BOOL;
   TIEMPO_1        : S5TIME;
   TIEMPO_2        : TIME;
   FECHA_1         : DATE;
   HORA_DEL_DIA_1  : TIME_OF_DAY;
END_VAR

BEGIN
// Asignación de una constante a una variable
   INTERRUPTOR_1   := -17;
   VALOR_TEORICO_1 := 100.1;
   CONSULTA_1      := TRUE;
   TIEMPO_1        := TIME#1H_20M_10S_30MS;
   TIEMPO_2        := TIME#2D_1H_20M_10S_30MS;
   FECHA_1         := DATE#1996-01-10;

// Asignación de una variable
   VALOR_TEORICO_1 := VALOR_TEORICO_2;
   INTERRUPTOR_2_ := INTERRUPTOR_1;

// Asignación de una expresión a una variable
   INTERRUPTOR_2  := INTERRUPTOR_1 * 3;
END_FUNCTION_BLOCK
```

Ejemplo 14-1 Asignación de valor con tipos de datos simples

14.3 Asignación de valores con variables del tipo STRUCT o UDT

Variable del tipo STRUCT o UDT

Las variables del tipo STRUCT o UDT son variables estructuradas que representan o una estructura completa o un componente de dicha estructura.

Son especificaciones válidas de una variable estructurada las siguientes:

```
Imagen          //Identificador de una estructura
Imagen.elemento //Identificador de un componente
                //de estructura
Imagen.array    //Identificador de un array simple
                //dentro de una estructura
Imagen.array[2,5]//Identificador de un componente de
                //array dentro de una estructura
```

Asignación de una estructura completa

Una estructura completa sólo puede asignarse a otra estructura cuando coincide tanto los componentes de las estructuras como sus tipos de datos y sus nombres. Una asignación válida es la siguiente:

```
nom_estruct_1:=nom_estruct_2;
```

Asignación de componentes de estructuras

A cualquier componente de estructura puede asignar una variable de tipo compatible, una expresión de tipo compatible u otro componente de estructura. Son asignaciones válidas las siguientes:

```
nom_estruct_1.elemento1 := Valor;
nom_estruct_1.elemento1 := 20.0;
nom_estruct_1.elemento1 := nom_estruct_2.elemento1;
nom_estruct_1.nom_array1 := nom_estruct_2.nom_array1;
nom_estruct_1.nom_array[10]:= 100;
```

Ejemplos

Con ayuda de los siguientes ejemplos vamos a ilustrar las asignaciones de valor para datos de estructuras:

```
FUNCTION_BLOCK FB10
VAR
      VAR_AUX     :REAL;
      VALOR_MEDIDO :STRUCT  //Estructura de destino
            TENSION     :REAL;
            RESISTENCIA :REAL;
            Array simple:ARRAY[1..2,1..2] OF INT;
      END_STRUCT;

      VALOR_REAL  :STRUCT  //Estructura fuente
            TENSION     :REAL;
            RESISTENCIA :REAL;
            Array simple:ARRAY[1..2,1..2] OF INT;
      END_STRUCT;
END_VAR
BEGIN
//Asignación de una estructura completa a una
//estructura completa
      VALOR_MEDIDO:= VALOR_REAL;

//Asignación de un componente de estructura a un
//componente de estructura
      VALOR_MEDIDO.TENSION:= VALOR_REAL.TENSION;

// Asignación de un componente de estructura a
// una variable del mismo tipo
      VAR_AUX:= VALOR_REAL.RESISTENCIA;

// Asignación de una constante a un componente
// de estructura
      VALOR_MEDIDO.RESISTENCIA:= 4.5;

// Asignación de una constante a un
// ARRAY simple
      VALOR_MEDIDO.ARRAY_SIMPLE[1,2]:= 4;
END_FUNCTION_BLOCK
```

Ejemplo 14-2 Asignación de valor con variables del tipo STRUCT o UDT

14.4 Asignación de valores con variables del tipo ARRAY

Variable ARRAY

Un array consta de un número de dimensiones comprendido entre 1 y 6 o por elementos que son todos del mismo tipo. Para la asignación de arrays a una variable existen dos variantes posibles:

Puede referenciar arrays **completos** o **partes** de arrays. Referencie un array completo indicando el nombre de variable de array.

```
nom_array_1
```

Un elemento de array individual se referencia con el nombre del array seguido de los valores de índice encerrados entre corchetes. Para cada dimensión se dispone de un índice, separado por comas y encerrado asimismo entre corchetes. Uno de los índices debe ser una expresión aritmética del tipo de datos `INT`.

```
nom_array_1[2]
nom_array_1[4,5]
```

Asignación de un array completo

Un array completo puede asignarse a otro array cuando coinciden tanto los tipos de datos de los componentes como de los límites del array (índice de array mínimo y máximos posibles). Son asignaciones válidas las siguientes:

```
nom_array_1:=nom_array_2;
```

Asignación de un componente de array

Obtendrá una asignación de valor para una parte de array admisible omitiendo los índices que se encuentran después del nombre del array dentro de los corchetes, ,comenzando por la derecha. De esta forma especifica un área parcial cuyo número de dimensiones es igual al número de índices omitidos.

De todo ello resulta que puede referenciar en una matriz filas y componentes individuales, pero no columnas cerradas (cerradas significa de ... hasta).
Son asignaciones válidas las siguientes:

```
nom_array_1[i]:=nom_array_2[j];
nom_array_1[i]:=expresion;
identificador_1 :=nom_array_1[i];
```

Ejemplos

Con ayuda de los ejemplos siguientes va a ilustrarse las asignaciones de valor para ARRAY:

```
FUNCTION_BLOCK FB3
VAR
   VALORES_TEORICOS:ARRAY [0..127] OF INT;
   VALORES_REALES  :ARRAY [0..127] OF INT;

// Declaración de una matriz
// (=array bidimensional)
// de 3 líneas y 4 columnas
      REGULADOR: ARRAY [1..3, 1..4] OF INT;

// Declaración de un vector
// (=array unidimensional)
// de 4 componentes
      REGULADOR_1: ARRAY [1..4] OF INT;
END_VAR

BEGIN
// Asignación de un ARRAY completo a un
// ARRAY
      VALORES_TEORICOS:= VALORES_REALES;

// Asignación de un vector a la segunda línea
// del array REGULADOR
      REGULADOR [2]:= REGULADOR_1;

//Asignación de un componente de ARRAY a un
//componente del array REGULADOR
      REGULADOR [1,4]:= REGULADOR_1 [4];
END_FUNCTION_BLOCK
```

Ejemplo 14-3 Asignación de valor para arrays

14.5 Asignación de valores con variables de tipo STRING

Variable STRING

Una variable del tipo de datos STRING incluye una cadena de caracteres de un máximo de 254 caracteres.

Asignación

Toda variable del tipo de datos STRING puede asignarse a otra variable del mismo tipo. Son asignaciones válidas las siguientes:

```
variable_string1 := literal_string ;
variable_string1 := variable_string2 ;
```

Ejemplo

Con ayuda de los ejemplos siguientes va a ilustrarse la asignación de valores con variables STRING:

```
FUNCTION_BLOCK FB3
VAR
   INDICACION_1    : STRING[50] ;
   STRUCT1         : STRUCT
      INDICACION_2 : STRING[100] ;
      INDICACION_3 : STRING[50] ;
      END_STRUCT;
END_VAR
BEGIN
// Asignación de una constante a una variable
// STRING
   INDICACION_1 := 'Error en módulo 1';
// Asignación de un componente de estructura
// a una variable STRING
   INDICACION_1 := STRUCT1.INDICACION_3;
// Asignación de una variable STRING a un
// componente de estructura STRING
   If INDICACION_1 <> STRUCT.INDICACION_3 THEN
      STRUCT.INDICACION_3 := INDICACION_1;
   END_IF;
END_FUNCTION_BLOCK
```

Ejemplo 14-4 Asignación de valor para variables STRING

14.6 Asignación de valores con variables del tipo DATE_AND_TIME

Variable DATE_AND_TIME

El tipo de datos DATE_AND_TIME define un área de 64 Bits (8 Bytes) para especificar la fecha y la hora del día.

Asignación

Todas las variables del tipo de datos DATE_AND_TIME pueden asignarse a otra variable o constante del mismo tipo. Son asignaciones válidas las siguientes:

```
variable_dt1 := Fecha_y_literal_de_hora_del_día;
variable_dt1 := variable_dt2;
```

Ejemplo

Con ayuda de los ejemplos siguientes pretende ilustrarse las asignaciones de valor con variables DATE_AND_TIME.

```
FUNCTION_BLOCK FB3
VAR
      TIEMPO_1    : DATE_AND_TIME;
      STRUCT1     : STRUCT
            TIEMPO_2 : DATE_AND_TIME;
            TIEMPO_3 : DATE_AND_TIME;
      END_STRUCT;
END_VAR

BEGIN
// Asignación de una constante a una variable
// DATE_AND_TIME
TIEMPO_1 := DATE_AND_TIME#1995-01-01-12:12:12.2;
STRUCT1.TIEMPO_3 := DT#1995-02-02-11:11:11;

// Asignación de un componente de estructura a
// una variable DATE_AND_TIME

TIEMPO_1 := STRUCT1.TIEMPO_2;


// Asignación de una variable DATE_AND_TIME
// a un componente de estructura DATE_AND_TIME
If TIEMPO_1 < STRUCT1.TIEMPO_3 THEN
   STRUCT1.TIEMPO_3 := TIEMPO_1;
END_IF;
END_FUNCTION_BLOCK
```

Ejemplo 14-5 Asignación de valor para variables DATE_AND_TIME

14.7 Asignación de valores con variables absolutas para áreas de memoria

Variable absoluta

La variable absoluta referencia las áreas de memoria válidas de una CPU. Al hacer una asignación de valor dispone de las tres posibilidades descritas en el capítulo 12 para referenciar dichas áreas.

Variable absoluta

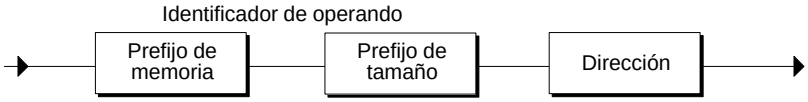


Figura 14-2 *Sintaxis: Variable absoluta*

Asignación

Con excepción de las entradas de periferia y las entradas mediante la imagen del proceso, usted puede asignar a todas las variables absolutas una variable o una expresión del mismo modo.

Ejemplo

Con ayuda de los ejemplos siguientes pretende ilustrarse las asignaciones de valor para identificadores absolutos:

```
FUNCTION_BLOCK FB10
VAR
      PALABRA_ESTADO1   : WORD;
      PALABRA_ESTADO2   : BOOL;
      PALABRA_ESTADO3   : BYTE;
      PALABRA_ESTADO4   : BOOL;
      DIRECCION   : INT:= 10;
END_VAR
BEGIN
    // Asignación de una palabra de entrada
    // a una variable (acceso simple)
      PALABRA_ESTADO1 := EW4 ;

    // Asignación de un bit de salida a una
    // variable (acceso simple)
      PALABRA_ESTADO2 := a1.1 ;

    // Asignación de un byte de entrada a una
    // variable (acceso indizado)
      PALABRA_ESTADO3 := EB[DIRECCION];

    // Asignación de un bit de entrada a una
    // variable (acceso indizado)
      FOR DIRECCION := 0 TO 7 BY 1 DO
            PALABRA_ESTADO4 := e[1,DIRECCION] ;
      END_FOR;
END_FUNCTION_BLOCK
```

Ejemplo 14-6 Asignación de valor para variables absolutas

14.8 Asignación de valores con variables globales

Variables en DB

El acceso a datos en los bloques de datos globales también se realiza por medio de una asignación de valor a variables del mismo tipo o viceversa. Dispone de las posibilidades de acceso estructurado, absoluto o indizado (v. cap. 12).

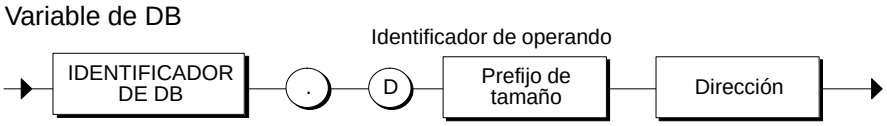


Figura 14-3 *Sintaxis: Variable de DB*

Asignación

A todas las variables globales puede asignar una variable o una expresión del mismo tipo. Son válidas las asignaciones siguientes:

```
DB11.DW10 := 20;
DB11.DW10 := Estado;
```

Ejemplos

El siguiente ejemplo presupone que en el bloque de datos DB11 se ha declarado una variable ”NUMERO” del tipo de datos ENTERO y una estructura ”NUMERO1” que tiene el componente ”NUMERO2” del tipo ENTERO.

```
// Bloque de datos DB11 necesario
DATA_BLOCK DB11
STRUCT
   NUMERO   :     INT  := 1;
   NUMERO1  :     STRUCT
            ZAHL2 :     INT := 256;
   END_STRUCT;
   PALABRA3 :     WORD := W#16#aa;
   PALABRA4 :     WORD := W#16#aa;
   PALABRA5 :     WORD := W#16#aa;
   PALABRA6 :     WORD := W#16#aa;
   PALABRA7 :     WORD := W#16#aa;
   PALABRA8 :     WORD := W#16#aa;
   PALABRA9 :     WORD := W#16#aa;
   PALABRA0:      WORD;
END_STRUCT

BEGIN
PALABRA10 := W#16#bb;
END_DATA_BLOCK
```

Ejemplo 14-7 Asignación de valor para variables globales

El bloque de datos DB11 puede utilizarse, por ejemplo, de la siguiente forma:

```
VAR
      REGULADOR_1   : ARRAY [1..4] OF INT;
      PALABRA_ESTADO1: WORD ;
      PALABRA_ESTADO2: ARRAY [1..10] OF INT;
      PALABRA_ESTADO3: INT
      DIRECCION   : INT ;
END_VAR
BEGIN
      // Asignar la palabra 10 del DB11 a una
      // variable (acceso simple)
      PALABRA_ESTADO1:= DB11.DW10
      // El primer componente del array se ha
      // asignado a la variable ”NUMERO” de DB11
      // (acceso estructurado)
      REGULADOR_1[1]:= DB11.NUMERO;
       // Asignar el componente estructurado
       // ”NUMERO2” de la estructura ”NUMERO1” a la
       // variable PALABRA_ESTADO3
      PALABRA_ESTADO3:= DB11.NUMERO1.NUMERO2
      // Asignar una palabra con índice DIRECCION
      // procedente del DB11 a una variable
      // (acceso indizado)
      FOR DIRECCION:= 1 TO 10 BY 1 DO
      PALABRA_ESTADO2[DIRECCION]:=DB11.DW[DIRECCION];
      END_FOR;
```

Ejemplo 14-8 Asignación de valor para variables globales de un DB

Instrucciones de control

15

Resumen

Son pocos los bloques que puede programar de forma que todas las instrucciones se ejecuten consecutivamente hasta el final del bloque. En el caso general ocurre que sólo se ejecutarán determinadas instrucciones (alternativas) dependiendo de determinadas condiciones, o que incluso se repitan múltiples veces (bucles). Los medios técnicos de que dispone el programa para ello son las instrucciones de control dentro del bloque SCL.

Indice del capítulo

15.1 Resumen

Instrucciones de selección

En los programas se presenta el problema de que dependiendo de determinadas condiciones deben ejecutarse diferentes instrucciones. Con una instrucción de selección tiene la posibilidad de ramificar el flujo del programa en un número de instrucciones sucesivas y alternativas comprendido entre 2 y n.

Tabla 15-1 Tipos de ramificaciones

Tipo de ramificación	Función
Instrucción `IF`	Con la instrucción IF puede ramificar el flujo del programa hacia una de dos alternativas, en función de una condición, que puede ser TRUE o FALSE.
Instruccíon `CASE`	Con una instrucción CASE puede controlar el flujo del programa en una ramificación 1:n, haciendo que una variable adopte un valor de entre n valores posibles.

Instrucciones de repetición

El procesamiento de bucles puede controlarlo con ayuda de las instrucciones de repetición. Una instrucción de repetición indica la parte de su programa que debe repetirse en función de determinadas condiciones.

Tabla 15-2 Tipos de instrucciones para procesamiento de bucles

Tipo de ramificación	Función
Instrucción `FOR`	Sirve para repetir una secuencia de instrucciones hasta que la variable en ejecución coincida dentro del rango indicado.
Instrucción `WHILE`	Sirve para repetir una secuencia de instrucciones hasta que se cumpla una condición de ejecución.
Instrucción `REPEAT`	Sirve para repetir una secuencia de instrucciones hasta que se cumpla una condición de interrupción.

Instrucciones de salto

Un salto de programa provoca el salto inmediato hasta una meta de salto especificada, y por lo tanto, hasta otra instrucción dentro del mismo bloque.

Tabla 15-3 Tipos de instrucciones de salto

Tipo de ramificación	Función
Instrucción `CONTINUE`	Sirve para interrumpir la ejecución del bucle que se está recorriendo en ese momento.
Instrucción `EXIT`	Sirve para abandonar un bucle en un punto cualquiera e independientemente de que se haya cumplido o no la condición de interrupción.
Instrucción `GOTO`	Provoca el salto inmediato hasta una meta de salto especificada.
Instrucción `RETURN`	Provoca que se abandone un bloque que se está editando actualmente.

Condiciones

La condición puede ser una expresión de comparación o una expresión lógica. La condición es del tipo BOOL y puede adoptar los dos valores siguientes: TRUE y/o FALSE.

Algunos ejemplos de **expresiones de comparación** válidas son los siguientes:

```
CONTADOR<=100
SQR(A)>0.005
Respuesta = 0
SALDO>=SUMA_Y_SIGUE
ch1< 'T'
```

Los siguientes son ejemplos de utilización de expresiones de comparación con operadores **lógicos**:

```
(CONTADOR<=100) AND(CH1<'*')
(SALDO<100.0) OR (ESTADO ='R')
(Respuesta<0)OR((Respuesta>5.0) AND (RESPUESTA<10.0))
```

Nota

Tenga en cuenta que los operandos lógicos (aquí, expresiones de comparación) figuran entre paréntesis para evitar cualquier ambigüedad sobre el orden de valoración.

15.2 Instrucción IF

Principio

La instrucción IF es una instrucción condicional. Ofrece una o varias opciones y selecciona una de ellas de su área de instrucciones (en algunos casos ninguna); la opción seleccionada se ejecutará.

Instrucción IF

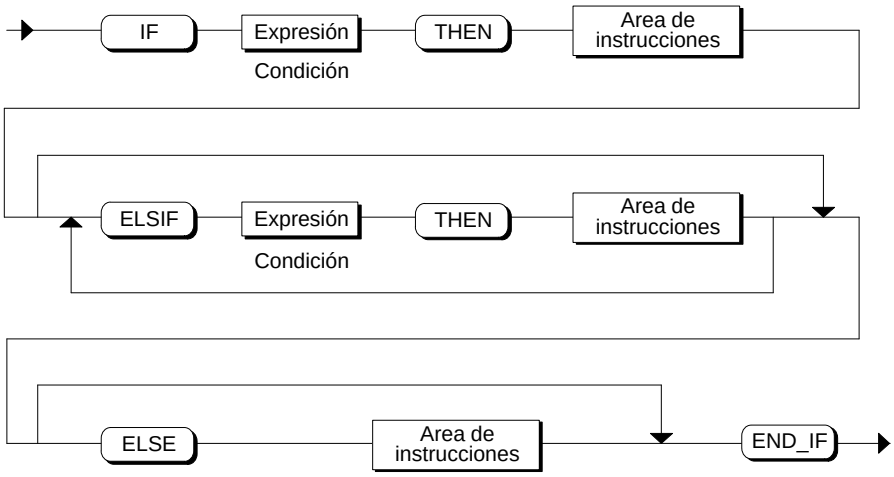


Figura 15-1 *Sintaxis: Instrucción IF*

La ejecución de la instrucción condicional provoca la valoración de las expresiones lógicas especificadas. Si el valor de una expresión es TRUE, la condición se considera cumplida; si el valor es FALSE, la condición no se ha cumplido.

Ejecución

La instrucción IF se ejecuta según las siguientes reglas:

1. Si el valor de la primera expresión es TRUE, después del primer THEN se ejecuta la siguiente parte de la instrucción; en caso contrario se valoran las expresiones de las ramas ELSIF.
2. En caso de que en las ramas ELSIF ninguna expresión booleana sea TRUE, se ejecuta la secuencia de instrucciones de ELSE (o ninguna secuencia de instrucciones,en caso de que la rama ELSE no exista).

Puede existir un número cualquier de instrucciones ELSIF.

Tenga en cuenta que pueden faltar las ramas ELSIF y/o la rama ELSE. Estos casos se tratarán como si estas ramas existieran con instrucciones en blanco.

Nota

Tenga en cuenta que la instrucción END_IF termina con un **punto y coma**.

Nota

Con respecto a una secuencia de instrucciones IF, la utilización de una o varias ramas ELSIF presenta la ventaja de que ya no se valoran las expresiones lógicas que siguen a una expresión válida. De esta forma se acorta el tiempo de ejecución de un programa.

Ejemplo

El ejemplo 15-1 ilustra el uso de la instrucción IF:

```
IF E1.1 THEN
      N:= 0;
      SUM:= 0;
      OK:= FALSE; // Colocar marca OK en FALSE
ELSIF START = TRUE THEN
      N:= N + 1;
      SUM:= SUM + N;
ELSE
      OK:= FALSE;
END_IF;
```

Ejemplo 15-1 Instrucción IF

15.3 Instrucción CASE

Principio

La instrucción CASE sirve para seleccionar 1 sección de programa de n posibles. Esta selección se basa en el valor en curso de una expresión de selección.

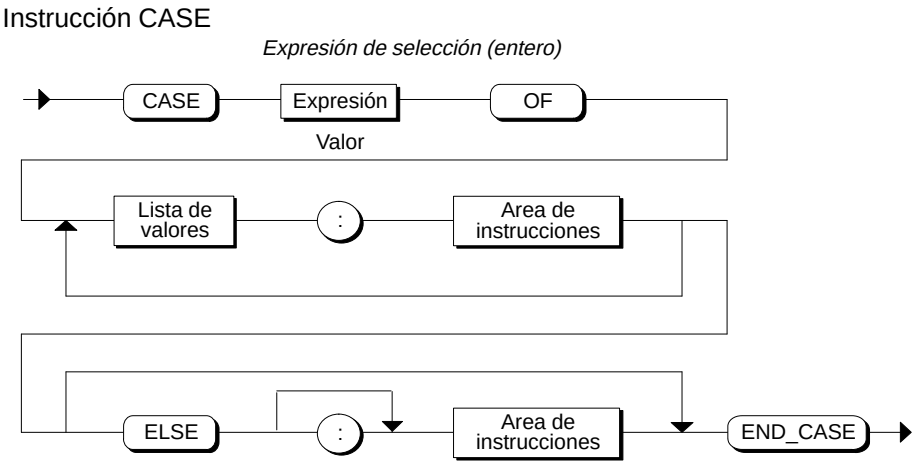


Figura 15-2 *Sintaxis: Instrucción CASE*

Ejecución

La instrucción CASE se ejecuta según las siguientes reglas:

1. Al ejecutar la instrucción `CASE` se comprueba si el valor de la expresión de selección se encuentra en una lista de valores especificada. Cada valor de esta lista representa uno de los valores permitidos para la expresión de selección. La expresión de selección debe proporcionar un valor del tipo INT (ENTERO).
2. En caso de coincidencia se ejecuta el área de instrucciones asignada de la lista.
3. La rama ELSE es opcional. Se ejecuta si el proceso de comparación no arroja ninguna coincidencia

---

Nota

Tenga en cuenta que la instrucción `END_CASE` debe terminar con un punto y coma.

Lista de valores

La lista de valores incluye los valores permitidos para la expresión de selección.

Lista de valores

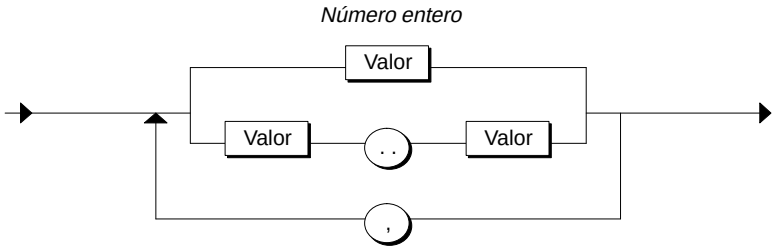


Figura 15-3 *Sintaxis: Lista de valores*

Reglas

Al formular la lista de valores debe tener en cuenta que:

- toda lista de valores o lista puede comenzar con una constante, una lista de constantes o un rango de constantes,
- los valores de la lista de valores deben ser valores ENTEROS,
- cada valor sólo debe aparecer una vez.

Ejemplos

El ejemplo 15-2 ilustra el uso de la instrucción CASE. Generalmente, la variable TW es del tipo entero.

```
CASE TW OF
   1:  VISUALIZACION         := OVEN_TEMP;
   2:  VISUALIZACION         := MOTOR_SPEED;
   3:  VISUALIZACION         := GROSS_TARE;
                      AW4    := 16#0003;
   4..10:VISUALIZACION       := INT_TO_DINT (TW);
                      AW4    := 16#0004;
   11,13,19:VISUALIZACION    := 99;
                      AW4    := 16#0005;
ELSE:  VISUALIZACION         := 0;
       TW_ERROR              := 1;
END_CASE;
```

Ejemplo 15-2 Instrucción CASE

Nota

Asegúrese de que el tiempo de ejecución de los bucles no sea demasiado largo, pues de lo contrario la CPU pasaría a STOP con demora en la confirmación.

15.4 Instrucción FOR

Principio

Una instrucción FOR o una instrucción de repetición ejecuta una secuencia de instrucciones en un bucle, para lo cual se asignan valores continuos a una variable (las variables en ejecución). La variable en ejecución debe ser el identificador de una variable local del tipo INT o DINT.

Instrucción FOR

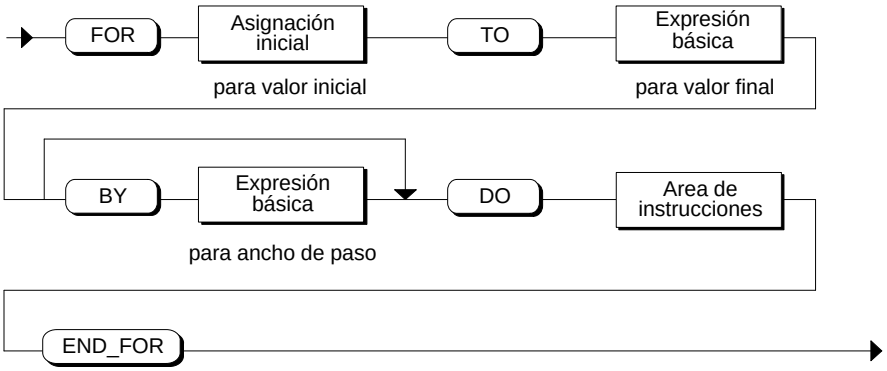


Figura 15-4 *Sintaxis: Instrucción FOR*

La definición de un bucle con FOR incluye la determinación de un valor inicial y un valor final. Ambos valores deben ser del mismo tipo que las variables en ejecución.

Ejecución

La instrucción FOR se ejecuta según las siguientes reglas:

1. Al iniciarse el bucle la variable en ejecución toma el valor inicial, que aumenta el paso especificado (paso positivo) o lo disminuye (paso negativo) después de cada vez que se recorre el ciclo, hasta que se alcanza el valor final.
2. En cada ciclo se comprueba si la condición

   |`Valor_inicial`| `<=` |`Valor_final`|

   se ha cumplido. En caso afirmativo, se ejecuta la secuencia de instrucciones; en caso contrario, se salta el bucle y, por tanto, la secuencia de instrucciones.

Nota

Tenga en cuenta que la instrucción END_FOR debe terminar con un punto y coma.

Asignación inicial

Para formar el valor inicial de la variable en ejecución se dispone de la asignación inicial representada en la figura 15-5.

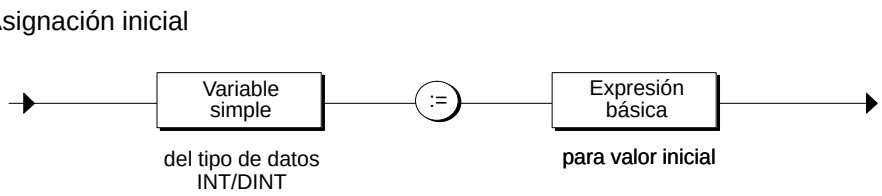


Figura 15-5 *Sintaxis: Asignación inicial*

Ejemplos:

```
FOR I  := 1 TO 20 DO
FOR I  := 1 TO (Inicio+J) DO
```

Valor final y paso

Puede formar una expresión básica para formar el valor final, y otra expresión básica para formar el paso deseado.

Reglas

La instrucción FOR se ejecuta observando las siguientes reglas:

- La especificación de *BY [paso]* puede omitirse. Si falta la indicación, el paso es de +1.
- El valor inicial, el valor final y el paso son expresiones (v. cap. 13). La valoración se realiza una sola vez al principio de la ejecución de la instrucción FOR.
- No está permitido un cambio de los dos valores (del valor final y del paso) durante la ejecución.

Ejemplo

El ejemplo 15-3 ilustra el uso de la instrucción FOR:

```
FUNCTION_BLOCK BUSCAR
VAR
   INDICE            : INT;
   PALABRA_CLAVE     : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
FOR INDEX:= 1 TO 50 BY 2 DO
   IF PALABRA_CLAVE [INDICE] = 'CLAVE' THEN
   EXIT;
   END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

Ejemplo 15-3 Instrucción FOR

15.5 Instrucción WHILE

Principio

La instrucción WHILE permite repetir la ejecución de una secuencia de instrucciones controlando una condición de ejecución. La condición de ejecución se forma siguiendo las reglas de una expresión lógica.

Instrucción WHILE

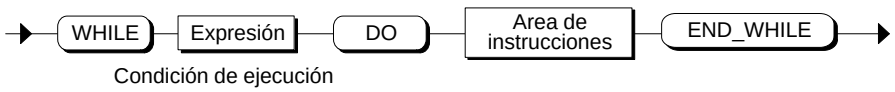


Figura 15-6 *Sintaxis: Instrucción WHILE*

El área de instrucciones que sigue a DO se repite hasta que la condición de ejecución posee el valor TRUE.

Ejecución

La orden WHILE se ejecuta según las siguientes reglas:

1. **Antes** de cada ejecución del área de instrucciones parcial se valora la condición de ejecución.
2. Si se produce el valor TRUE, se ejecuta el área de instrucciones.
3. Si se produce el valor FALSE, se cancela la ejecución de la instrucción WHILE. Esta circunstancia puede ocurrir en la primera valoración.

Nota

Tenga en cuenta que la instrucción END_WHILE debe terminar con un punto y coma.

Ejemplo

El ejemplo 15-4 ilustra el uso de la instrucción WHILE:

```
FUNCTION_BLOCK BUSCAR
VAR
   INDICE            : INT;
   PALABRA_CLAVE     : ARRAY [1..50] OF STRING;
END_VAR
BEGIN
INDICE:= 1;
WHILE INDICE <= 50 AND PALABRA_CLAVE[INDICE]<>'CLAVE' DO
   INDICE:= INDICE + 2;
END_WHILE;
END_FUNCTION_BLOCK
```

Ejemplo 15-4 Instrucción WHILE

15.6 Instrucción REPEAT

Principio

Una instrucción REPEAT produce la ejecución repetida de una secuencia de instrucciones que se encuentre comprendida entre REPEAT y UNTIL, hasta que se cumpla una condición de interrupción. La condición de interrupción se forma con las reglas de una expresión lógica.

Instrucción REPEAT

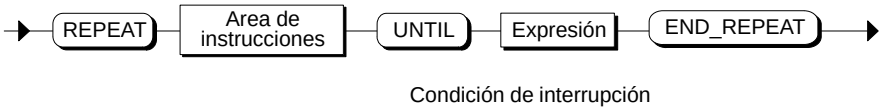


Figura 15-7 *Sintaxis: Instrucción REPEAT*

La condición se comprueba **después** de concluirse la ejecución de la secuencia de instrucciones. Esto significa que el cuerpo debe ejecutarse al menos una vez, incluso cuando la condición de interrupción se cumpla desde un principio.

Nota

Tenga en cuenta que la instrucción END_REPEAT debe terminar con un punto y coma.

Ejemplo

El ejemplo 15-5 ilustra el uso de la instrucción REPEAT.

```
FUNCTION_BLOCK BUSCAR
VAR
   INDICE            : INT;
   PALABRA_CLAVE     : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
INDICE:= 0;
REPEAT
   INDICE:= INDICE + 2;
UNTIL
   INDICE > 50 OR PALABRA_CLAVE[INDICE] = 'CLAVE'
END_REPEAT;
END_FUNCTION_BLOCK
```

Ejemplo 15-5 Instrucción REPEAT

15.7 Instrucción CONTINUE

Principio

Una instrucción CONTINUE provoca la interrupción del recorrido instantáneo del bucle de una instrucción de repetición (instrucción FOR, WHILE o REPEAT), y produce la reinicialización del bucle.

Instrucción CONTINUE

Figura 15-8 *Sintaxis: Instrucción CONTINUE*

En un bucle WHILE la condición inicial (en el caso de un bucle REPEAT, la condición final) decide si se repite una secuencia de instrucciones.

En una instrucción FOR, inmediatamente después de una instrucción CONTINUE la variable en ejecución aumenta el paso especificado.

Ejemplo

El ejemplo 15-6 ilustra el uso de la instrucción CONTINUE:

```
FUNCTION_BLOCK_CONTINUE
VAR
   INDICE            :INT;
   ARRAY             :ARRAY[1..100] OF INT;
END_VAR
BEGIN
INDICE:= 0;
WHILE INDICE <= 100 DO
   INDICE:= INDICE + 1;
   // Cuando ARRAY[INDICE] es igual a INDICE,
   // entonces ARRAY[INDICE] no varía:
       IF ARRAY[INDICE] = INDICE THEN
                     CONTINUE;
       END_IF;
       ARRAY[INDICE]:= 0;
   // Siguientes instrucciones
   //....
END_WHILE;
END_FUNCTION_BLOCK
```

Ejemplo 15-6 Instrucción CONTINUE

15.8 Instrucción EXIT

Principio

Una instrucción EXIT sirve para abandonar un bucle (bucle FOR, WHILE o REPEAT) en un punto cualquiera y con independencia que se haya cumplido o no la condición de interrupción.

Instrucción EXIT

Figura 15-9 *Sintaxis: Instrucción EXIT*

Esta instrucción hace que se abandone de inmediato la instrucción de repetición que rodea inmediatamente a la instrucción EXIT.

La ejecución del programa continúa después del final del bucle de repetición (p.ej., después de END_FOR).

Ejemplo

El ejemplo 15-7 ilustra el uso de la instrucción EXIT:

```
FUNCTION_BLOCK_EXIT
VAR
   INDICE_1          : INT;
   INDICE_2          : INT;
   INDICE_BUSCADO    : INT;
   PALABRA_CLAVE     : ARRAY[1..51] OF STRING;
END_VAR
BEGIN
INDICE_2    := 0;
FOR INDICE_1  := 1 TO 51 BY 2 DO
   //Abandona el bucle FOR cuando
   //PALABRA_CLAVE[INDICE_1] es igual a 'CLAVE':
   IF PALABRA_CLAVE[INDICE_1] = 'CLAVE' THEN
       INDICE_2:= INDICE_1;
       EXIT;
   END_IF;
END_FOR;
// La siguiente asignación de valor se ejecuta
// después de ejecutar EXIT o después del final
// normal del bucle FOR
INDICE_BUSCADO:= INDICE_2;
END_FUNCTION_BLOCK
```

Ejemplo 15-7 Instrucción EXIT

15.9 Instrucción GOTO

Principio

Con una instrucción GOTO puede ejecutar un salto en el programa. Produce el salto inmediato hasta una meta de salto especificada y, por tanto, hasta otra instrucción dentro del mismo bloque.

Las instrucciones GOTO deben utilizarse sólo en casos especiales; p.ej., para procesar errores. Según las reglas de la programación estructurada la instrucción GOTO no debería utilizarse.

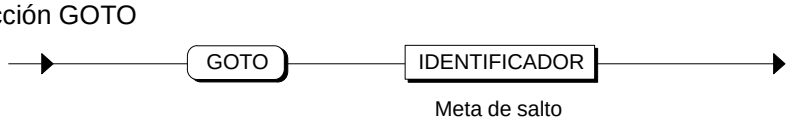


Figura 15-10 *Sintaxis: Instrucción GOTO*

La meta del salto designa una marca dentro del bloque de declaración LABEL / END_LABEL. Esta marca precede a la instrucción que se ejecuta después de GOTO.

Reglas

Para utilizar la instrucción GOTO deben tenerse en cuenta las siguientes reglas:

- La meta de una instrucción de salto debe de estar situada dentro del mismo bloque.
- La meta del salto debe ser unívoca.
- No se puede saltar a un bucle, pero sí desde un bucle.

Ejemplo El ejemplo 15-8 ilustra el uso de la instrucción GOTO:

```
FUNCTION_BLOCK FB3//GOTO_BSP
VAR
    INDICE              : INT;
    A                   : INT;
    B                   : INT;
    C                   : INT;
    PALABRA_CLAVE       : ARRAY[1..51] OF STRING;
END_VAR
LABEL
    MARCA1,  MARCA2, MARCA3;
END_LABEL
BEGIN
IF A > B THEN GOTO MARCA1;
    ELSIF A > C THEN GOTO MARCA2;
END_IF;
//...
MARCA1  :               INDICE:= 1;
        GOTO MARCA3;
MARCA2  :               INDICE:= 2;
//...
MARCA3  :               ;
//...

END_FUNCTION_BLOCK
```

Ejemplo 15-8 Instrucción de salto GOTO

15.10 Instrucción RETURN

Principio

Una instrucción RETURN hace que se abandone el bloque que se está ejecutando actualmente (OB, FB, FC) y se retorna al bloque invocante (o al sistema operativo, en caso de que se abandone un OB).

Instrucción RETURN

Figura 15-11 *Sintaxis: Instrucción RETURN*

Nota

Una instrucción RETURN situada al final de la parte de ejecución de un bloque lógico o de la tabla de declaración de un bloque de datos es redundante, puesto que se ejecuta automáticamente.

Llamada a funciones y bloques de función 16

Resumen

Desde un bloque SCL puede llamar a otras funciones (FC) o bloques de función (FB). Los bloques a los que puede llamarse son los siguientes:

- Funciones y bloques de función creados en SCL
- Funciones y bloques de función programados en otro lenguaje STEP 7 (p.ej.: AWL, KOP)
- Funciones del sistema (SFC) y bloques de función del sistema (SFB), disponibles en el sistema operativo de la CPU utilizada por usted.

Indice del capítulo

16.1 Llamada y transferencia de parámetros

Transferencia de parámetros

Al llamar a funciones o bloques de función se produce un intercambio de datos entre el bloque invocante y el bloque llamado. Los parámetros que deben transferirse deben especificarse en la llamada como lista de parámetros. Los parámetros se escriben entre paréntesis. Varios paréntesis juntos se separan por comas.

Principio

En el siguiente ejemplo de una llamada a función se especifican respectivamente un parámetro de entrada, un parámetro de entrada/salida y un parámetro de salida.

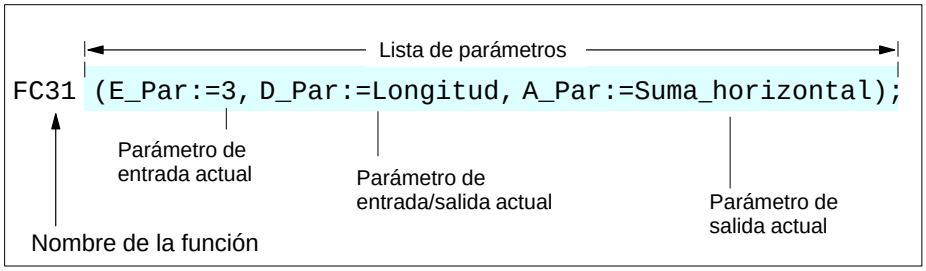


Figura 16-1 *Principio de la transferencia de parámetros*

La especificación de los parámetros es formalmente una asignación de valor (v. fig. 16-2). Mediante esta asignación de valor usted asigna un valor determinado (parámetro actual) a los parámetros que ha definido en la tabla de declaración del bloque llamado (parámetros formales).

Parámetros actuales		Parámetros formales
3	⇒	E_Par
LONGITUD	⇔	D_Par
Suma_horizontal	⇐	A_Par

Figura 16-2 *Asignación de valor dentro de la lista de parámetros*

Parámetros formales

Los parámetros formales son los que espera el bloque cuando efectúa la llamada. Se trata únicamente de ”comodines” para los parámetros actuales que son transferidos al bloque cuando se efectúa la llamada. Estos parámetros han sido definidos por usted en las tablas de declaración de un bloque (FB o FC).

Tabla 16-1 Bloques de declaración permitidos para parámetros formales

Bloques de declaración	Datos	Palabra clave
Bloque de parámetros	Parámetro de entrada	VAR_INPUT *Lista de declaración* END_VAR
	Parámetro de salida	VAR_OUTPUT *Lista de declaración* END_VAR
	Parámetro de entrada/salida	VAR_IN_OUT *Lista de declaración* END_VAR

16.2 Llamada a bloques de función (FB o SFB)

Instancia global y local

Al llamar a un bloque de función con SCL puede utilizar:

- bloques de datos de instancia globales,
- y áreas de instancia locales del bloque de datos de instancia actual

La llamada de un FB como instancia local se diferencia de la llamada como instancia global en el memorizado de los datos. En este último caso los datos no se depositan en un DB separado, sino que se anidan en el bloque de datos de instancia del FB invocante.

Llamada a FB

FB: bloque de función
SFB: bloque de función del sistema

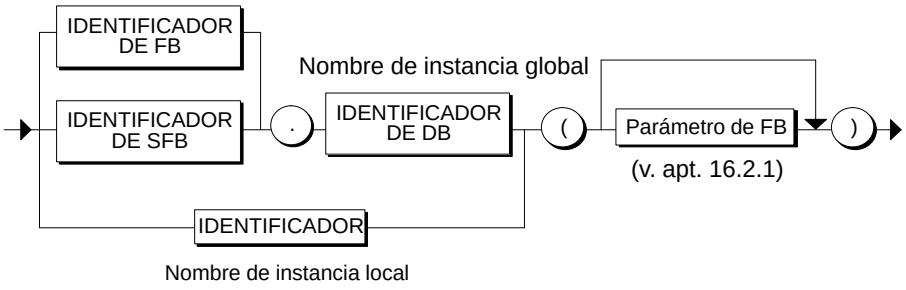


Figura 16-3 *Sintaxis: Llamada a FB*

Llamada como instancia global

La llamada se realiza en una instrucción de llamada indicando:

- el nombre del bloque de función o del bloque de función del sistema (identificación de FB o de SFB),
- el bloque de datos de instancia (identificación de DB),
- la especificación de parámetros (parámetros de FB).

Una llamada de una instancia global puede definirse absoluta o simbólicamente.

Llamada absoluta:

```
FB10.DB20 (X1:=5,X2:=78,......);
```

Especificación de parámetros

Llamada simbólica:

```
ACCIONAMENTO.INTRO (X1:=5,X2:=78,......);
```

Figura 16-4 *Llamada a FB10 con bloque de instancia DB20*

Llamada como instancia local

La llamada se realiza en una instrucción de llamada indicando:

- el nombre de instancia local (IDENTIFICADOR),
- la especificación de parámetros (parámetros de FB).

Una llamada a una instancia local es siempre simbólica, p.ej.:

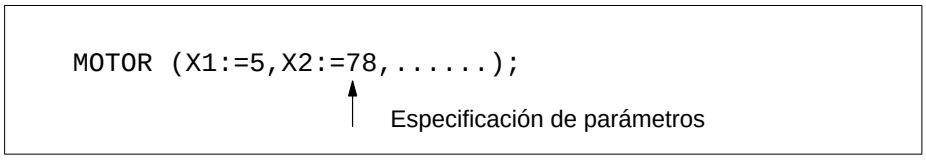


Figura 16-5 *Llamada a una instancia local*

16.2.1 Parámetros de FB

Principio

Al efectuar una llamada de un bloque de función (como instancia global o como instancia local) debe diferenciar, dentro de la lista de parámetros, entre:

- los parámetros de entrada, y
- los parámetros de entrada/salida

de un FB. En ambos casos, mediante **asignaciones de valor** asignará los parámetros actuales a los parámetros formales:

```
Parámetros      Parámetros
formales        actuales
E_Par    ⇐    3       //Asignación de entrada
D_Par    ⇔    LONGITUD//Asignación de entrada/salida
```

Figura 16-6 *Asignación de valor dentro de la lista de parámetros*

Los parámetros de salida no se especifican al llamar a un FB, sino que se asignan después de la llamada.

La sintaxis de especificación de parámetros de FB es igual en llamada de instancias globales y en llamada de instancias locales.

Parámetros de FB

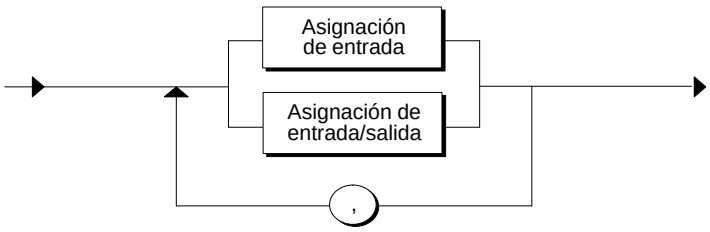


Figura 16-7 *Sintaxis:Parámetros de FB*

Ejemplo

Una llamada con una asignación del parámetro de entrada y del parámetro de entrada/salida podría tener, por ejemplo, el siguiente aspecto:

```
FB31.DB77(E_Par:=3, D_Par:=LONGITUD);
```

Reglas

Para la asignación de parámetros se aplican las siguientes reglas:

- Las asignaciones pueden estar en cualquier orden.
- Los tipos de datos del parámetro formal y del parámetro actual deben coincidir.
- Cada una de las asignaciones se separan por comas.
- Las asignaciones de salida no son posibles en llamadas a FB. El valor de un parámetro de salida declarado está memorizado en los datos de instancia, donde están accesibles para todos los FB. Para leerlos debe definir un acceso a partir de un FB (v. apt. 14.8).

Resultado después de la llamada

Después de recorrer el bloque:

- Los parámetros de entrada actuales transferidos permanecen inalterados.
- Los valores transferidos o modificados de los parámetros de entrada/salida están actualizados. Una excepción la constituyen los parámetros de entrada/salida de un tipo de datos simple (a este respecto, v. apt. 16.2.3).
- Los parámetros de salida del bloque invocante pueden leerse a partir del bloque de instancia global o desde el bloque de instancia local. A este respecto, ver el ejemplo 16-3 en la página 16-11.

16.2.2 Asignación de entrada (FB)

Principio

Mediante las asignaciones de entrada se asignan parámetros actuales a los parámetros formales de entrada. El FB **no** puede cambiar estos parámetros actuales. La asignación de parámetros de entrada actuales es opcional. Si no se especifica ningún parámetro actual se mantienen los valores de la última llamada.

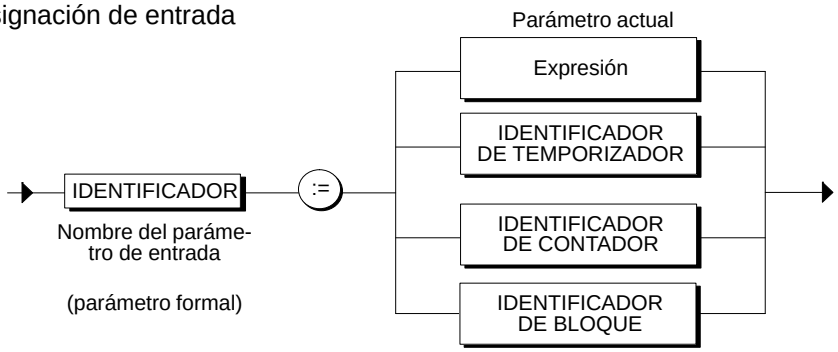


Figura 16-8 *Sintaxis: Asignación de entrada*

Posibles parámetros actuales

En las asignaciones de entrada son posibles los siguientes parámetros formales:

Tabla 16-2 Parámetros actuales en asignaciones de entrada

Parámetro actual	Explicación
Expresión	• Expresión aritmética, lógica o de comparación • Constante • Variable ampliada
Identificador de TEMPORIZADOR/ CONTADOR	Usted determina un temporizador determinado o un contador determinado que debe utilizarse en la ejecución de un bloque (v. cap. 17).
Identificador de BLOQUE	Usted especifica termina un determinado bloque que debe utilizarse como parámetro de entrada. El tipo de bloque (FB, FC, DB) se determina en la declaración del parámetro de entrada. Al especificar el parámetro especifique el número del bloque. El número puede ser tanto absoluto como simbólico (v. cap. 9).

16.2.3 Asignación de entrada/salida (FB)

Principio

Las asignaciones de entrada/salida sirven para asignar parámetros actuales a los parámetros formales de entrada/salida del FB llamado.

A diferencia de los parámetros de entrada, el FB llamado puede cambiar los parámetros de entrada/salida. El nuevo valor del parámetro que se origina al ejecutar el FB se reescribe en el parámetro actual, escribiéndose sobre el valor original.

Cuando en el FB llamado existen declarados parámetros de entrada/salida, éstos se transfieren en la primera llamada. En ese caso la especificación de parámetros actuales es opcional.

Asignación de entrada/salida

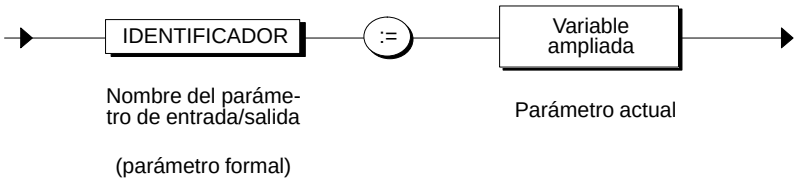


Figura 16-9 *Sintaxis: Asignación de entrada/salida*

Parámetros actuales de una asignación de entrada/salida

Dado que los parámetros actuales asignados pueden variar durante el recorrido del FB por tratarse de parámetros de entrada/salida), debe ser una variable. Por esta razón en las asignaciones de entrada/salida no puede asignarse un parámetro de entrada (el nuevo valor no podría reescribirse).

Tabla 16-3 Parámetros actuales en asignaciones de entrada/salida

Parámetro actual	Explicación
Variable ampliada	Se dispone de los siguientes tipos de variables ampliadas: Variable simple y parámetro Acceso a variable absoluta Acceso a bloque de datos Llamada a función (ver también el capítulo 14)

Particularidades

Tenga en cuenta las siguientes particularidades:

- Después de recorrer el bloque se actualiza el valor que ha cambiado del parámetro de entrada/salida. Una excepción la constituyen los parámetros de entrada/salida de un tipo de datos **simple**. Aquí la actualización sólo se produce cuando en la llamada se ha especificado un parámetro actual.
- En el caso de parámetros de entrada/salida del tipo de dato **no simple** no están permitidos como parámetros actuales:
  - parámetros de entrada/salida de FB,
  - parámetros de FC.
- **Parámetros ANY:** básicamente se aplica en este caso lo expuesto arriba. Adicionalmente hay que tener en cuenta que **no** está permitido que las constantes actúen de parámetros actuales.

16.2.4 Ejemplo de llamada a una instancia global

Principio

Un bloque de función con un bucle FOR podría tener el siguiente aspecto (v. ej. 16-1). En los ejemplos se ha supuesto que en la tabla de símbolos se ha declarado el símbolo TEST para el FB17.

```
FUNCTION_BLOCK TEST
   VAR_INPUT
   VALOR_FINAL  :  INT; //Parámetro de entrada
   END_VAR
   VAR_IN_OUT
   IQ1          :  REAL; //Parámetro de entrada/salida
   END_VAR
   VAR_OUTPUT
   CONTROL      :  BOOL; //Parámetro de salida
    END_VAR
    VAR
   INDICE       :  INT;
   END_VAR
BEGIN
   CONTROL      := FALSE;
   FOR INDICE   := 1 TO VALOR_FINAL DO
      IQ1:= IQ1 * 2;
      IF IQ1 > 10000 THEN
         CONTROL:= TRUE;
      END_IF;
    END_FOR;
END_FUNCTION_BLOCK
```

Ejemplo 16-1 Ejemplo de un FB

Llamada

Para llamar a este FB puede elegir entre las siguientes variantes. Se presupone que en el bloque invocante la VARIABLE1 está declarada como variable REAL.

```
      //Llamada absoluta, instancia global:
FB17.DB10 (VALOR_FINAL:=10, IQ1:= VARIABLE1);

      //Llamada simbólica, instancia global:
TEST.TEST_1 (VALOR_FINAL:= 10, IQ1:= VARIABLE1) ;
```

Ejemplo 16-2 Ejemplo de llamada a FB con bloque de datos de instancia

Resultado

Después de ejecutar el bloque, para el parámetro de entrada/salida IQ1 se dispone del valor calculado en la VARIABLE1.

Leer valor de salida

Con ayuda de los dos ejemplos siguientes pretenden explicarse las posibles variantes para suprimir el parámetro de salida `CONTROL`.

```
//El acceso al parámetro de salida se realiza
//mediante:
    RESULTADO:= DB10.CONTROL;
//También puede recurrir a los parámetros
//de salida de otra llamada directa a FB para
//especificar un parámetro de entrada:
    FB17.DB12 (EIN_1:= DB10.CONTROL);
```

Ejemplo 16-3 Resultado de llamada a FB con bloque de datos de instancia

16.2.5 Ejemplo de llamada a una instancia local

Principio

Un bloque de función con un bucle FOR simple podría programarse como en el ejemplo 16-1, donde se supone que en la tabla de símbolos se ha declarado el símbolo `TEST` para el `FB17`.

Llamada

Presuponiendo que la VARIABLE1 se ha declarado como variable REAL en el bloque invocante, puede llamar a este FB de la siguiente forma:

```
// Llamada, instancia local:
TEST_L (VALOR_FINAL:= 10, IQ1:= VARIABLE1) ;
```

Ejemplo 16-4 Ejemplo de llamada a FB como instancia local

Para ello TEST_L debe haberse declarado en la declaración de variables como sigue:

```
VAR
  TEST_L : TEST;
END_VAR
```

Leer parámetros de salida

El parámetro de salida `CONTROL` puede leerse de la siguiente forma:

```
// El acceso al parámetro de salida se realiza
// mediante
RESULTADO:= TEST_L.CONTROL;
```

Ejemplo 16-5 Resultado de llamada a FB como instancia local

16.3 Llamada a funciones

Valor de respuesta

A diferencia de los bloques de función, las funciones proporcionan como resultado el valor de respuesta. Por esta razón las funciones pueden tratarse como operandos. Una excepción la constituye la función cuyo valor de respuesta es del tipo VOID.

Por ejemplo, en la siguiente asignación de valor se ha llamado a la función DISTANCIA con determinados parámetros:

```
LONGITUD:= DISTANCIA (X1:=-3, Y1:=2);
¡El valor de respuesta es DISTANCIA!
```

La función calcula el valor de respuesta que tiene el mismo nombre que la función, y lo devuelve al bloque invocante. Allí este valor sustituye a la llamada la función.

El valor de respuesta puede utilizarse en los siguientes elementos de una FC o un FB:

- en una asignación de valor,
- en una expresión lógica, aritmética o de comparación, o
- como parámetro para otra llamada a bloque de función o a función.

Una excepción la constituyen las funciones del tipo VOID. No tienen valor de respuesta, por lo que no pueden utilizarse en expresiones.

La figura 16-10 ilustra la sintaxis de una llamada a función.

Llamada a función

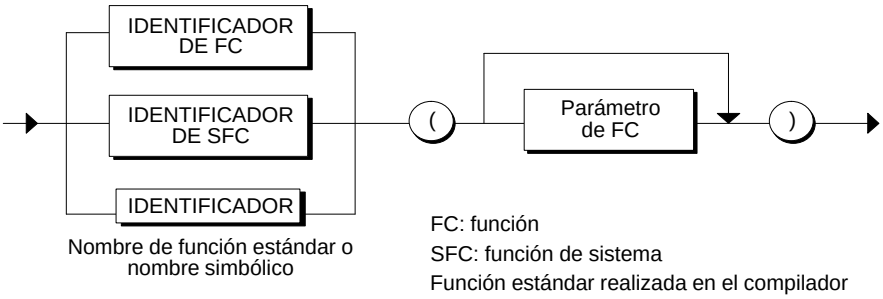


Figura 16-10 *Sintaxis: Llamada a función*

Nota

Si en SCL se llama a una función que aún no tiene asignado valor de respuesta, el programa de usuario puede ejecutarse de forma errónea.

En una función de SCL puede producirse este caso cuando el valor de respuesta se haya asignado pero no se haya atravesado la instrucción correspondiente.

En una función de AWL, KOP o FUP puede producirse este caso cuando la función se ha programado sin asignarle un valor de respuesta o cuando no se ha atravesado la instrucción correspondiente.

Llamada

La llamada de una función se realiza especificando:

- el nombre de la función (`IDENTIFICADOR_DE_FC, IDENTIFICADOR_DE_SFC, IDENTIFICADOR`), y
- la lista de parámetros.

Ejemplo

El nombre de la función que identifica al valor de respuesta puede especificarse de forma absoluta o simbólica, p.ej.:

```
FC31 (X1:=5, Q1:= Suma_horizontal)
DISTANCIA (X1:=5, Q1:= Suma_horizontal)
```

Resultado de la llamda

Después de la llamada puede disponerse de los resultados de una llamada a función en forma de:

- valor de respuesta, o
- parámetro de salida y parámetro de entrada/salida (parámetros actuales).

Encontrará más información al respecto en el capítulo 18.

16.3.1 Parámetros de FC

Principio

A diferencia de los bloques de función, las funciones no tienen memoria en la que poder guardar los valores de los parámetros. Durante la ejecución de la función los datos locales sólo se guardan temporalmente. Por esta razón, en la llamada deben asignarse parámetros actuales a todos los parámetros formales de entrada, de entrada/salida y de salida definidos en la tabla de declaración de una función.

La figura 16-11 muestra la sintaxis de la asignación de parámetros de FC:

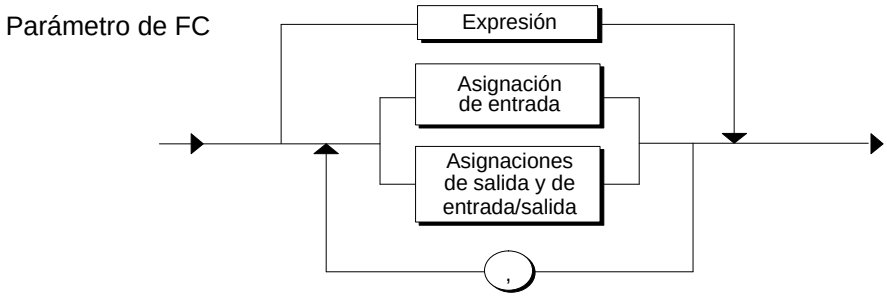


Figura 16-11 *Sintaxis: Parámetro de FC*

Una llamada con asignación de un parámetro de entrada, de salida y de entrada/salida podría tener, p.ej., el siguiente aspecto:

```
FC32 (E_Param1:=5,D_Param1:=LONGITUD,
      A_Param1:=Suma_horizontal)
```

Reglas

Para asignar parámetros se aplican las siguientes reglas:

- Las asignaciones pueden efectuarse en cualquier orden.
- Los tipos de datos del parámetro formal y del parámetro actual deben coincidir.
- Las asignaciones están separadas entre sí por comas.

16.3.2 Asignación de entrada (FC)

Principio

Mediante las asignaciones de entrada se asignan parámetros actuales a los parámetros formales de entrada de la FC llamada. La FC puede trabajar con estos parámetros actuales, pero no cambiarlos. A diferencia de la llamada al FB, esta asignación **no es opcional** en la llamada a FC. Las asignaciones de entrada tienen la siguiente sintaxis:

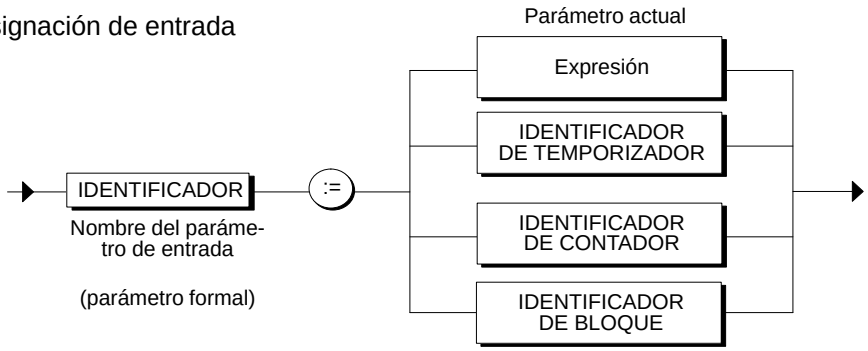


Figura 16-12 *Sintaxis: Asignación de entrada*

Parámetros actuales en asignaciones de entrada

En las asignaciones de entrada pueden asignarse los siguientes parámetros actuales:

Tabla 16-4 Parámetros actuales en asignaciones de entrada

Parámetro actual	Explicación
Expresión	Una expresión representa un valor y está compuesta por operandos y operadores. Se dispone de los siguientes tipos de expresiones: • Expresión aritmética, lógica o de comparación • Constante • Variable ampliada
Identificador de TEMPORIZADOR / CONTADOR	Usted determina un temporizador determinado o un contador determinado que debe utilizarse en la ejecución de un bloque (v. cap. 17).
Identificador de BLOQUE	Usted especifica termina un determinado bloque que debe utilizarse como parámetro de entrada. El tipo de bloque (FB, FC, DB) se determina en la declaración del parámetro de entrada. Al especificar el parámetro especifique el número del bloque. El número puede ser tanto absoluto como simbólico (v. cap. 9).

Particularidades

Tenga en cuenta que en los parámetros de FC que no sean del **tipo de datos simple**, no están permitidos como parámetros actuales los parámetros de entrada/salida de FB, ni los parámetros de FC.

16.3.3 Asignaciones de salida y de entrada/salida (FC)

Principio

En una asignación de salida puede establecer dónde se escribirán los valores de salida generados en la ejecución de una función. Con una asignación de entrada/salida asigne un valor actual a un parámetro de entrada/salida.

La figura 16-13 muestra la sintaxis de las asignaciones de salida y de entrada/salida:

Asignación de salida y de entrada/salida

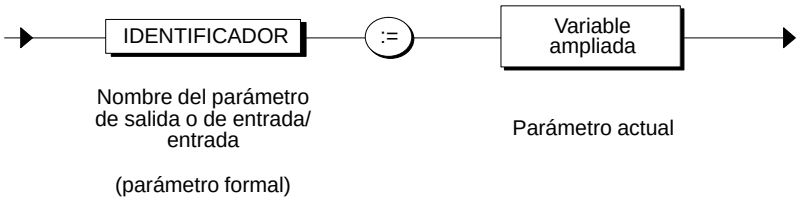


Figura 16-13 *Sintaxis: Asignación de salida y de entrada/salida*

Parámetros en asignaciones de salida y de entrada/salida

Los parámetros actuales en asignaciones de salida y de entrada/salida deben ser una variable, puesto que la FC debe escribir valores en los parámetros. Por esta razón, en asignaciones de entrada/salida no pueden asignarse parámetros de entrada (el valor no podría escribirse) sino sólo variables ampliadas.

Tabla 16-5 Parámetros actuales en asignaciones de salida y de entrada/salida

Parámetro actual	Explicación
Variable ampliada	Se dispone de los siguientes tipos de variables ampliadas: Variable simple y parámetro Acceso a variable absoluta Acceso a bloque de datos Llamada a función (ver también el capítulo 14)

Particularidades

Tenga en cuenta las siguientes particularidades:

- Después de ejecutarse el bloque se actualiza el valor modificado del parámetro de entrada/salida.
- En caso de parámetros de entrada/salida que **no** sean de tipo de datos **simple**, no están permitidos como parámetros actuales:
  - parámetros de entrada de FB,
  - parámetros de entrada/salida de FB, o
  - parámetros de FC.
- **Parámetros ANY**: básicamente se aplica el enunciado primero. En caso de parámetros de entrada/salida que **no** sean de tipo de datos **simple**, no están permitidos como parámetros actuales:
  - parámetros de entrada de FB,
  - parámetros de entrada de FC.

  Adicionalmente hay que tener en cuenta que **no** está permitido que los constantes actúen como parámetros actuales.
  Cuando para el resultado de la función (valor de respuesta) se ha especificado el tipo ANY, se aplican además las siguientes reglas:
  - Todos los parámetros ANY deben especificarse con operandos cuyos tipos de datos correspondan a una misma clase. Se entiende por tipos de datos, por ejemplo, el conjunto de los tipos de datos numéricos (INT, DINT, REAL) o el conjunto de los tipos de datos de bits (BOOL, BYTE, WORD, DWORD). Cada uno de los demás tipos de datos constituye un tipo de datos diferente.
  - El compilador SCL presupone que para el tipo de datos del resultado actual de la función puede especificarse el tipo superior de los parámetros actuales asignados a los parámetros ANY.
    Con el resultado de la función están permitidas todas las operaciones definidas para ese tipo de datos.
- **Parámetros POINTER**: básicamente se aplica el primer enunciado. En caso de parámetros de entrada/salida que **no** sean de un tipo de datos **simple** no están permitidos como parámetros actuales:
  - parámetros de entrada de FB,
  - parámetros de entrada de FC.

16.3.4 Ejemplo de una llamada a función

Principio

Una función DISTANCIA para calcular la distancia entre dos puntos (X1, Y1) y (X2, Y2) en el plano utilizando coordenadas cartesianas podría tener el siguiente aspecto (en los ejemplos se ha supuesto siempre que en una tabla de símbolos se ha declarado el símbolo DISTANCIA para la FC37):

```
FUNCTION DISTANCIA: REAL
      VAR_INPUT
            X1: REAL;
            X2: REAL;
            Y1: REAL;
            Y2: REAL;
      END_VAR
      VAR_OUTPUT
            Q2: REAL;
      END_VAR
      BEGIN
            DISTANCIA:= RAIZ
            ( (X2-X1)**2 + (Y2-Y1)**2 );
            Q2:= X1+X2+Y1+Y2;
END_FUNCTION
```

Ejemplo 16-6 Cálculo de distancia

Para utilizar posteriormente un valor de la función dispone de las siguientes posibilidades:

en una asignación de valor, p.ej.:

```
LONGITUD:= DISTANCIA (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Suma_horizontal);
```

en una expresión aritmética o lógica, p.ej.:

```
RADIO + DISTANCIA (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Suma_horizontal)
```

en la especificación de parámetros de un bloque al que se llama, p.ej.:

```
FB32 (DISTANCIA:= DISTANCIA (X1:=-3, Y1:=2, X2:=8.9,
Y2:=7.4,
Q2:=Suma_horizontal);
```

Ejemplo 16-7 Cálculo de valores dentro de una FC

16.4 Parámetros definidos implícitamente

Resumen

Los parámetros definidos implícitamente son los parámetros que puede utilizar sin necesidad de declararlos previamente en un bloque. SCL pone a su disposición dos parámetros definidos de esta forma:

- el parámetro de entrada `EN` y
- el parámetro de salida `ENO`.

Ambos parámetros son del tipo de datos BOOL y están depositados en el área de datos temporales del bloque.

Parámetro de entrada EN

Todo bloque de función y toda función poseen el parámetro de entrada EN definido implícitamente. Cuando EN es igual a TRUE, el bloque llamado se ejecuta; en caso contrario, no se ejecuta. La especificación del parámetro EN es opcional.

Tenga en cuenta que EN no puede declararse en la tabla de declaración de un bloque o de una función.

Como EN es un parámetro de entrada, el usuario no puede cambiar EN dentro de un bloque.

Nota

El valor de respuesta de una función no está definido en caso de que la función no se haya llamado por ser EN:=FALSE.

Ejemplo

El siguiente ejemplo ilustra el uso del parámetro EN:

```
FUNCTION_BLOCK FB57
VAR
      RESULTADO   : REAL;
      MI_ENABLE   : BOOL;
END_VAR
//...
BEGIN
MI_ENABLE:= FALSE;
// Llamada a una función,
// donde se especifica el parámetro EN:

RESULTADO:= FC85 (EN:= MI_ENABLE, PAR_1:= 27);
// La FC85 no se ha ejecutado porque MI_ENABLE
// era igual a FALSE
//...
END_FUNCTION_BLOCK
```

Ejemplo 16-8 Utilización de EN

Parámetro de salida ENO

Todo bloque de función y toda función poseen el parámetro de salida ENO definido implícitamente, que es del tipo de datos BOOL. Al terminar la ejecución de un bloque el valor actual de las variables OK se deposita en ENO.

Inmediatamente después de llamar a un bloque puede comprobar, con ayuda del valor de ENO si todas las operaciones del bloque se han ejecutado correctamente o si se han producido errores.

Ejemplo

El siguiente ejemplo ilustra el uso del parámetro ENO.

```
FUNCTION_BLOCK FB57
      //...
      //...
BEGIN
// Llamada a un bloque de función:
FB30.DB30 (X1:=10, X2:=10.5);

// Comprobar si en el bloque llamado
// todo se ha ejecutado correctamente:

IF ENO THEN
      // Todo en orden
      //...
ELSE
      // Error.
      // Por lo tanto, tratamiento del error
      //...
END_IF;
      //...
      //...
END_FUNCTION_BLOCK
```

Ejemplo 16-9 Utilización de ENO

Ejemplo

El siguiente ejemplo muestra la encadenación de EN y ENO:

```
// EN y ENO también pueden utilizarse
// encadenados:

FB30.DB30(X1:=10, X2:=10.5);

// La siguiente función sólo se ejecutará
// cuando el FB 30 se haya recorrido sin
// errores:

RESULTADO:= FC 85 (EN:= ENO, PAR_1:= 27);
```

Ejemplo 16-10 Utilización de EN y ENO

Contadores y temporizadores

17

Resumen

En SCL puede controlar la ejecución del programa en función de un dato de temporizador o un estado de contador.

Para ello STEP 7 le ofrece funciones de contador y de temporizador estándar que puede utilizar en su programa SCL sin necesidad de declararlas previamente.

Indice del capítulo

17.1 Funciones de contaje

Resumen

STEP 7 dispone de una serie de funciones de contaje estándar. Dichos contadores puede utilizarlos en su programa SCL sin necesidad de declararlos previamente. Lo único que tiene que hacer es especificar los parámetros necesarios. STEP 7 ofrece las siguientes funciones de contaje:

- contadores incrementales (Counter Up)
- contadores decrementales (Counter Down)
- contadores incrementales y decrementales (Counter Up Down)

Llamada

A las funciones de contaje puede llamarse igual que las funciones. La identificación de función puede utilizarse en una expresión en sustitución de un operando, siempre que el tipo del resultado de la función sea compatible con el del operando sustituido.

Tabla 17-1 Nombres de las funciones de contaje

Nombre de la función	Significado
S_CU	contador incremental (Counter Up)
S_CD	contador decremental (Counter Down)
S_CUD	contador incremental/decremental (Counter Up Down)

Valor de la función

El valor de la función (valor de respuesta) que se devuelve al punto de llamada es el valor de contaje actual (formato BCD), de tipo de datos WORD. En el apartado 17.1.1 encontrará información al respecto.

Parámetros de llamada

En la tabla 17-2 puede consultar los parámetros de llamada junto con su identificación y su significado para las tres funciones de contaje. En general hay que distinguir los siguientes tipos de parámetros:

- Parámetros de control (p.ej.: ajustar, inicializar, indicar sentido de conteo)
- Valor predefinido para un estado de contador
- Salida de estado (que muestra si se ha alcanzado un valor final de contaje)
- Estado de contador en formato binario.

Tabla 17-2 Parámetros de llamada

Identificador	Parámetro	Tipo de datos	Descripción
C_NO		COUNTER	Identificador de contador (IDENTIFICACION DE CONTADOR); el área depende de la CPU.
CU	Entrada	BOOL	Entrada CU: contaje incremental
CD	Entrada	BOOL	Entrada CD: contaje decremental
S	Entrada	BOOL	Entrada para preajuste del contador
PV	Entrada	WORD	Valor comprendido entre 0 y 999 para ajuste del contador (introducido como 16#<Valor>, donde el valor está en formato BCD).
R	Entrada	BOOL	Entrada de inicialización
Q	Salida	BOOL	Estado del contador
CV	Salida	WORD	Estado del contador (binario)

Ejemplo

La llamada mencionada en el ejemplo 17-1 hace que al calcular la función se reserve un área de memoria global del tipo `COUNTER` (`CONTADOR`) de nombre Z12.

```
Valor_contador:= S_CUD (C_NO :=Z12,
                        CU    :=E0.1,
                        CD    :=E0.0,
                        S     :=E0.2 & E0.3,
                        PV    :=120,
                        R     :=FALSE,
                        Q     :=act_Flag,
                        CV    :=binVal);
```

Ejemplo 17-1 Llamada a una función de contaje decremental

Llamada dinámica

El lugar de un número absoluto de contador (p.ej.: C_NO=Z10), también puede especificar en la llamada una variable del tipo de datos COUNTER. Esto tiene la ventaja de que puede organizar dinámicamente la llamada a los contadores asignando otro número absoluto a esas variables cada vez que se efectúa una llamada.

```
Ejemplo:
FUNCTION_BLOCK CONTADOR;
VAR_INPUT
 MiContador: Contador;
END_VAR
:
currVAL:=S_CD (C_NO:=MiContador,........);
```

Reglas

Dado que los valores de los parámetros (p.ej., CD:=E.0) se guardan globalmente, su especificación es opcional en determinados casos. Al especificar parámetros deben observarse las siguientes reglas:

- El parámetro para identificación de contador C_NO debe especificarse siempre en la llamada.
- Dependiendo de la función de contaje, debe especificarse el parámetro CU (contador incremental) o el parámetro CD (contador decremental)
- Pueden omitirse por parejas las especificaciones del parámetro PV (valor predefinido) y S (ajuste).
- El valor del resultado en formato BCD es siempre el valor de la función.

Nota

Los nombres de las funciones y parámetros son iguales en nemotécnica SIMATIC y en nemotécnica IEC. Sólo la identificación del contador depende de la nemotécnica utilizada: *SIMATIC: Z* e *IEC: C*.

Ejemplo de llamada a funciones de contaje

El ejemplo 17-2 ilustra la llamada a funciones de contaje:

```
FUNCTION_BLOCK FB1


VAR
 currVal, binVal: word;
 actFlag: bool;
END_VAR
```

```
BEGIN
currVal  :=S_CD(C_NO:=Z10, CD:=TRUE, S:=TRUE,
               PV:=100, R:=FALSE, CV:=binVal,
               Q:=actFlag);
```

```
currVal  :=S_CU(C_NO:=Z11, CU:=M0.0, S:=M0.1,
               PV:=16#110, R:=M0.2, CV:=binVal,
               Q:=actFlag);
```

```
currVal  :=S_CUD(C_NO:=Z12, CD:=E0.0,
               CU:=E0.1,S:=E0.2 & E0.3, PV:=120,
               R:=FALSE,CV:=binVal, Q:=actFlag);
```

```
currVal  :=S_CD(C_NO:=Z10,CD:=FALSE,
               S:=FALSE,
               PV:=100, R:=TRUE, CV:=binVal,
               Q:=actFlag);


END_FUNCTION_BLOCK
```

Ejemplo 17-2 Llamada a funciones de contaje

17.1.1 Introducción y valoración del valor del contador

Resumen

Para introducir el valor predefinido o la valoración del resultado de la función necesita la representación interna del valor del contador (v. fig. 17-1).

Cuando se ajusta el contador (parámetro S) se escribe en el contador el valor definido por usted. El rango está comprendido entre 0 y 999. Usted puede variar el valor del contador dentro de este rango utilizando las operaciones contaje incremental/decremental, contaje incremental y contaje decremental.

Formato

La figura 17-1 ilustra la configuración de bits del valor del contador.

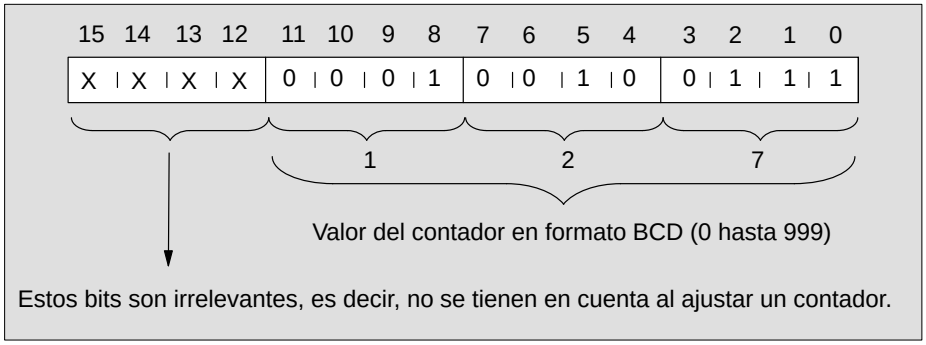


Figura 17-1 *Configuración de bits para el valor del contador*

Introducción

Puede cargar un valor de contador predefinido utilizando los siguientes formatos:

- Decimal como valor entero: por ejemplo 295, siempre que este valor corresponda a un código BCD válido.
- En formato BCD (introducción como constante hexadecimal); p.ej.: 16#127

Valoración

Puede valorar el resultado en dos formatos diferentes:

- Como resultado de función (tipo WORD): en formato BCD
- Como parámetro de salida CV (tipo WORD): binario

17.1.2 Contaje incremental (Counter Up)

Descripción

Con el contador Counter Up puede ejecutar operaciones de contaje incremental.

Funcionamiento

Tabla 17-3 Funcionamiento del contador incremental

Operación	Funcionamiento
Contaje incremental	El valor del contador avanza ”1” cuando el estado de la señal en la entrada **CU** cambia de ”0” a ”1” y el valor del contador es menor que 999.
Ajustar contador	Cuando el estado de la señal en la entrada **S** cambia de ”0” a ”1”, en el contador se ajusta el valor de la entrada **PV**. Este cambio de la señal es siempre necesario para ajustar el contador.
Inicializar	El contador se inicializa cuando la entrada **R**=1. Al inicializar el contador el valor del contador se coloca en ”0”.
Consultar contador	Al consultar el estado de la señal en la salida **Q** se obtiene ”1” cuando el valor del contador es mayor que ”0”. La consulta arroja el resultado ”0” cuando el valor del contador es igual ”0”.

17.1.3 Contaje decremental (Counter Down)

Descripción

Con el contador Counter Down puede ejecutar operaciones de contaje decremental.

Funcionamiento

Tabla 17-4 Funcionamiento del contador decremental

Operación	Funcionamiento
Contaje decremental	El valor del contador disminuye ”1” cuando el estado de la señal en la entrada **CD** cambia de ”0” a ”1” (flanco positivo) y el valor del contador es mayor que ”0”.
Ajustar contador	Cuando el estado de la señal en la entrada **S** cambia de ”0” a ”1”, en el contador se ajusta el valor de la entrada **PV**. Este cambio de la señal es siempre necesario para ajustar el contador.
Inicializar	El contador se inicializa cuando la entrada **R**=1. Al inicializar el contador el valor del contador se coloca en ”0”.
Consultar contador	Al consultar el estado de la señal en la salida **Q** se obtiene ”1” cuando el valor del contador es mayor que ”0”. La consulta arroja el resultado ”0” cuando el valor del contador es igual ”0”.

17.1.4 Contaje incremental/decremental (Counter Up Down)

Descripción

Con el contador Counter Up Down puede ejecutar tanto operaciones de contaje incremental como decremental. Cuando los impulsos de contaje incremental y decremental son simultáneos se ejecutan ambas operaciones.
El valor del contador permanece inalterado.

Funcionamiento

Tabla 17-5 Funcionamiento del contador incremental/decremental

Operación	Funcionamiento
Contaje incremental	El valor del contador avanza ”1” cuando el estado de la señal en la entrada **CU** cambia de ”0” a ”1” y el valor del contador es menor que 999.
Contaje decremental	El valor del contador disminuye ”1” cuando el estado de la señal en la entrada **CD** cambia de ”0” a ”1” (flanco positivo) y el valor del contador es mayor que ”0”.
Ajustar contador	Cuando el estado de la señal en la entrada **S** cambia de ”0” a ”1”, en el contador se ajusta el valor de la entrada **PV**. Este cambio de la señal es siempre necesario para ajustar el contador.
Inicializar	El contador se inicializa cuando la entrada **R**=1. Al inicializar el contador el valor del contador se coloca en ”0”.
Consultar contador	Al consultar el estado de la señal en la salida **Q** se obtiene ”1” cuando el valor del contador es mayor que ”0”. La consulta arroja el resultado ”0” cuando el valor del contador es igual ”0”.

17.1.5 Ejemplo de función S_CD (contador decremental)

Especificación de parámetros

La tabla 17-6 muestra la especificación de parámetros de la función de ejemplo S_CD.

Tabla 17-6 Parámetros de llamada

Parámetro	Descripción
C_NO	`MiContador`
CD	`Entrada E0.0`
S	`AJUSTAR`
PV	`Predefinir 16#0089`
R	`Inizializar`
Q	`A0.7`
CV	`VALOR_BINARIO`

Ejemplo El ejemplo 17-3 ilustra la función de contaje S_CD:

```
FUNCTION_BLOCK CONTAJE
VAR_INPUT
 MICONTADOR: COUNTER;
END_VAR
VAR_OUTPUT
 RESULTADO: INT;
END_VAR
VAR
 AJUSTAR          : BOOL;
 INICIALIZAR      : BOOL;
 VALOR_BCD        : WORD; //Estado de contador BCD
 VALOR_BINARIO    : WORD; //Estado de contador binario
 PREDEFINICION    : WORD;
END_VAR
BEGIN
 A0.0:= 1;
 AJUSTAR:= E0.2;
 INICIALIZAR:= E0.3;
 PREDEFINICION:= 16#0089;
 VALOR_BCD:= S_CD
            (C_NO := MICONTADOR,//Contador decr.
            CD    := E0.0,
            S     := AJUSTAR,
            PV    := PREDEFINICION,
            R     := INIZIALIZAR,
            CV    := VALOR_BINARIO,
            Q     := A0.7);
RESULTADO:=WORD_TO_INT(VALOR_BINARIO);
             //Continúa procesamiento como
             //parámetro de salida
AW4:= VALOR_BCD;  //Para visualizar en la salida
END_FUNCTION_BLOCK
```

Ejemplo 17-3 Ejemplo de función de contaje

17.2 Funciones de temporización (TIMER)

Resumen

Los temporizadores son elementos de función de su programa que ejecutan y supervisan procesos controlados por tiempo. STEP 7 dispone de una serie de funciones de temporización estándar a las que puede acudir con SCL. Con las operaciones de temporización, en su programa puede:

- ajustar tiempos de espera,
- permitir tiempos de vigilancia,
- generar impulsos,
- medir tiempos.

Llamada

A las funciones de temporización se llama de igual forma que a las funciones de contaje. La identificación de función puede utilizarse en cualquier expresión en lugar de un operando siempre que el tipo del resultado de la función sea compatible con el del operando sustituido.

Tabla 17-7 Funciones de temporización STEP 7

Nombre de la función	Significado
S_PULSE	Impulso (Pulse)
S_PEXT	Impulso prolongado (Pulse Extended)
S_ODT	Retardo a la conexión (On Delay Time)
S_ODTS	Retardo a la conexión con memoria (Stored On Delay Time)
S_OFFDT	Retardo a la desconexión (Off Delay Time)

Valor de la función

El valor de la función (valores de respuesta) que se devuelve al punto de llamada es un valor de temporización del tipo de datos S5TIME. En el apartado 17.2.1 encontrará más información.

Parámetros de llamada

Los parámetros que deben especificarse están explicados en la descripción de cada una de las funciones estándar de la tabla. En la tabla 17-8 puede consultar los nombres de las 5 funciones de temporización junto con sus tipos de datos correspondientes.

En general hay que distinguir los siguientes tipos de parámetros:

- Parámetros de control (por ejemplo: ajustar, inicializar)
- Valor predefinido para el temporizador de arranque
- Salida de estado, que indica si el temporizador continúa funcionando
- Valor residual de temporización en formado binario

Tabla 17-8 Parámetros de llamada

Parámetro	Tipo de dato	Descripción
T_NO	TIMER	Identificador del temporizador; el área depende de la CPU
S	BOOL	Entrada inicial
TV	S5TIME	Predefinición del valor de temporización (formato BCD)
R	BOOL	Entrada de inicialización
Q	BOOL	Estado del temporizador
BI	WORD	Valor de temporización residual (binario)

Ejemplo

Al ejecutar la llamada mencionada en el ejemplo 17-4 se reserva un área de memoria global del tipo TIMER con el nombre T10.

```
RETARDO:= S_ODT (T_NO :=    T10,
                 S    :=    TRUE,
                 TV   :=    T#1s,
                 R    :=    FALSE,
                 BI   :=    biVal,
                 Q    :=    actFlag
                 );
```

Ejemplo 17-4 Llamada a una función de contaje decremental

Llamada dinámica

En lugar del número absoluto de temporizador (por ejemplo, T10), en la llamada también puede especificar una variable de tipo de datos TIMER. Esto último tiene la ventaja de que puede organizar dinámicamente la llamada al temporizador, asignando otro número absoluto a esta variable cada vez que se efectúa una llamada.

```
Ejemplo:
FUNCTION_BLOCK TEMPORIZADOR
VAR_INPUT
MiTemporizador: timer;
END_VAR
:
currTime:=S_ODT (T_NO:=MiTemporizador,.........)
```

Reglas

Dado que los valores de parámetros se guardan globalmente, su especificación es opcional en determinados casos. Al especificar parámetros deben observarse las siguientes reglas generales:

- El parámetro para la identificación del temporizador T_NO debe especificarse simbólica o absolutamente en la llamada.
- Puede omitirse por parejas la especificación del parámetro TV (valor predefinido) y S (ajuste).
- La especificación del parámetro es opcional. Puede acceder a Q y BI por medio de una asignación de valor.
- El valor del resultado en formato S5TIME es siempre el valor de la función.

Nota

Los nombres de las funciones en nemotécnica SIMATIC y en nemotécnica IEC son iguales.

Ejemplo de llamada a funciones de temporización

El ejemplo 17-5 ilustra la llamada a funciones de temporización:

```
FUNCTION_BLOCK FB2

VAR
      currTime    : S5time;
      biVal       : word;
      actFlag     : bool;
END VAR

BEGIN
currTime:= S_ODT (T_NO:=T10, S:=TRUE, TV:=T#1s,
                  R:=FALSE, BI:=biVal,
                  Q:=actFlag);

currTime:= S_ODTS (T_NO:=T11, S:=M0.0, TV:=T#1s,
                  R:=M0.1, BI:=biVal,
                  Q:=actFlag);

currTime:=S_OFFDT (T_NO:=T12, S:=E0.1 & actFlag,
                  TV:=T#1s, R:=FALSE, BI:=biVal,
                  Q:=actFlag);

currTime:= S_PEXT (T_NO:=T13, S:=TRUE,
                  TV:=T#1s, R:=E0.0, BI:=biVal,
                  Q:=actFlag);

currTime:= S_PULSE (T_NO:=T14, S:=TRUE,
                  TV:=T#1s, R:=FALSE, BI:=biVal,
                  Q:=actFlag);
END_FUNCTION_BLOCK
```

Ejemplo 17-5 Llamada a funciones de temporización

17.2.1 Introducción y valoración del valor de temporización

Resumen

Para introducir el valor predefinido o la valoración del resultado de la función BCD necesita la representación interna del valor de temporización (v. fig. 17-2).

Al actualizar el temporizador el valor de temporización se reduce en un unidad, a intervalos que vienen definidos por la base de tiempo. El valor de temporización se reduce hasta que se iguala a ”0”. El rango del temporizador abarca desde 0 a 9990 segundos.

Formato

La figura 17-2 muestra la representación interna del valor de temporización.

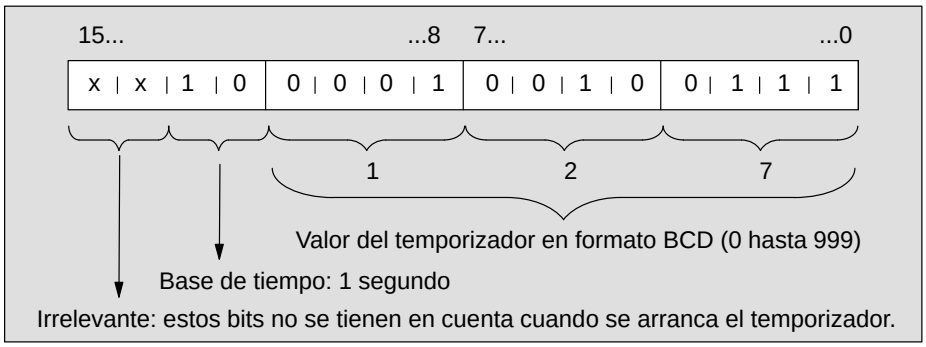


Figura 17-2 *Representación del valor de temporización*

Introducción

Puede cargar un valor de temporización predefinido con las siguientes representaciones:

- En representación escalonada: `TIME#aH_bbM_ccS_dddMS`
- En representación decimal: `TIME#2.4H`

Valoración

Puede evaluar el resultado en dos formatos diferentes:

- Como resultado de función (Tipo S5TIME): en formato BCD
- Como parámetro de salida (valor de temporización sin base de tiempo del tipo WORD): binario

Base de tiempo

Los bits 12 y 13 de la palabra del temporizador incluyen la base de tiempo para el código binario. La base de tiempo define el intervalo en el que el valor de temporización se reduce una unidad (v. tabla 17-9 y fig. 17-2). La base de tiempo más pequeña es 10 ms; la mayor, 10 s.

Tabla 17-9 Base de tiempo y código binario

Base de tiempo	Código binario para base de tiempo
10 ms	00
100 ms	01
1 s	10
10 s	11

Nota

Dado que los valores de temporización sólo se guardan en un intervalo de tiempo, los valores que no son múltiplo exacto del intervalo de tiempo se truncan.

Los valores cuya resolución es demasiado poco precisa para el rango deseado se redondean, de forma que se obtenga el rango deseado, **pero no** la resolución deseada.

17.2.2 Arrancar temporizador como impulso

Descripción

El tiempo máximo que la señal de salida permanece a ”1” es igual al tiempo de temporización programado ’t’.

Si durante el tiempo de funcionamiento del temporizador de la entrada se produce el estado de señal 0, la salida Q cambia a ”0” (es decir, el temporizador se detiene).

La figura 17-3 ilustra el funcionamiento del temporizador ”arrancar temporizador como impulso”.

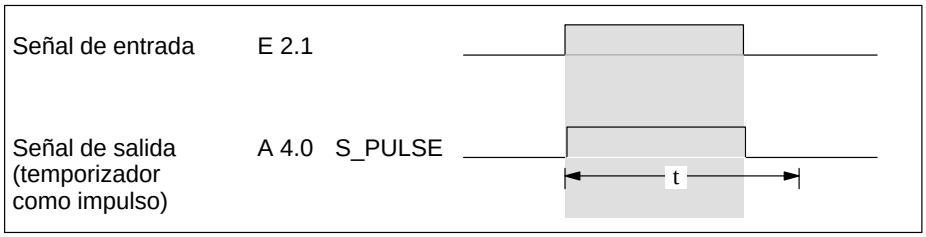


Figura 17-3 *Operación ”arrancar temporizador como impulso”*

Funcionamiento

Tabla 17-10 Funcionamiento de ”arrancar temporizador como impulso”

Operación	Funcionamiento
Arrancar temporizador	La operación ”arrancar temporizador como impulso” arranca un temporizador especificado cuando el estado de la señal de la entrada de arranque **S** cambia de ”0” a ”1”. Para habilitar el temporizador siempre se necesita un cambio de señal.
Definir tiempo de funcionamiento	El temporizador funciona on el valor indicado en la entrada **TV** hasta que concluye el tiempo programado y la entrada S=1.
Concluir anticipadamente el tiempo de funcionamiento	Si la entrada S cambia de ”1” a ”0” antes de que transcurra el valor de temporización, el temporizador se detiene.
Reinicializar	El temporizador se reinicializa cuando la entrada de inicialización – también llamada entrada de puesta a 0 – **R** cambia de ”0” a ”1” mientras funciona el temporizador. Este cambio también pone a ”0” el valor de temporización y la base de tiempo. El estado de la señal ”1” en la entrada R no influye si el temporizador no está en funcionamiento.
Consultar estado de señal	Cuando el temporizador está en funcionamiento y se consulta el estado de la señal en la salida **Q** se obtiene siempre el resultado ”1”. Si el tiempo de funcionamiento concluye anticipadamente, al consultar el estado de la señal en la salida Q se obtiene el resultado ”0”.
Consultar valor de temporización actual	El valor de temporización actual puede consultarse en la salida **BI** y mediante el valor de la función S_PULSE.

17.2.3 Arrancar temporizador como impulso prolongado

Descripción

La señal de salida permanece en ”1” durante el tiempo programado (’t’) independientemente del tiempo que la señal de entrada permanezca a ”1”. Un nuevo disparo del impulso de arranque provoca un nuevo recorrido del intervalo, de forma que el impulso de salida se prolonga por este tiempo (redisparo).

La figura 17-4 ilustra el funcionamiento de la operación ”arrancar temporizador como impulso prolongado”.

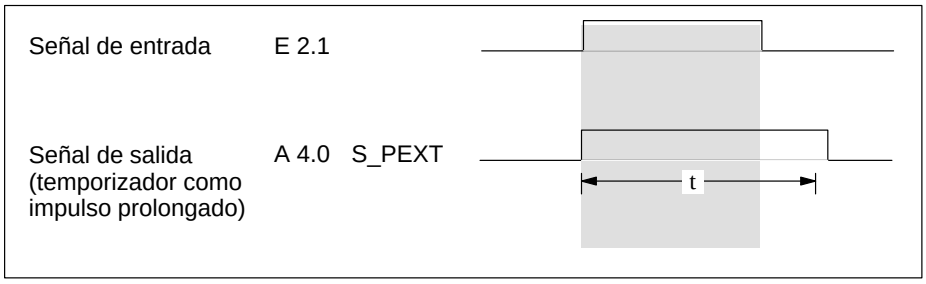


Figura 17-4 *Operación ”arrancar temporizador como impulso prolongado”*

Tabla 17-11 Funcionamiento de ”arrancar temporizador como impulso prolongado”

Funcionamiento

Operación	Funcionamiento
Arrancar temporizador	La operación ”arrancar temporizador como impulso prolongado” arranca un temporizador especificado cuando el estado de la señal de la entrada de arranque **S** cambia de ”0” a ”1”. Para habilitar el temporizador siempre se necesita un cambio de señal.
Rearranque del tiempo de funcionamiento	Si el estado de la señal de la entrada **S** cambia de nuevo a ”1” durante el tiempo de ejecución, el temporizador arranca de nuevo con el valor de temporización especificado.
Preajuste del tiempo de funcionamiento	El temporizador continúa funcionando con el valor especificado en la entrada **TV** hasta que concluye el tiempo programado.
Reinicializar	El temporizador se reinicializa cuando la entrada de reinicialización **R** cambia de ”0” a ”1” mientras funciona el temporizador. Este cambio reinicializa a ”0” también el valor de temporización y la base de tiempo. El estado de la señal ”1” en la entrada R no influye si el temporizador no está en funcionamiento.
Consultar estado de señal	Mientras el temporizador está en funcionamiento, si se consulta el estado de la señal en la salida **Q** se obtiene siempre el resultado ”1”, independientemente de la longitud de la señal de entrada.
Consultar valor de temporización actual	El valor de temporización actual puede consultarse en la salida **BI** y mediante el valor de la función S_PEXT.

17.2.4 Arrancar temporizador como retardo a la conexión

Descripción

La señal de salida cambia de ”0” a ”1” cuando ha transcurrido el tiempo programado y la señal de entrada continúa en ”1”. Es decir, la salida se conecta con retardo. Las señales de entrada cuyo intervalo es menor que el tiempo programado no aparecen en la salida.

La figura 17-5 ilustra el funcionamiento de la operación ”arrancar temporizador como retardo a la conexión”.

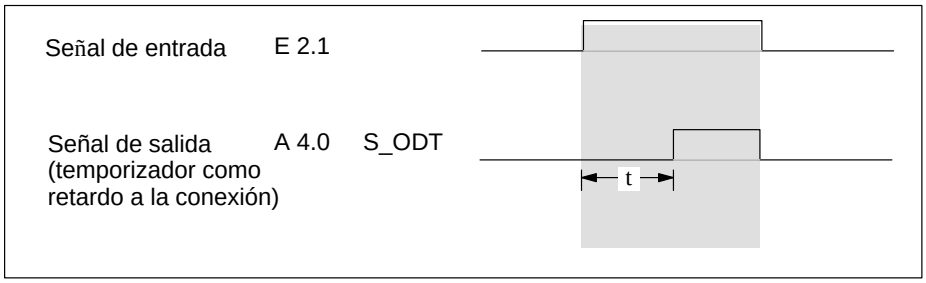


Figura 17-5 *Operación ”arrancar temporizador como retardo a la conexión”*

Funcionamiento

Tabla 17-12 Funcionamiento de ”arrancar temporizador como retardo a la conexión”

Operación	Funcionamiento
Arrancar temporizador	La operación ”arrancar temporizador como retardo a la conexión” arranca un temporizador especificando cuando el estado de la señal de la entrada de arranque **S** cambia de ”0” a ”1”. Para habilitar el temporizador siempre se necesita un cambio de señal.
Conservar temporizador	Si el estado de la señal en la entrada **S** cambia de ”1” a ”0” mientras el temporizador está en funcionamiento, éste se detiene.
Definir tiempo de funcionamiento	El temporizador continúa funcionando con el valor indicado en la entrada **TV** mientras el estado de la señal en la salida S = 1.
Reinicializar	El temporizador se reinicializa cuando la entrada de reinicialización **R** cambia de ”0” a ”1” mientras funciona el temporizador. Si R = 1 el temporizador también se reinicializa mientras el temporizador no está en funcionamiento.
Consultar estado de señal	Cuando se consulta el estado de la señal en la salida **Q** se obtiene el resultado ”1” si el tiempo se ha agotado sin existir error y en la entrada S continúa el valor ”1”. Si el temporizador se ha detenido, al consultar el estado de la señal se obtiene siempre ”0”. Al consultar el estado de la señal en la salida Q se obtiene también ”0” cuando el temporizador no está en funcionamiento y el RLO (resultado lógico, ver /232/) en la entrada S continúa siendo ”1”.
Consultar valor de temporización actual	El valor de temporización actual puede consultarse en la salida **BI** y mediante el valor de la función S_ODT.

17.2.5 Arrancar temporizador como retardo a la conexión con memoria

Descripción

La señal de salida cambia de ”0” a ”1” cuando ha transcurrido el tiempo programado, independientemente del tiempo que la señal de entrada permanezca en ”1”.

La figura 17-6 ilustra el funcionamiento de la operación ”arrancar temporizador como retardo a la conexión con memoria”.

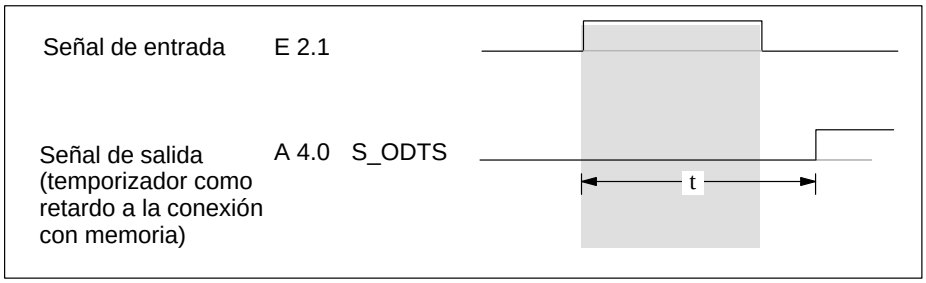


Figura 17-6 *Operación ”arrancar temporizador como retardo a la conexión con memoria”*

Tabla 17-13 Funcionamiento de ”arrancar temporizador como retardo a la conexión con memoria”

Funcionamiento

Operación	Funcionamiento
Arrancar temporizador	La operación ”arrancar temporizador como retardo a la conexión con memoria” arranca un temporizador especificado cuando el estado de la señal de la entrada de arranque **S** cambia de ”0” a ”1”. Para habilitar el temporizador siempre se necesita un cambio de señal.
Rearrancar temporizador	El temporizador rearranca con el valor especificado cuando la entrada **S** cambia de ”0” a ”1” mientras el temporizador esté en funcionamiento.
Definir tiempo de funcionamiento	El temporizador continúa funcionando con el valor indicado en la entrada **TV** cuando el estado de la señal en la entrada S cambia a ”0” antes de que se agote el temporizador.
Reinicializar	Cuando la entrada de reinicialización **R** cambia de ”0” a ”1”, el temporizador se reinicializa independientemente del RLO (resultado lógico ver /232/) en la entrada S.
Consultar estado de señal	Al consultar el estado de señal en la salida **Q** se obtiene el resultado ”1” después de agotado al temporizador, independientemente del estado de la señal en la entrada S.
Consultar valor de temporización actual	El valor de temporización actual puede consultarse en la salida **BI** y mediante el valor de la función S_ODTS.

17.2.6 Arrancar temporizador como retardo a la desconexión

Descripción

Cuando en la entrada de arranque S se produce un cambio del estado de señal de ”0” a ”1”, en la salida Q aparece el estado ”1”. Cuando el estado de la entrada de arranque pasa de ”1” a ”0”, arranca el temporizador. La salida sólo cambia al estado de señal ”0” después de transcurrido el intervalo. De esta forma la salida se desconecta con retardo.

La figura 17-7 ilustra el funcionamiento de la operación ”arrancar temporizador como retardo a la desconexión”.

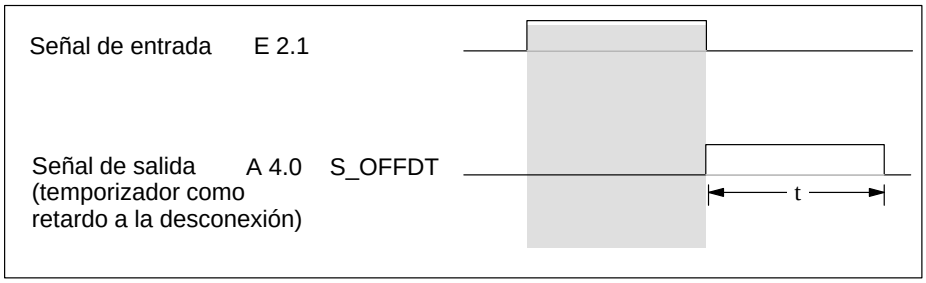


Figura 17-7 *Operación ”arrancar temporizador como retardo a la desconexión”*

Funcionamiento

Tabla 17-14 Funcionamiento de ”arrancar temporizador como retardo a la desconexión”

Operación	Funcionamiento
Arrancar temporizador	La operación ”arrancar temporizador como retardo a la desconexión” arranca el temporizador especificado cuando en la entrada de arranque **S** el estado de señal cambia de ”1” a ”0”. Siempre se necesita un cambio de señal para habilitar el temporizador.
Rearrancar temporizador	El temporizador rearranca cuando el estado de la señal en la entrada **S** vuelve a cambiar de ”1” a ”0” (p.ej., después de reinicialización).
Definir tiempo de funcionamiento	El temporizador funciona con el valor especificado en la entrada **TV**.
Reinicializar	Si la entrada de reinicialización **R** cambia de ”0” a ”1” mientras el temporizador está en funcionamiento, se reinicializa el temporizador.
Consultar estado de señal	Al consultar el estado de señal en la salida **Q** se obtiene el resultado ”1” cuando el estado de señal en la salida S = 1 o cuando el temporizador está en funcionamiento.
Consultar valor de temporización actual	El valor de temporización actual puede consultarse en la salida **BI** y mediante el valor de la función S_OFFDT.

17.2.7 Ejemplo de programa para un impulso prolongado

Ejemplo S_PEXT

El ejemplo 17-6 muestra un programa para aplicar la función de temporización ”impulso prolongado”:

```
FUNCTION_BLOCK TEMPORIZADOR
VAR_INPUT
 MITEMPORIZADOR: TIMER;
END_VAR
VAR_OUTPUT
 RESULTADO: S5TIME;
END_VAR
VAR
 AJUSTAR    : BOOL;
 INICIALIZAR: BOOL;
 VALOR_BCD  : S5TIME;//Base de tiempo y valor rest.
                     //Codificación BCD
 VALOR_BINARIO    : WORD; //Valor de temporiz. binario
 PREDEFINICION    : S5TIME;
END_VAR
BEGIN
 A0.0:= 1;
 AJUSTAR:= E0.0;
 INICIALIZAR:= E0.1;
 PREDEFINICION:= T#25S;
```

```
 VALOR_BCD:= S_PEXT(T_NO:= MITEMPORIZADOR,
                    S   := AJUSTAR,
                    TV  := PREDEFINICION,
                    R   := INICIALIZAR,
                    BI  := VALOR_BINARIO,
                    Q   := A0.7);
```

```
RESULTADO:=VALOR_BCD;//Continúa procesamiento como
                     //parámetro de salida
AW4:= VALOR_BINARIO //A visualización en salida
END_FUNCTION_BLOCK
```

Ejemplo 17-6 Ejemplo de función de temporización

17.2.8 Selección de la operación correcta

La figura 17-8 ofrece un resumen de los cinco temporizadores diferentes que se han descrito en esta sección. Este resumen pretende ayudarle a que seleccione el temporizador más adecuado a sus fines.

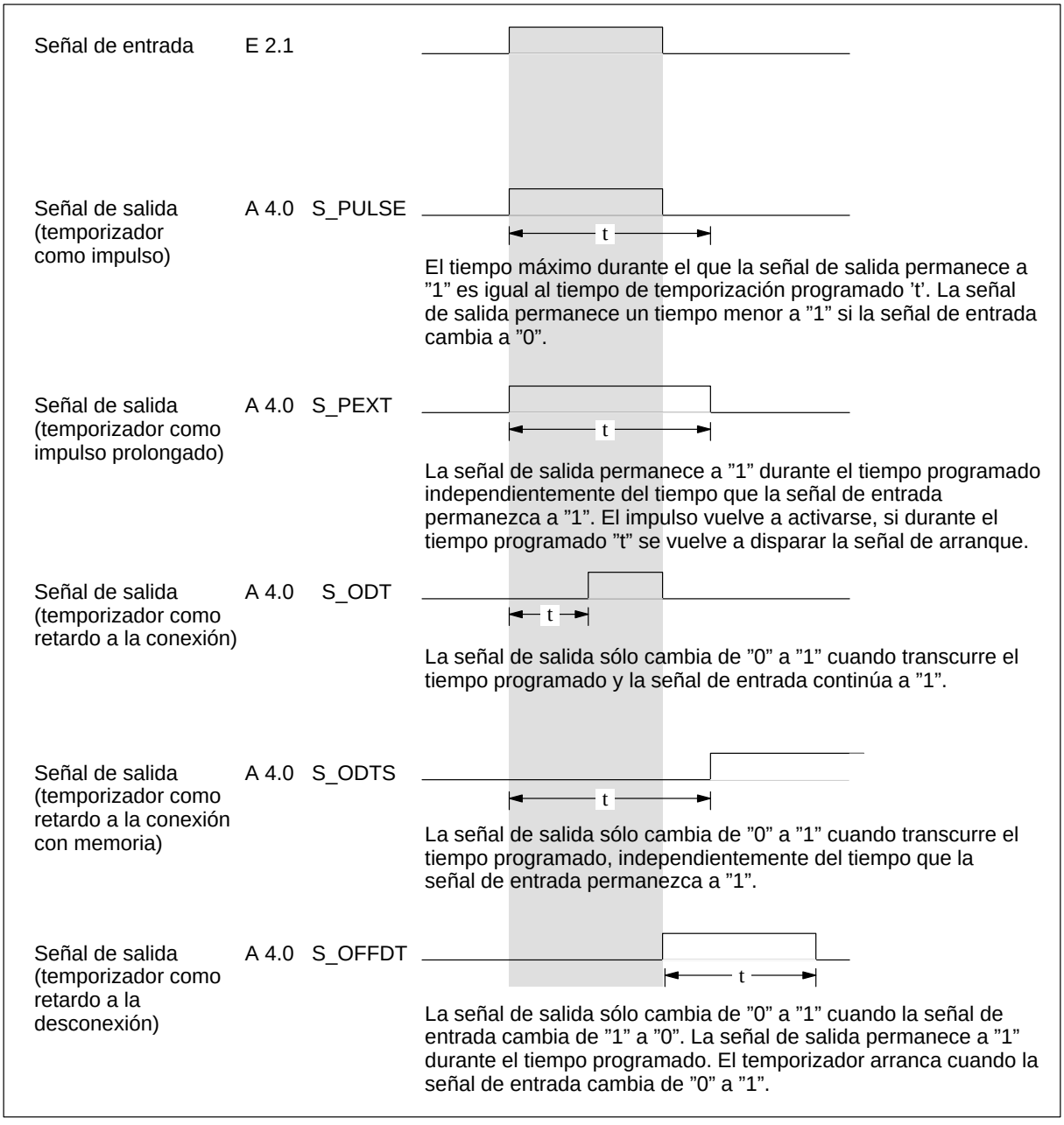


Figura 17-8 *Selección del temporizador correcto*

Funciones estándar de SCL

18

Resumen

SCL pone a su disposición una serie de funciones estándar para solucionar las tareas que se presentan frecuentemente; usted puede llamar a dichas funciones estándar en sus bloques SCL.

Indice del capítulo

18.1 Conversión de tipos de datos

Resumen

Cuando relacione lógicamente dos operandos de tipos de datos diferentes o asigne expresiones a las variables, debe tener en cuenta la compatibilidad entre los diferentes tipos de datos que intervienen. En los siguientes casos cabe esperar resultados erróneos:

- al cambiar a otra categoría de tipo de datos, por ejemplo, de un tipo de datos de bits a un tipo de datos numérico;
- al cambiar dentro de una misma categoría de tipos de datos. cuando el tipo de datos de destino sea de menor rango que el tipo de datos fuente.

Por ello en estos casos debe realizar una conversión **explícita** del tipo de datos. En el apartado 18.2 encontrará la información necesaria.

En los demás casos el compilador fuerza una conversión automática a un formato común. Es lo que se denomina conversión **implícita** de tipo de datos.

Conversión implícita de tipo de datos

Dentro de las categorías de tipos de dato auxiliares definidas en la tabla 18-1, el compilador realiza conversiones implícitas de tipos de datos en el orden indicado. En todos los casos el formato común que se define para dos operandos es el tipo estándar más pequeño, cuyo rango incluye a los dos operandos. Así, por ejemplo, el formato común para ”byte” y ”entero” es ”entero”.

Tenga también en cuenta que en una conversión de tipo de datos dentro de la categoría `ANY_BIT` los bits iniciales se convierten en 0.

Tabla 18-1 Orden de las conversiones implícitas de tipo de datos

Categorías	Orden de la conversión
ANY_BIT	`BOOL` ⇒ `BYTE` ⇒ `WORD` ⇒ `DWORD`
ANY_NUM	`INT` ⇒ `DINT` ⇒ `REAL`

El siguiente ejemplo pretende ilustrar la conversión implícita de tipos de datos:

```
FUNCTION_BLOCK FB10
VAR
      PID_REGULADOR_1:BYTE;
      PID_REGULADOR_2:WORD;
END_VAR
BEGIN
IF (PID_REGULADOR_1 <> PID_REGULADOR_2) THEN
      // ...
(* En la condición de la instrucción IF / THEN
superior, PID_REGULADOR_1 se convierte implícita-
mente en una variable del tipo de datos WORD. *)
END_FUNCTION_BLOCK
```

Ejemplo 18-1 Conversión implícita de tipo de datos

18.2 Funciones estándar para conversiones de tipo de datos

Conversión explícita de tipo de datos

La conversión explícita de tipo de datos puede realizarla con funciones estándar. Estas funciones estándar puede consultarlas en las tablas 18-2 y 18-3.

Llamada a la función

En el capítulo 16 encontrará una descripción general de una llamada a función.

- Parámetro de entrada:

  Toda función para conversión de un tipo de dato tiene exactamente un parámetro de entrada llamado `IN`. Como se trata de una función con un solo parámetro, es suficiente especificar el parámetro actual.

- Valor de la función

  El resultado es siempre el valor de la función. Las dos tablas muestran las reglas que se aplican para convertir una fecha. La tabla 18-3 indica si la función correspondiente influye sobre la marca `OK`.

- Denominación

  Como los tipos de datos del parámetro de entrada y del valor de la función derivan de sus nombres respectivos de función, no se han incluido específicamente en la lista de las tablas 18-2 y 18-3: por ejemplo, en la función `BOOL_TO_BYTE` el tipo de datos del parámetro de entrada es `BOOL`, y el tipo de datos del valor de la función, `BYTE`.

Lista de funciones de conversión (categoría A)

La tabla 18-2 representa las funciones de conversión de tipo de datos de categoría A. Estas funciones son establecidas implícitamente por el compilador o puede especificarlas usted explícitamente. El resultado es siempre definido.

Tabla 18-2 Funciones de conversión de tipo de datos de la categoría A

Nombre de la función	Regla de conversión
BOOL_TO_BYTE	Completar con ceros a la izquierda.
BOOL_TO_DWORD	
BOOL_TO_WORD	
BYTE_TO_DWORD	
BYTE_TO_WORD	
CHAR_TO_STRING	Transformación en una cadena (de longitud 1) que contiene el mismo carácter.
DINT_TO_REAL	Transformación en `REAL` de acuerdo con la norma IEEE. El valor puede cambiar debido al diferente grado de precisión de `REAL`.
INT_TO_DINT	La palabra de mayor rango del valor de la función se completa con `16#FFFF` en caso de tratarse de un parámetro de entrada negativo, y con ceros en caso contrario. El valor no varía.
INT_TO_REAL	Transformación en `REAL` de acuerdo con la norma IEEE. El valor no varía.
WORD_TO_DWORD	Completar con ceros a la izquierda.

Lista de funciones de conversión (categoría B)

La tabla 18-3 representa las funciones de conversión de tipos de datos de categoría B. Estas funciones debe especificarlas explícitamente. El resultado también puede ser indeterminado, si el tamaño del tipo de datos de destino es insuficiente.

Puede revisar este caso usted mismo anteponiendo una comprobación de los límites, o haciendo que el sistema efectúe una revisión activando la opción OK en la compilación. Entonces, en los casos en que el resultado es indeterminado el sistema coloca la variable OK en posición FALSE. La valoración debe efectuarla usted mismo.

Tabla 18-3 Funciones de conversión de tipo de datos de categoría B

Nombre de la función	Regla de conversión	OK
BYTE_TO_BOOL	Copiar el bit de menor valor.	S
BYTE_TO_CHAR	Aceptar la cadena de bits.	N
CHAR_TO_BYTE	Aceptar la cadena de bits.	N
CHAR_TO_INT	La cadena de bits existente en el parámetro de entrada se registra en el byte de menor valor del valor de la función. El byte de mayor valor se completa con ceros.	N
DATE_TO_DINT	Aceptar la cadena de bits.	N
DINT_TO_DATE	Aceptar la cadena de bits.	S
DINT_TO_DWORD	Aceptar la cadena de bits.	N
DINT_TO_INT	Copiar el bit de signo; el dato existente en el parámetro de entrada se interpreta como tipo de dato s INT. Si el valor es menor de –32_768 o mayor de 32_767, la variable OK se coloca en la posición FALSE.	S
DINT_TO_TIME	Aceptar la cadena de bits.	N
DINT_TO_TOD	Aceptar la cadena de bits.	S
DWORD_TO_BOOL	Copia del bit de menor valor	S
DWORD_TO_BYTE	Copia de los 8 bits de menor valor	S
DWORD_TO_DINT	Aceptar la cadena de bits.	N
DWORD_TO_REAL	Aceptar la cadena de bits.	N
DWORD_TO_WORD	Copiar los 16 bits de menor valor	S
INT_TO_CHAR	Aceptar la cadena de bits.	S
INT_TO_WORD	Aceptar la cadena de bits.	N
REAL_TO_DINT	Redondear a DINT el valor REAL. de IEEE. Cuando el valor es menor de –2_147_483_648 o mayor de 2_147_483_647, la variable OK se coloca en posición FALSE.	S
REAL_TO_DWORD	Aceptar la cadena de bits.	N
REAL_TO_INT	Redondear a INT el valor IEEE REAL de IEEE. Si el valor es menor de –32_768 o mayor de 32_767, la variable OK pasa a estado FALSE.	S
STRING_TO_CHAR	Copiar el primer carácter de la cadena. Si la STRING no tiene la longitud 1, la variable OK pasa a posición FALSE.	S

Tabla 18-3 Funciones de conversión de tipo de datos de categoría B, continuación

Nombre de la función	Regla de conversión	OK
TIME_TO_DINT	Aceptar la cadena de bits.	N
TOD_TO_DINT	Aceptar la cadena de bits.	N
WORD_TO_BOOL	Copia del bit de menor valor	S
WORD_TO_BYTE	Copia de los 8 bits de menor valor.	S
WORD_TO_INT	Aceptación de la cadena de bits.	N
WORD_TO_BLOCK_DB	La configuración binaria de WORD se interpreta como número del bloque de datos.	N
BLOCK_DB_TO_WORD	El número del bloque de datos se interpreta como configuración binaria de WORD.	

Nota

Además tiene la posibilidad de utilizar **funciones IEC** para efectuar conversiones de tipo de datos. En **/235/** encontrará información sobre cada una de estas funciones IEC.

Ejemplos de conversión explícita

En el ejemplo 18-2 es necesario efectuar una conversión explícita, puesto que el tipo de datos de destino es de menor rango que el tipo de dato fuente.

```
FUNCTION_BLOCK FB10
VAR
      INTERRUPTOR : INT;
      REGULADOR   : DINT;
END_VAR

BEGIN
INTERRUPTOR := DINT_TO_INT (REGULADOR);
(* INT es de menor rango que DINT *)
// ...
END_FUNCTION_BLOCK
```

Ejemplo 18-2 Tipo de datos de destino que no concuerda con el tipo de datos fuente

En el ejemplo 18-3 es necesaria una conversión, puesto que no está permitido el tipo de datos REAL para una expresión aritmética con el operador MOD.

```
FUNCTION_BLOCK FB20
    VAR
      valor_int:INT:=17;
      KONV2 := INT;
    END_VAR

BEGIN
    KONV2 := valor_int MOD REAL_TO_INT (2.3);
    (* MOD sólo debe aplicarse a datos del tipo
    INT o DINT. *)
// ...
END_FUNCTION_BLOCK
```

Ejemplo 18-3 Conversión debida a tipo de datos no permitido

En el ejemplo 18-4 es necesaria una conversión, puesto que no está presente el tipo de datos correcto para un operador lógico. El operador NOT sólo puede aplicarse a datos del tipo BOOL, BYTE, WORD o DWORD.

```
FUNCTION_BLOCK FB30
    VAR
      valor_int: INT := 17;
      KONV1    : WORD;
    END_VAR

BEGIN
    KONV1 := NOT INT_TO_WORD(valor_int);
    (* NOT no debe aplicarse a datos del
    tipo INT*)
// ...
END_FUNCTION_BLOCK
```

Ejemplo 18-4 Conversión debida a tipo de datos erróneo

El ejemplo 18-5 muestra la conversión de entradas/salidas para la periferia:

```
FUNCTION_BLOCK FB40
 VAR
 ent_radio        : WORD;
 radio            : INT;
 END_VAR

BEGIN
 ent_radio     := EB0;
 radio         := WORD_TO_INT(ent_radio);
(* Conversión por cambio a otra categoría de tipo.
El valor viene de la entrada y se convierte para su
cálculo posterior*)
 radio := Radio(superficie:= datos_circulo_superficie);
 AB0   := WORD_TO_BYTE(INT_TO_WORD(radio));
(* El radio se recalcula a partir de la superficie, y
es un entero. Para la salida primero se convierte
en otra categoría de tipo de datos (INT_TO_WORD) y
después a un tipo de menor rango (WORD_TO_BYTE) *)
    // ...
END_FUNCTION_BLOCK
```

Ejemplo 18-5 Conversión en entradas y salidas

Funciones de redondeo y truncado

Las funciones de redondeo y truncado de números se consideran también funciones de conversión de tipo de datos. La tabla 18-4 indica los nombres, los tipos de datos (para los parámetros de entrada y el valor de la función) y las tareas que desempeñan estas funciones.

Tabla 18-4 Funciones de redondeo y truncado

Nombre de la función	Tipo de dato del parámetro de entrada	Tipo de dato del valor de la función	Tarea
ROUND	REAL	DINT	Redondeo (formar un número DINT)
TRUNC	REAL	DINT	Truncado (formar un número DINT)

Los siguientes ejemplos demuestran los diferentes efectos:

```
• ROUND (3.14)     // En este caso se redondea por defecto,
                   // por lo tanto el resultado es : 3

• ROUND (3.56)     // En este caso se redondea por exceso,
                   // por lo tanto el resultado es : 4

• TRUNC (3.14)     // En este caso se trunca,
                   // por lo tanto el resultado es : 3

• TRUNC (3.56)     // En este caso se trunca,
                   // por lo tanto el resultado es : 3
```

18.3 Funciones estándar numéricas

Funcionalidad

Cada función estándar numérica tiene un parámetro de entrada. El resultado es siempre el valor de la función. Cada una de las tablas 18-5, 18-6 y 18-7 especifica un número de funciones estándar numéricas indicando los nombres de la función y los tipos de datos. El tipo de datos ANY_NUM representa a INT, DINT o REAL.

Lista de las funciones generales

Son las funciones para el calcular el valor absoluto, el cuadrado o la raíz cuadrada de una magnitud.

Tabla 18-5 Funciones generales

Nombre de la función	Tipo de dato del parámetro de entrada	Tipo de dato del valor de la función	Descripción
ABS	ANY_NUM[1)]	ANY_NUM	Valor absoluto
SQR	ANY_NUM[1)]	REAL	Cuadrado
SQRT	ANY_NUM[1)]	REAL	Raíz

1) *Tenga en cuenta que a nivel interno los parámetros de entrada de tipo ANY_NUM se transforman en parámetros reales.*

Lista de las funciones logarítmicas

Las funciones logarítmicas son funciones para el cálculo de un valor exponencial o de un logaritmo de una magnitud.

Tabla 18-6 Funciones logarítmicas

Nombre de la función	Tipo de dato del parámetro de entrada	Tipo de dato del valor de la función	Descripción
EXP	ANY_NUM[1)]	REAL	e elevado a `IN`
EXPD	ANY_NUM[1)]	REAL	10 elevado a `IN`
LN	ANY_NUM[1)]	REAL	Logaritmo natural
LOG	ANY_NUM[1)]	REAL	Logaritmo en base diez

1) *Tenga en cuenta que a nivel interno los parámetros de entrada del tipo ANY_NUM se transforman en variables reales.*

Nota

Además tiene la posibilidad de utilizar funciones **IEC** como función numérica estándar. En este caso copie la función deseada de la librería de STEP 7 STDLIBS\IEC en el directorio de su programa. En **/235/** encontrará información sobre cada una de estas funciones IEC.

Lista de las funciones trigonométricas

Las funciones trigonométricas representadas en la tabla 18-7 calculan magnitudes de arcos.

Tabla 18-7 Funciones trigonométricas

Nombre de la función	Tipo de dato del parámetro de entrada	Tipo de dato del valor de la función	Descripción
ACOS	ANY_NUM[1)]	REAL	Arco coseno
ASIN	ANY_NUM[1)]	REAL	Arco seno
ATAN	ANY_NUM[1)]	REAL	Arco tangente
COS	ANY_NUM[1)]	REAL	Coseno
SIN	ANY_NUM[1)]	REAL	Seno
TAN	ANY_NUM[1)]	REAL	Tangente

1) *Tenga en cuenta que a nivel interno los parámetros de entrada del tipo ANY_NUM se transforman en variables reales.*

Ejemplos

La tabla 18-8 muestra llamadas posibles a funciones estándar numéricas y sus resultados respectivos:

Tabla 18-8 Llamada s posibles a funciones estándar numéricas

Llamada	Resultado
RESULTADO := ABS (-5);	5
RESULTADO := SQRT (81.0);	9
RESULTADO := SQR (23);	529
RESULTADO := EXP (4.1);	60.340 ...
RESULTADO := EXPD (3);	1_000
RESULTADO := LN (2.718_281);	1
RESULTADO := LOG (245);	2.389_166 ...
PI := 3. 141 592; RESULTADO := SIN (PI / 6);	0.5
RESULTADO := ACOS (0.5);	1.047_197 (=PI / 3)

18.4 Funciones estándar de cadena de bits

Funcionalidad

Cada función estándar de cadena de bits tiene dos parámetros de entrada que se denominan IN o N. El resultado es siempre el valor de la función. La tabla 18-9 muestra los nombres del valor de las funciones y los tipos de datos de los dos parámetros de entrada, además del valor de la función. Significan lo siguiente:

- Parámetro de entrada IN: búfer en el que se ejecutan las operaciones de desplazamiento de bits.
- Parámetro de entrada N: número de rotaciones en las funciones ROL y ROR del búfer de circulación, o el número de posiciones que deben desplazarse en SHL y SHR.

Lista de las funciones

La tabla 18-9 muestra las posibles funciones estándar de cadena de bits.

Tabla 18-9 Funciones estándar de cadena de bits

Nombre de la función	Tipo de dato del parámetro de entrada `IN`	Tipo de dato del parámetro de entrada `N`	Tipo de dato del valor de la función	Tarea
ROL	BOOL	INT	BOOL	El valor existente en el parámetro `IN` rota hacia la izquierda tanto puestos como indique el contenido del parámetro `N`.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
ROR	BOOL	INT	BOOL	El valor existente en el parámetro `IN` rota hacia la derecha tantos puestos como indique el contenido del parámetro `N`.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHL	BOOL	INT	BOOL	En el valor existente en el parámetro `IN` se desplazan hacia la izquierda tantos bits como indique el contenido del parámetro `N` y se sustituye por 0 el mismo número de posiciones en el lado derecho.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHR	BOOL	INT	BOOL	En el valor existente en el parámetro `IN` se desplazan hacia la derecha tantos bits como indique el contenido del parámetro `N` y se reemplazan por 0 el mismo número de posiciones en el lado izquierdo.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	

Nota

Además tiene la posibilidad de utilizar **funciones IEC** para operaciones de cadena de bits. En este caso copie la función deseada desde la librería STEP 7 STDLIBS\IEC en el directorio de su programa. En **/235/** encontrará más información sobre cada una de las funciones IEC.

Ejemplos

La tabla 18-10 muestra posibles llamadas de funciones estándar de cadenas de bits y sus resultados correspondientes.

Tabla 18-10 Llamadas de funciones estándar de cadenas de bits

Llamada	RESULTADO
RESULTADO := ROL (IN:=2#1101_0011, N:=5); // IN := 211 decimal	2#0111_1010 (= 122 decimal)
RESULTADO := ROR (IN:=2#1101_0011, N:=2); // IN := 211 decimal	2#1111_0100 (= 244 decimal)
RESULTADO := SHL (IN:=2#1101_0011, N:=3); // IN := 211 decimal	2#1001_1000 (= 152 decimal)
RESULTADO := SHR (IN:=2#1101_0011, N:=2); // IN := 211 decimal	2#0011_0100 (= 52 decimal)

Interface de llamada

19

Resumen

Las CPU de S7 tienen en el sistema operativo funciones del sistema y funciones estándar integradas que usted puede utilizar al programar en SCL. En concreto son las siguientes:

- Bloques de organización (OB)
- Funciones del sistema (SFC)
- Bloques de función del sistema (SFB)

Indice del capítulo

19.1 Interface de llamada

Resumen

Puede llamar a los bloques absoluta o simbólicamente. Para ello necesitará o el nombre simbólico (que debe estar declarado en la lista de señales), o el número para la identificación absoluta del bloque.

Al efectuar las llamadas debe asignar a los **parámetros formales** (cuyos nombres y tipo de datos se establecieron al crear el bloque parametrizable) los **parámetros actuales**, con cuyos valores de bloque trabaja su programa durante el tiempo de ejecución.

Todas las informaciones que necesita al respecto se encuentran descritas en **/235/**. El manual le ofrece un resumen de las funciones básicas disponibles en S7 y, como información de consulta, descripciones detalladas de los interfaces para utilizar en su programa de usuario.

Ejemplo: SFC 31

Las siguientes líneas de comando le permiten llamar a la función del sistema SFC31 (consultar alarma horaria):

```
FUNCTION_BLOCK FY20

VAR
      Resultado: INT;
END_VAR
BEGIN
      // ...
      Resultado:= SFC31 (OB_NR:= 10,ESTADO:= MW100 );
      // ...
      // ...
END_FUNCTION_BLOCK
```

Ejemplo 19-1 Consulta de alarma horaria

Resultados

El valor de la función es del tipo entero. Si su valor es mayor o igual que 0, el bloque se ha ejecutado sin errores; si el valor es menor que 0 se ha producido un error. Después de efectuar la llamada puede valorar el parámetro de salida ENO definido implícitamente.

Llamada condicional

Para efectuar una llamada condicional primero debe asignar 0 a los **parámetros de entrada EN** predefinidos (por ejemplo, mediante la entrada E0.3), y después no es necesario llamar al bloque. Cuando EN sea 1 se llamará a la función. En este caso el **parámetro de salida ENO** pasa a ”1” (en caso contrario, a ”0”) en caso de que durante la ejecución del bloque no se produzca ningún error.

Nota

En los bloques de función o en los bloques de función del sistema las informaciones que pueden transmitirse con el valor de la función se guardan en el parámetro de salida. A continuación se valoran mediante el bloque de datos de instancia. Consulte más información en el capítulo 16.

19.2 Interface de transmisión a OB

Bloques de organización

Los bloques de organización constituyen el interface entre el sistema operativo de la CPU y el programa de usuario. Con ayuda de los OB puede ejecutar de forma dirigida secciones del programa:

- al arrancar la CPU,
- en ejecución cíclica o por ciclos de tiempo,
- a determinadas horas del día o en determinados días,
- después de que transcurra un tiempo predefinido,
- cuando aparezcan errores,
- cuando se produzcan alarmas del proceso o de comunicación.

Los bloques de organización se ejecutan siguiendo el orden de prioridad asignado por usted.

Bloques de organización disponibles

No todas las CPU pueden ejecutar todos los OB disponibles en S7. En las hojas de datos de su CPU podrá consultar los OB de que dispone.

Información adicional

En la ayuda online podrá consultar más información. También podrá consultar los siguientes manuales:

- **/70/** Manual:
  Autómata programable S7-300, Configuración, instalación y datos de las CPU
  Este manual incluye las hojas de datos que describen las prestaciones de las diferentes CPU del S7-300. Se incluyen también los posibles sucesos para el arranque de cada OB.
- **/100/** Manual:
  Sistemas de automatización S7-400, M7-400, Configuración e instalación
  Este manual incluye las hojas de datos que describen las prestaciones de las diferentes CPU del S7-400. También se incluyen los posibles sucesos para el arranque de cada OB.

Anexos

Descripción formal del lenguaje

A

Descripción del lenguaje SCL

La base para la descripción del lenguaje SCL dentro de cada uno de los capítulos son los diagramas sintácticos, que le proporcionarán una vision de conjunto de la estructura sintáctica (es decir, gramatical) de SCL. En los anexos B y C encontrará un resumen completo de todos los diagramas, con los elementos de lenguaje correspondientes.

Indice del capítulo

A.1 Resumen

¿Qué es un diagrama sintáctico?

El diagrama sintáctico es una representación gráfica de la estructura del lenguaje. La estructura se describe mediante una secuencia de reglas. Una regla puede basarse en reglas introducidas previamente.

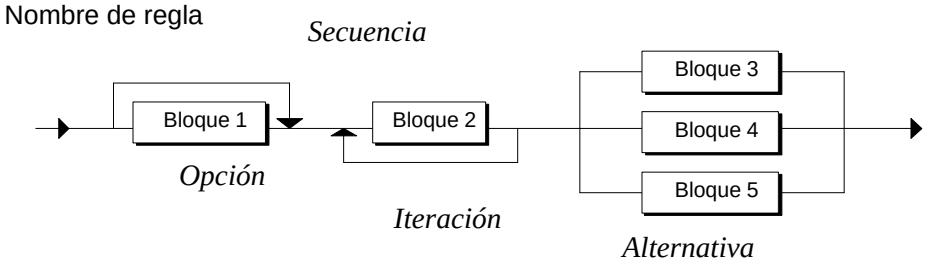


Figura A-1 *Ejemplo de un diagrama sintáctico*

El diagrama sintáctico se lee de izquierda a derecha. Deben observarse las estructuras que tienen las reglas:

- Secuencia: secuencia de bloques
- Opción: rama que puede saltarse
- Iteración: repetición de ramas
- Alternativa: ramificación

¿Qué clases de bloques existen?

Un bloque es un elemento fundamental o un elemento que a su vez está compuesto de bloques. La figura siguiente muestra los tipos de símbolos que corresponden a los bloques:

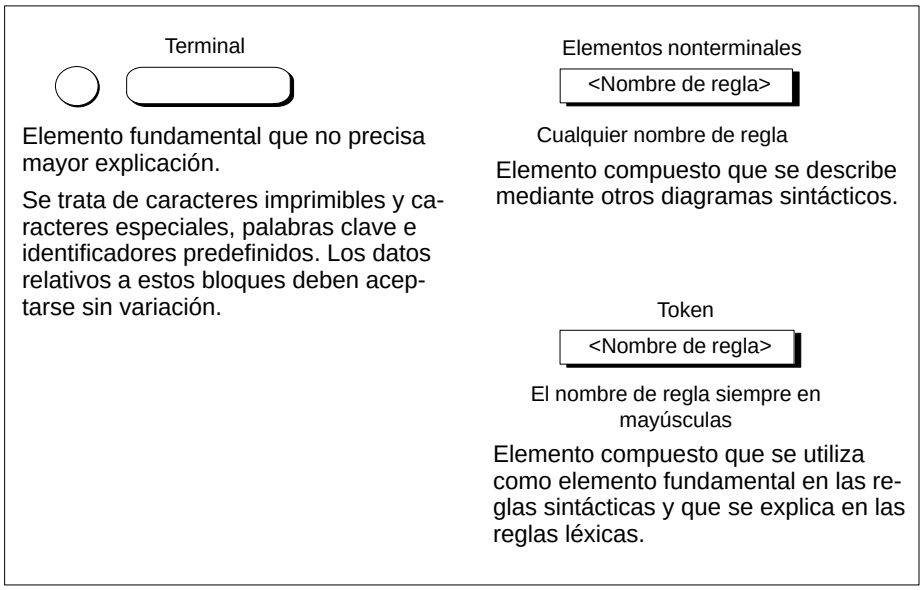


Figura A-2 *Tipos de símbolos de los bloques*

Reglas

Las reglas que puede utilizar para estructurar su programa en SCL se clasifican en dos niveles: reglas **léxicas** y reglas **sintácticas**.

Reglas léxicas

Las reglas léxicas describen la estructura de los elementos (token) que se procesan durante el análisis léxico del compilador. Por ello, en la escritura no hay libertad de formato y deben respetarse estrictamente las reglas. En particular esto significa que:

- no está permitida la inserción de signos de formateado;
- no pueden insertarse bloques de comentario ni líneas de comentario;
- no pueden insertarse atributos para los identificadores.

IDENTIFICADOR

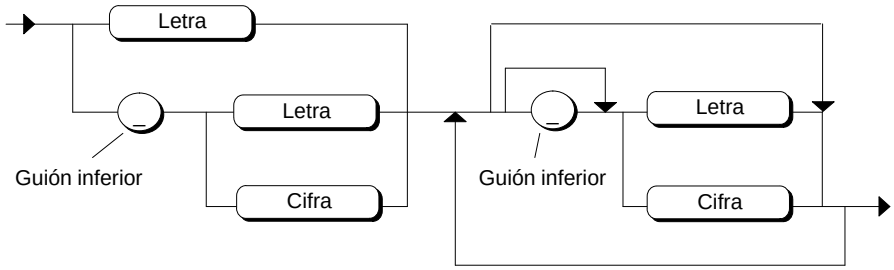


Figura A-3 *Ejemplo de regla léxica*

El ejemplo muestra la regla léxica IDENTIFICADOR. Describe la estructura de un identificador (nombre), p.ej.:

```
ARRAY_MEDICION_12
VALOR_TEORICO_B_1
```

Reglas sintácticas

En las reglas sintácticas se describe la estructura de SCL, basándose en las reglas léxicas. Dentro de estas reglas dispone de libertad de formato para elaborar su programa SCL:

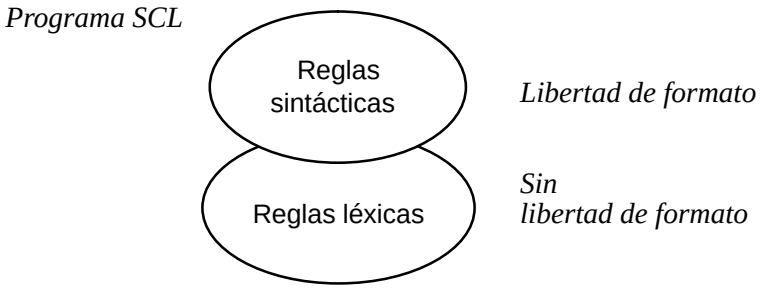


Figura A-4 *Niveles de las reglas y libertad de formato*

Consideraciones formales

A cada regla se le antepone un nombre de regla. Cuando la regla se utiliza dentro de una regla superior, el nombre de la regla aparece encerrado en un rectángulo.

Si el nombre de la regla está escrito en mayúsculas se trata de un token, que se describe en las reglas léxicas.

Semántica

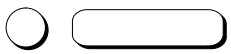
En las reglas sólo puede representarse la estructura formal del lenguaje, de la cual no siempre puede deducirse su significado, es decir, su semántica. Por ello, en puntos importantes se escriben informaciones adicionales junto a las reglas. Ejemplos:

- con elementos de igual clase y diferente significado se indica un nombre adicional: p.ej., en la regla de indicación de fecha se indica año, mes o día en SECUENCIA DE CIFRAS DECIMALES. El nombre indica a la utilización.
- Las restricciones importantes se hacen constar junto a las reglas: p.ej., con un símbolo que debe definirse en la tabla de símbolos.

A.2 Resumen de terminales

Definición

Un terminal es un elemento fundamental que no se explica mediante otras reglas, sino verbalmente. En los diagramas sintácticos aparecerán con el siguiente símbolo:



El elemento terminal se representa mediante un óvalo o un círculo. Se indicará directamente el contenido o el nombre (en letras mayúsculas).

Define la cantidad de caracteres ASCII que pueden utilizarse.

Figura A-5 *Símbolos de terminales*

Resumen

En los apartados A.3 y A.4 se exponen los tipos de utilización de cada uno de los caracteres. Son los siguientes:

- Letras, cifras, caracteres imprimibles y caracteres especiales
- Signos de formateado y signos de separación en las reglas léxicas
- Prefijos de literales
- Signos de formateado y signos de separación en las reglas sintácticas
- Operadores

En los apartados A.5 y A.6 se exponen palabras clave e identificadores predefinidos de cadenas de caracteres. Las tablas están ordenadas alfabéticamente. Cuando existan diferencias entre la nemotécnica alemana y la nemotécnica internacional se indicará la nemotécnica internacional correspondiente:

- Palabras clave e identificadores predefinidos
- Identificadores de operando y palabras clave de bloques

A.3 Terminales de las reglas léxicas

Resumen

En las tablas siguientes los terminales se especifican indicando el conjunto de elementos del juego de caracteres ASCII.

Letras y cifras

Las letras y cifras son los caracteres utilizados principalmente. El IDENTIFICADOR consta, p.ej., de letras, cifras y guiones inferiores.

Tabla A-1 Letras y cifras

Caracteres	Subgrupo	Elementos del juego de caracteres
Letra	Mayúscula	A.. Z
	Minúscula	a.. z
Cifra	Cifra decimal	0.. 9
Cifra octal	Cifra octal	0.. 7
Cifra hexadecimal	Cifra hexadecimal	0.. 9, A.. F, a.. f
Bit	Cifra binaria	0, 1

Caracteres imprimibles y caracteres especiales

El juego de caracteres ASCII ampliado completo puede utilizarse en cadenas, comentarios y símbolos.

Tabla A-2 Caracteres imprimibles y caracteres especiales

Caracteres	Subgrupo	Elementos del juego de caracteres
Caracteres imprimbles	depende del código de caracteres utilizado. En código ASCII, p.ej., a partir del equivalente decimal 31, sin DEL y sin los siguientes caracteres complementarios:	todos los caracteres imprimbles
Caracteres de complementario	Signo de dólar	$
	Apóstrofo	’
Signos de control	$P o $p	Salto de página (formfeed page)
	$L o $l	Salto de línea (linefeed)
	$R o $r	Retorno de carro (carriage return)
	$T o $t	Tabulador
Representación complementaria en código hexadecimal	$hh	cualquier carácter, expresable en código hexadecimal (hh)

A.4 Caracteres de formateado, caracteres de separación y operadores

En reglas léxicas

En la table A-3 puede consultar el uso de caracteres aislados del juego de caracteres ASCII que sirven de caracteres de formateado y separación en al ámbito de las reglas léxicas (ver anexo B).

Tabla A-3 Caracteres de formateado y de separación en las reglas léxicas

Caracteres	Descripción
:	Carácter de separación entre horas, minutos y segundos Atributos
.	Carácter de separación para direccionamiento absoluto en representación de números reales e intervalos de tiempo
' '	Caracteres y cadena de caracteres
" "	Signo de inicio de símbolo según las reglas de la tabla de símbolos
- (Guión inferior)	Carácter de separación para valores numéricos dentro de literales; puede aparecer en IDENTIFICADORES
$	Símbolo de escape para indicar signos de control o caracteres complementarios
$> $<	Inserción de caracteres de formateado en string

Para literales

En la tabla A-4 puede consultar el uso de algunos caracteres y cadenas de caracteres para literales en el ámbito de las reglas léxicas (ver anexo B). Esta tabla es válida para nemotécnica SIMATIC y nemotécnica IEC.

Tabla A-4 Nemotécnica para literales, ordenada alfabéticamente

Prefijo	Identificador de	Regla léxica
2#	LITERAL ENTERO	Secuencia de cifras binarias
8#	LITERAL ENTERO	Secuencia de cifras octales
16#	LITERAL ENTERO	Secuencia de cifras hexadecimales
D#	Dato de temporizador	FECHA
DATE#	Dato de temporizador	FECHA
DATE_AND_TIME#	Dato de temporizador	FECHA Y HORA
DT#	Dato de temporizador	FECHA Y HORA
E	Carácter de separación para LITERAL NUMERO REAL	Exponente
e	Carácter de separación para LITERAL NUMERO REAL	Exponente
D	Carácter de separación para intervalo de tiempo (día)	Días (Regla: representación escalonada)
H	Carácter de separación para intervalo de tiempo (hora)	Horas (Regla: representación escalonada)
M	Carácter de separación para intervalo de tiempo (minutos)	Minutos (Regla: representación escalonada)
MS	Carácter de separación para intervalo de tiempo (milisegundos)	Milisegundos (Regla: representación escalonada)
S	Carácter de separación para intervalo de tiempo (segundos)	Segundos (Regla: representación escalonada)
T#	Dato de temporizador	INTERVALO
TIME#	Dato de temporizador	INTERVALO
TIME_OF_DAY#	Dato de temporizador	HORA DEL DIA
TOD#	Dato de temporizador	HORA DEL DIA

En reglas sintácticas

En la tabla A-5 puede consultar la utilización de caracteres concretos como caracteres de formateado y de separación en el ámbito de reglas sintácticas y en comentarios y atributos (v. apt. B.2 y B.3).

Tabla A-5 Caracteres de formateado y separación en las reglas sintácticas

Caracteres	Descripción	Regla sintáctica, comentario o atributo
:	Carácter de separación para indicación de tipo, en instrucción después de meta del salto	Declaración de variables, declaración de instancia, función, tabla de instrucciones, instrucción CASE
;	Fin de una declaración o instrucción	Declaración de constantes y variables, tabla de instrucciones, tabla de asignación de DB, bloque de constantes, bloque de metas de salto
'	Carácter de separación para listados	Declaración de variables, especificación de tipo de datos array, lista de inicialización de array, parámetros de FB, parámetros de FC, lista de valores
. .	Indicación de rango	Especificación de tipo de datos array, lista de valores
.	Carácter de separación para nombre de FB y DB, direccionamiento absoluto	Llamada a FB, variable estructurada
( )	Llamada a función y bloque de función Anidamiento en expresiones Lista de inicialización para arrays	Llamada a función, llamada a FB, expresión, lista de inicialización de array, multiplicación simple, expresión de potencia
[ ]	Declaración de arrays Variable estructurada, array parcial, indizado en variables globales y strings	Especficación de tipo de datos array, especificación, especificación de tipo de datos STRING
(* *)	Bloque de comentario	Ver anexo B
//	Línea de comentario	Ver anexo B
{ }	Bloque de atributos	Especificación de atributos
%	Introducción para identificadores directos	Para programar conforme a IEC se puede utilizar %M4.0 en lugar de M4.0.

Operadores

En la tabla A-6 se muestran todos los operadores de SCL, palabras clave (p.ej. AND) y los operadores habituales formados por caracteres individuales. Esta tabla es válida para nemotécnica SIMATIC y nemotécnica IEC.

Tabla A-6 Operadores SCL

Operador	Descripción	Regla sintáctica
:=	Operador de asignación	Asignación de valor, tabla de asignación de DB, bloque de constantes, asignación de salida y entrada/salida, asignación de entrada
+, -	Operadores aritméticos: operadores unitarios, signo	Expresión, expresión simple, expresión de potencia
+, -, +, / MOD; DIV	Operadores aritméticos básicos	Operador aritmético básico, expresión
**	Operadores aritméticos: operador de potencia	Expresión
NOT	Operadores lógicos: negación	Expresión
AND, &, OR; XOR,	Operadores lógicos básicos	Operador lógico básico, expresión
<,>, <=,>=,=,<>	Operador de comparación	Operador de comparación

A.5 Palabras clave e identificadores predefinidos

Palabras clave e identificadores predefinidos

En la tabla A-7 encontrará ordenadas alfabéticamente las palabras clave de SCL y los identificadores predefinidos. En el anexo C se acompaña una descripción y la regla sintáctica con la que se utilizan en los terminales. En general, las palabras clave no dependen de la nemotécnica (IEC y SIMATIC).

Tabla A-7 Palabras clave de SCL e identificadores predefinidos, ordenados alfabéticamente

Palabras clave	Descripción	Regla sintáctica
AND	Operador lógico	Operador lógico básico
ANY	Identificación de tipo de datos ANY	Especificación de tipo de datos de parámetro
ARRAY	Inicio de especificación de un array, después sigue la lista de índice encerrada entre ”[” y ”]”	Especificación de tipo de datos array
BEGIN	Inicio de tabla de asignación en bloques lógicos o tabla de inicialización en bloque de datos	Bloque de organización, función, bloque de función, bloque de datos
BLOCK_DB	Identificación de tipo de datos BLOCK_DB	Especificación de tipo de datos de parámetro
BLOCK_FB	Identificación de tipo de datos BLOCK_FB	Especificación de tipo de datos de parámetro
BLOCK_FC	Identificación de tipo de datos BLOCK_FC	Especificación de tipo de datos de parámetro
BLOCK_SDB	Identificación de tipo de datos BLOCK_SDB	Especificación de tipo de datos de parámetro
BOOL	Tipo de datos simple para datos binarios	Tipo de datos de bit
BY	Inicio de salto	Instrucción FOR
BYTE	Tipo de datos simple	Tipo de datos de bit
CASE	Inicio de instrucción de control para selección	Instrucción CASE
CHAR	Tipo de datos simple	Tipo de caracteres
CONST	Inicio de definición de constantes	Bloque de constantes
CONTINUE	Instrucción de control para bucle FOR, WHILE y REPEAT	Instrucción CONTINUE
COUNTER	Tipo de datos para contador, sólo utilizable en bloque de parámetros	Especificación de tipo de datos de parámetro
DATA_BLOCK	Inicio del bloque de datos	Bloque de datos
DATE	Tipo de datos simple para fecha	Tipo de temporizador
DATE_AND_TIME	Tipo de datos compuesto para fecha y hora del día	Ver tabla C–4
DINT	Tipo de datos simple para número (entero) precisión doble	Tipo de datos numérico
DIV	Operador de división	Operador aritmético básico, multiplicación simple
DO	Inicio de tabla de asignación en instrucción FOR	Instrucción FOR, instrucción WHILE
DT	Tipo de datos simple para fecha y hora del día	Ver tabla C–4
DWORD	Tipo de datos simple Doble palabra	Tipo de datos de bit
ELSE	Inicio de la condición cuando no se ha cumplido ninguna condición	Instrucción IF
ELSIF	Inicio de la condición alternativa	Instrucción IF
EN	Marca OK para confirmación de bloque	

Tabla A-7 Palabras clave de SCL e identificadores predefinidos, ordenados alfabéticamente, continuación

Palabras clave	Descripción	Regla sintáctica
END	Marca de error del bloque	
END_CASE	Final de instrucción CASE	Instrucción CASE
END_CONST	Final en definición de constantes	Bloque de constantes
END_DATA_BLOCK	Final de bloque de datos	Bloque de datos
END_FOR	Final de instrucción FOR	Instrucción FOR
END_FUNCTION	Final de función	Función
END_FUNC-TION_BLOCK	Final de bloque de función	Bloque de función
END_IF	Final de instrucción IF	Instrucción IF
END_LABEL	Final en la declaración de un bloque de metas de salto	Bloque de metas de salto
END_TYPE	Final de UDT	Tipo de datos de usuario
END_ORGANIZA-TION_BLOCK	Final del bloque de organización	Bloque de organización
END_REPEAT	Final de la instrucción REPEAT	Instrucción REPEAT
END_STRUCT	Final de la especificación de una estructura	Especificación de tipo de datos de estructura
END_VAR	Final del bloque de declaración	Bloque de variables temporales, bloque de variables estáticas, bloque de parámetros
END_WHILE	Final de la instrucción WHILE	Instrucción WHILE
EXIT	Cancelación directa de la edición de bucle	EXIT
FALSE	Constante booleana predefinida: condición lógica no cumplida, valor igual a 0	
FOR	Inicio de instrucción de control para edición de bucle	Instrucción FOR
FUNCTION	Inicio de función	Función
FUNCTION_BLOCK	Inicio de bloque de función	Bloque de función
GOTO	Instrucción para ejecutar un salto hasta una meta de salto	Salto del programa
IF	Inicio de instrucción de control para selección	Instrucción IF
INT	Tipo de datos elemental para número (entero), precisión simple	Tipo de datos numérico
LABEL	Inicio de declaración de un bloque de metas de salto	Bloque de metas de salto
MOD	Operador aritmético para resto de división	Operador aritmético básico, multiplicación simple
NIL	Puntero cero	
NOT	Operador lógico, pertenece a los operadores uniarios	Expresión, operando
OF	Inicio de especificación de tipo de datos	Especificación de tipo de datos array, instrucción CASE
OK	Marca OK que expresa si se han ejecutado sin error las instrucciones de un bloque	
OR	Operador lógico	Operador lógico básico
ORGANIZA-TION_BLOCK	Inicio del bloque de organización	Blque de organización

Tabla A-7 Palabras clave de SCL e identificadores predefinidos, ordenados alfabéticamente, continuación

Palabras clave	Descripción	Regla sintáctica
POINTER	Tipo de datos puntero, sólo permitido en declaración de parámetros en bloque de parámetros, no se edita en SCL	V. cap. 10
REAL	Tipo de datos simple	Tipo de datos numérico
REPEAT	Inicio de instrucción de control para edición de bucle	Instrucción REPEAT
RETURN	Instrucción de control para retorno de subrutina	Instrucción RETURN
S5TIME	Tipo de datos simple para datos de temporizador, formato S5 especial	Tipo de temporizador
STRING	Tipo de datos para cadena de caracteres	Especificación de tipo de datos STRING
STRUCT	Inicio de la especificación de una estructura, seguido de lista de componentes	Especificación de tipo de datos de estructura
THEN	Inicio de acciones consecutivas cuando se cumple una condición	Instrucción IF
TIME	Tipo de datos simple para datos de temporizador	Tipo de temporizador
TIMER	Tipo de datos para temporizador, sólo utilizable en bloque de parámetros	Especificación de tipo de datos de parámetro
TIME_OF_DAY	Tipo de datos simple para hora del día	Tipo de temporizador
TO	Inicio del valor final	Instrucción FOR
TOD	Tipo de datos simple para hora del día	Tipo de temporizador
TRUE	Constante booleana predefinida; condición lógica cumplida, valor distinto de 0	
TYPE	Inicio del UDT	Tipo de datos de usuario
VAR	Inicio de una tabla de declaración	Bloque de variables estáticas
VAR_TEMP	Inicio de una tabla de declaración	Bloque de variables temporales
UNTIL	Inicio de la condición de interrupción para instrucción REPEAT	Instrucción REPEAT
VAR_INPUT	Inicio de una tabla de declaración	Bloque de parámetros
VAR_IN_OUT	Inicio de una tabla de declaración	Bloque de parámetros
VAR_OUTPUT	Inicio de una tabla de declaración	Bloque de parámetros
WHILE	Inicio de la instrucción de control para edición de bucle	Instrucción WHILE
WORD	Tipo de datos simple (PALABRA)	Tipo de datos de bit
VOID	Sin valor de respuesta en una llamada a función	V. cap. 8
XOR	Operador lógico	Operador lógico

A.6 Identificadores de operando y palabras clave de bloques

Datos globales del sistema

En la tabla A-8 puede consultar la nemotécnica SIMATIC de los identificadores de operandos en SCL, junto con una descripción, ordenada alfabéticamente:

- Indicación del identificador de operando:
  prefijo de memoria (A, E, M, PA, PE) o bloque de datos (D)
- Indicación del tamaño del elemento de datos:
  prefijo de tamaño (opcional o B, D, W, X).

La nemotécnica constituye una combinación entre el indicador de operando (prefijo de memoria o D para bloque de datos) y el prefijo de tamaño. Ambas son reglas léxicas. La tabla está clasificada según la nemotécnica SIMATIC; adicionalmente se indica la correspondiente nemotécnica IEC.

Tabla A-8 Identificadores de operandos de datos globales del sistema

Nemotécnica SIMATIC	Nemotécnica IEC	Prefijo de memoria o bloque de datos	Prefijo de tamaño
A	Q	Salida (mediante imagen del proceso)	Bit
AB	QB	Salida (mediante imagen del proceso)	Byte
AD	QD	Salida (mediante imagen del proceso)	Doble palabra
AN	QW	Salida (mediante imagen del proceso)	Palabra
AX	QX	Salida (mediante imagen del proceso)	Bit
D	D	Bloque de datos	Bit
DB	DB	Bloque de datos	Byte
DD	DD	Bloque de datos	Doble palabra
DW	DW	Bloque de datos	Palabra
DX	DX	Bloque de datos	Bit
E	I	Entrada (mediante imagen del proceso)	Bit
EB	IB	Entrada (mediante imagen del proceso)	Byte
ED	ID	Entrada (mediante imagen del proceso)	Doble palabra
EW	IW	Entrada (mediante imagen del proceso)	Palabra
EX	IX	Entrada (mediante imagen del proceso)	Bit
M	M	Marca	Bit
MB	MB	Marca	Byte
MD	MD	Marca	Doble palabra
MW	MW	Marca	Palabra
MX	MX	Marca	Bit
PAB	PQB	Salida (directamente a periferia)	Byte
PAD	PQD	Salida (directamente a periferia)	Doble palabra
PAW	PQW	Salida (directamente a periferia)	Palabra
PEB	PIB	Entrada (directamente de periferia)	Byte
PED	PID	Entrada (directamente de periferia)	Doble palabra
PEW	PIW	Entrada (directamente de periferia)	Palabra

Palabras clave de bloques

Se utilizan para el direccionamiento absoluto de bloques. La tabla está organizada según la nemotécnica SIMATIC; adicionalmente se indica la correspondiente nemotécnica IEC.

Tabla A-9 Palabras clave de bloques, contadores y temporizadores

Nemotécnica SIMATIC	Nemotécnica IEC	Prefijo de memoria o bloque de datos
DB	DB	Bloque de datos (Data Block)
FB	FB	Bloque de función (Function Block)
FC	FC	Función (Function)
OB	OB	Bloque de organización (Organization Block)
SDB	SDB	Bloque de datos del sistema (System Data Block)
SFC	SFC	Función del sistema (System Function)
SFB	SFB	Bloque de función del sistema (System Function Block)
T	T	Temporizador (Timer)
UDT	UDT	Tipo de datos globales o de usuario (Userdefined Data Type)
Z	C	Contador (Counter)

A.7 Resumen de nonterminales

Definición

Un nonterminal es un elemento compuesto que se describe mediante otras reglas. El elemento nonterminal se representa mediante un cajetín. El nombre del cajetín corresponde al nombre de la regla que se aplica a continuación.

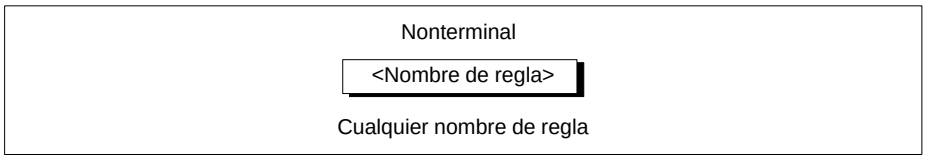


Figura A-6 Nonterminal

El elemento aparece en las reglas léxicas y en las reglas sintácticas.

A.8 Resumen de token

Definición

Un token es un elemento compuesto que se utiliza en las reglas sintácticas como elemento fundamental y que se explica en las reglas léxicas. El token se representa mediante un rectángulo. El NOMBRE (en mayúsculas) corresponde al nombre de la regla léxica que se aplica a continuación (sin rectángulo)

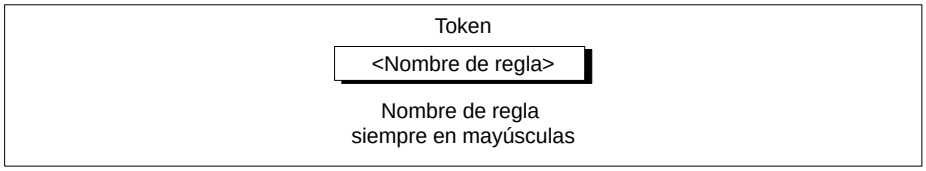


Figura A-7 Token

Resumen

Los token definidos representan identificadores que se escifican como resultados de reglas léxicas. Estos token describen:

- Identificadores
- Asignaciones de nombre en SCL
- Constantes predefinidas y marcas OK

A.9 Identificadores

Identificadores en SCL

Por identificadores nos referimos a objetos del lenguaje SCL. La tabla A-10 le informa sobre las clases de identificadores:

Tabla A-10 Conjunto de las clases de identificadores en SCL

Tipo de identificador	Observaciones, ejemplos
Palabras clave	P.ej., Instrucciones de control `BEGIN,DO,WHILE`
Nombres predefinidos	Nombres de • Tipos de datos estándar (p.ej.: BOOL, BYTE, INT) • Funciones estándar predefinidas, p. ej. `ABS` • Constantes estándar `TRUE` y `FALSE`
Identificadores de operandos en identificadores absolutos	sistema y bloques de datos: p.ej. `E1.2, MW10, FC20, T5, DB30, DB10.D4.5`
Nombres de libre definición según la regla de IDENTIFICADOR	Nombres de • Variables declaradas • Componentes de estructura • Parámetros • Constantes declaradas • Metas de salto
Símbolos de la tabla de símbolos	cumplen la regla léxica `IDENTIFICADOR` o la regla léxica de símbolo; es decir, encerrado entre comillas, p. ej. ”xyz”

Mayúsculas y minúsculas

En las palabras clave es indiferente que se escriba con mayúsculas o minúsculas. A partir de la versión 4.0 de SCL para S7 tampoco se distingue entre mayúsculas y minúsculas en los nombres predefinidos ni en los nombres de libre elección (p.ej., para variables, y en los símbolos de la tabla de símbolos). La tabla A-11 le ofrece un resumen.

Tabla A-11 Importancia del uso de mayúsculas o minúsculas

Clase de identificador	Diferenciación
Palabras clave	No
Nombres predefinidos en tipos de datos estándar	No
Nombres en funciones estándar predefinidas	Sí
Nombres predefinidos en constantes estándar	No
Identificadores de operando en identificadores absolutos	No
Nombres de libre elección	Sí
Símbolos de la tabla de símbolos	Sí

Por lo tanto, los nombres de las funciones estándar (p.ej., BYTE_TO_WORD y ABS) también pueden escribirse en minúsculas. Lo mismo se aplica para los parámetros de funciones de temporización y conteo, p.ej., SV, se o ZV.

A.10 Asignación de nombres en SCL

Asignación de nombres de libre elección

En general, para asignar nombres existen dos posibilidades:

- Puede asignar nombres dentro de SCL. Dichos nombres deben cumplir la regla IDENTIFICADOR (v. fig. A-8). La regla IDENTIFICADOR puede utilizarla para cualquier nombre dentro de SCL.
- También puede introducir nombres mediante STEP 7 con ayuda de la tabla de símbolos. En este caso, la regla es también IDENTIFICADOR, existiendo además la posibilidad del símbolo. Utilizando apóstrofos, el símbolo puede escribirse con caracteres imprimibles (p.ej., con blancos).

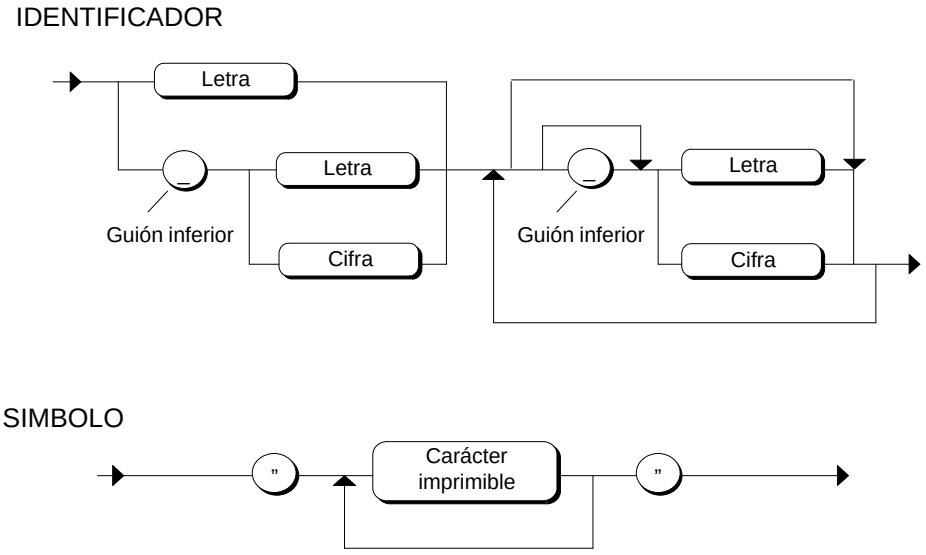


Figura A-8 *Reglas léxicas* IDENTIFICADOR *y* SIMBOLO

Reglas para la asignación de nombres

Tenga en cuenta lo siguiente:

- Para la asignación de nombres lo mejor es seleccionar nombres claros y expresivos que ayuden a la inteligibilidad del programa.
- Asegúrese de que el mismo nombre no ha sido asignado previamente por el sistema; p.ej., mediante identificadores para tipos de datos o funciones estándar.
- Rango de validez: en los nombres que tienen validez global, el rango de validez se extiende al programa completo. Los nombres válidos localmente sólo se aplican dentro de un bloque. Así es posible utilizar un mismo nombre en diferentes bloques. La tabla A-12 le informa sobre las posibilidades de que dispone.

Limitaciones de los nombres

En la asignación de nombres debe tener en cuenta algunas limitaciones.

Los nombres deben ser unívocos dentro de su rango de validez; es decir, los nombres que ya se han adjudicado dentro de un bloque no pueden utilizarse otra vez dentro del mismo bloque. Asimismo, tampoco pueden utilizarse los siguientes nombres ya asignados por el sistema:

- Nombres de palabras clave: p.ej., CONST, END_CONST, BEGIN
- Nombres de operadores: p.ej., AND, XOR
- Nombres de identificadores predefinidos: p.ej., nombres de tipos de datos como BOOL, STRING, INT
- Nombres de las constantes predefinidas TRUE y FALSE
- Nombres de funciones estándar: p.ej., ABS, ACOS, ASIN, COS, IN
- Nombres de identificadores absolutos o de operandos para datos globales del sistema: p.ej., EB, EW, ED, AB, AW, AD, MB, MD

Utilización de IDENTIFICADORES

La tabla A-12 le muestra los casos en que puede utilizar nombres que cumplen la regla de IDENTIFICADOR.

Tabla A-12 Utilización de IDENTIFICADORES

IDENTIFICADOR	Descripción	Regla
Nombre de bloque	Nombre simbólico para bloque	IDENTIFICACIÓN DE BLOQUE, llamada a función
Nombre de temporizador y contador	Nombre simbólico para temporizador y contador	IDENTIFICACIÓN DE TEMPORIZADOR, IDENTIFICACIÓN DE CONTADOR
Nombre de atributo	Nombre para un atributo	Asignación de atributo
Nombre de constante	Declaración de constantes simbólicas, utilización	Bloque de constantes, constante
Meta de salto	Declaración de meta de salto, utilización de meta de salto	Tabla de instrucciones de meta de salto, instrucción GOTO
Nombre de variable	Declaración de variables temporales o estáticas	Declaración de variables, variable simple, variable estructurada
Nombre de instancia local	Instancia de declaración local	Declaración de instancia, nombre de llamada a FB

IDENTIFICACION DE BLOQUE

En la regla `IDENTIFICACION DE BLOQUE` puede utilizar tanto `IDENTIFICADORES` como `símbolos`:

IDENTIFICACION DE BLOQUE

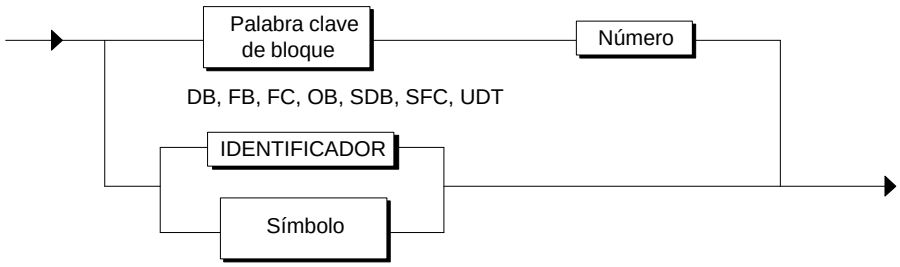


Figura A-9 *Regla léxica IDENTIFICACION DE BLOQUE*

Las reglas `IDENTIFICACION DE TEMPORIZADOR` e `IDENTIFICACION DE CONTADOR` se aplican análogamente a `IDENTIFICACION DE BLOQUE`.

A.11 Constantes predefinidas y marcas

Constantes predefinidas y marcas

Las dos tablas siguientes se aplican a la nemotécnica SIMATIC y a la nemotécnica IEC.

Tabla A-13 Constantes predefinidas

Nemotécnica	Descripción
`FALSE`	Constantes booleanas predefinidas (constantes estándar) de valor 0. Su significado lógico es que una condición no se ha cumplido.
`TRUE`	Constante booleana predefinida (constante estándar) con valor 1. Su significado lógico es que una condición se ha cumplido.

Tabla A-14 Marcas

Nemotécnica	Descripción
`EN`	Marca para habilitación de bloque
`ENO`	Marca de error de bloque
`OK`	Marca que se pone a FALSE si una instrucción se ha ejecutado con errores

Reglas léxicas

B

Indice del capítulo

Reglas léxicas

Las reglas léxicas describen la estructura de los elementos (token) que se procesan en el análisis léxico del compilador. No se dispone de libertad de formato para la escritura y deben respetarse estrictamente las reglas. Esto significa en concreto que:

- No está permitido insertar caracteres de formateado.
- No pueden insertarse bloques de comentarios ni líneas de comentario.
- No pueden insertarse atributos para identificadores.

Clasificación

Las reglas léxicas se clasifican en los siguientes grupos:

- Identificaciones
- Literales
- Direccionamiento absoluto

B.1 Identificaciones

Tabla B-1 Identificaciones

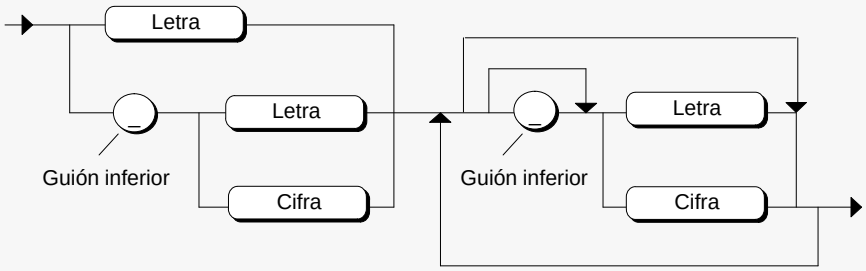
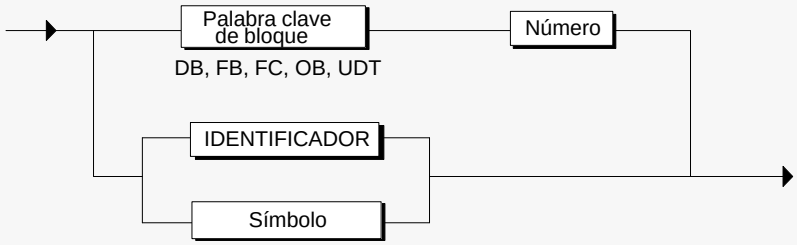
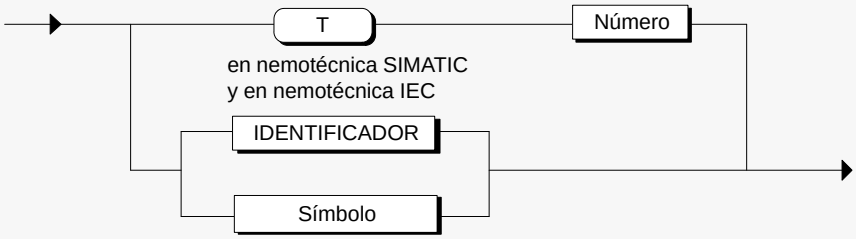
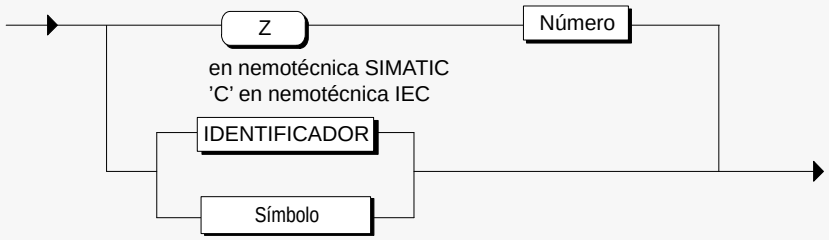
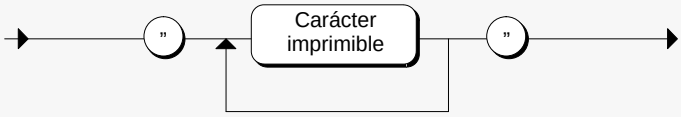
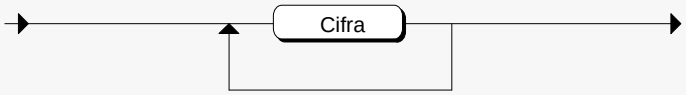
Regla	Diagrama sintáctico
IDENTIFICADOR	
Identificación de bloque	La regla se aplica también para los siguientes nombres de reglas: IDENTIFICACION DE DB IDENTIFICACION DE FB IDENTIFICACION DE FC IDENTIFICACION DE OB IDENTIFICACION DE UDT 
Identificación de temporizador	

Tabla B-1 Identificaciones, continuación

Regla	Diagrama sintáctico
IDENTIFICACION DE CONTADOR	
Palabra clave de bloque	
Símbolo	
Número	

B.1.1 Literales

Tabla B-2 Literales

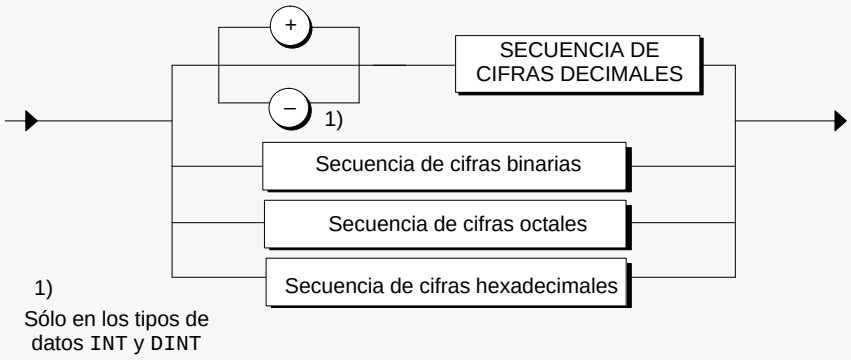
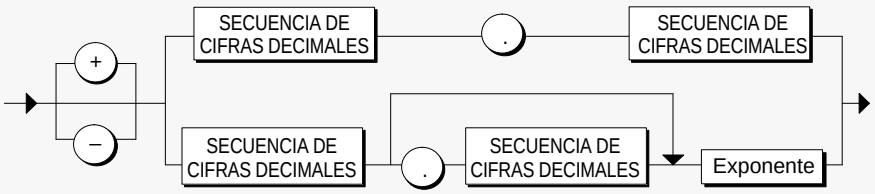
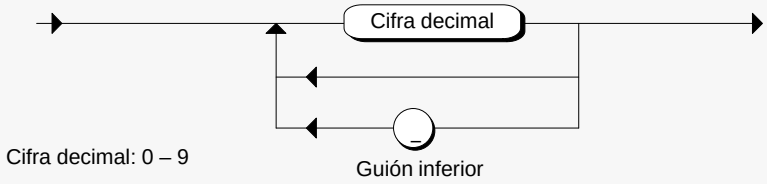
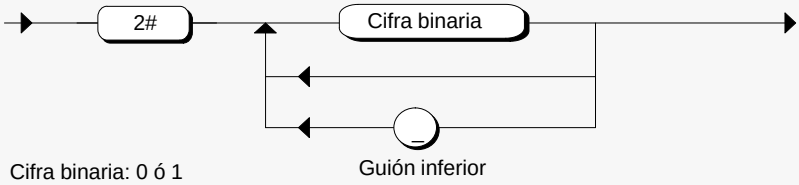
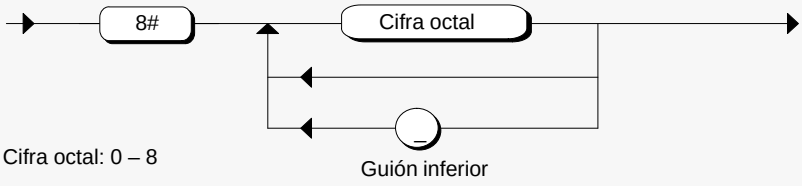
Regla	Diagrama sintáctico
LITERAL ENTERO	
Literal número real	
SECUENCIA DE CIFRAS DECIMALES	
Secuencia de cifras binarias	
Secuencia de cifras octales	

Tabla B-2 Literales, continuación

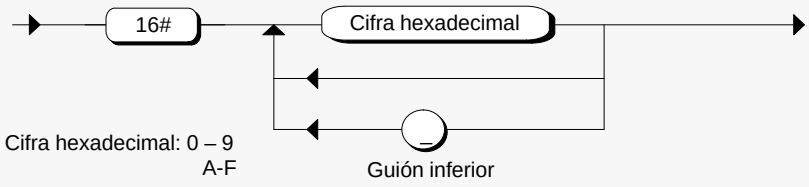
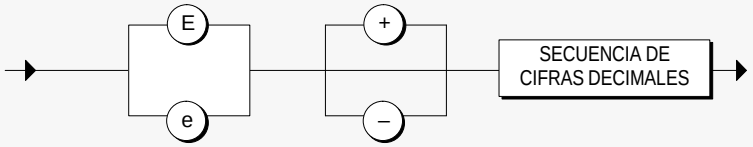
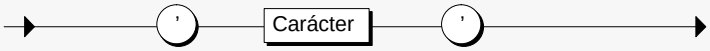
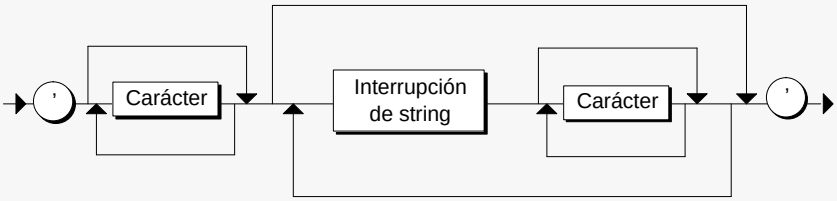
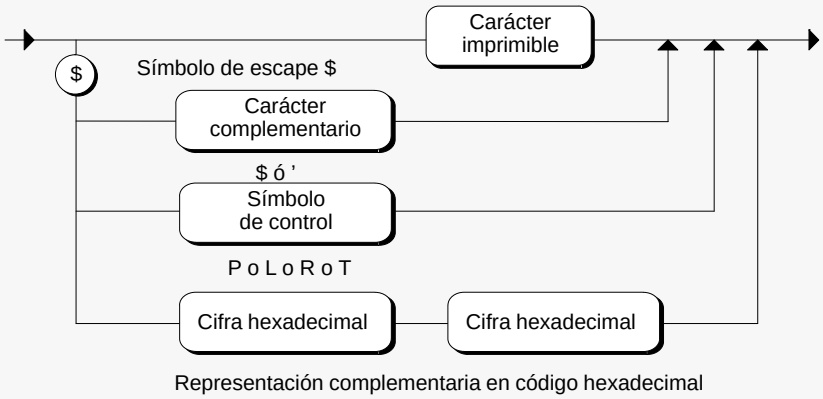
Regla	Diagrama sintáctico
Secuencia de cifras hexadecimales	
Exponente	
LITERAL DE CARACTER	
Literal STRING	
Carácter	

Tabla B-2 Literales, continuación

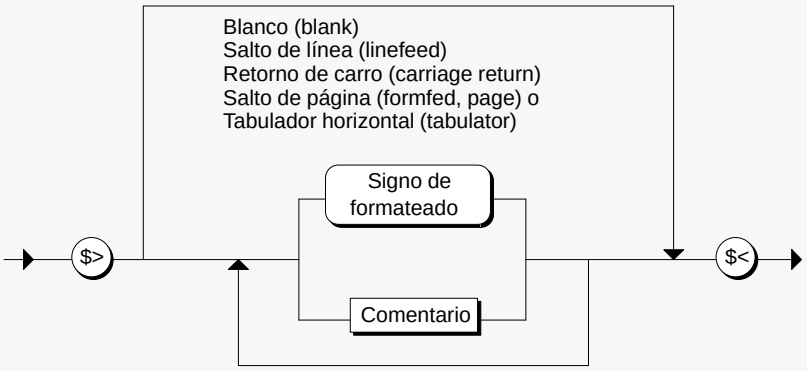
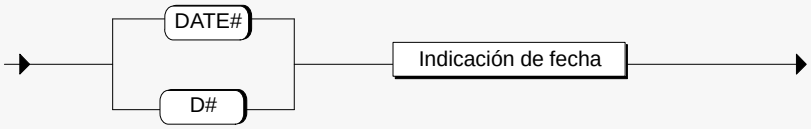
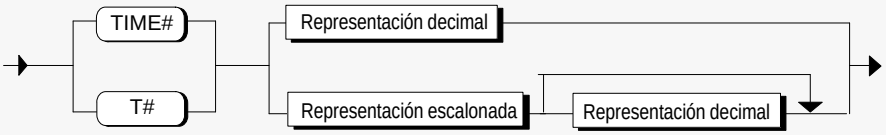
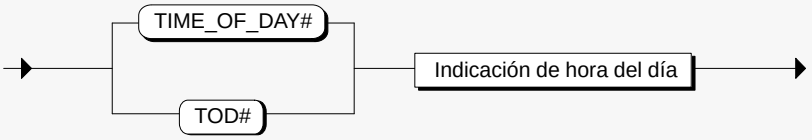
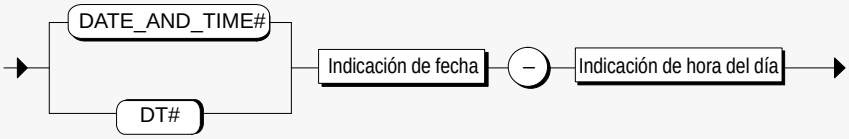
Regla	Diagrama sintáctico
Interrupción de string	
Fecha	
Intervalo	 Toda unidad de tiempo (p.ej. horas, minutos) sólo puede indicarse una vez. Debe respetarse la siguiente secuencia: días, horas, minutos, segundos, milisegundos.
Hora del día	
Fecha y hora	

Tabla B-2 Literales, continuación

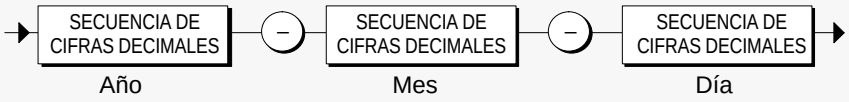
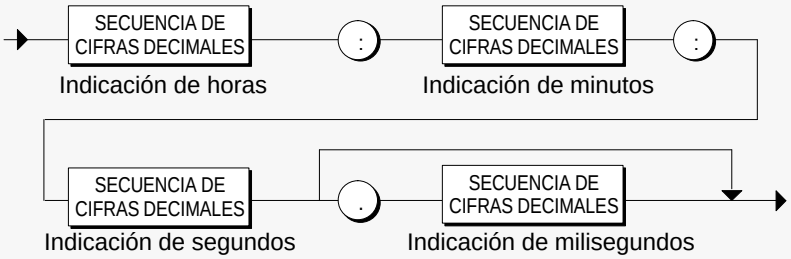
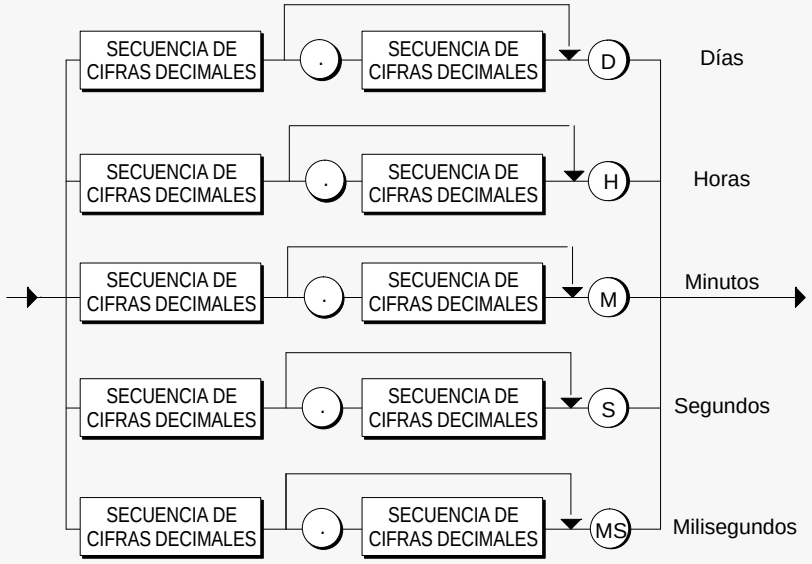
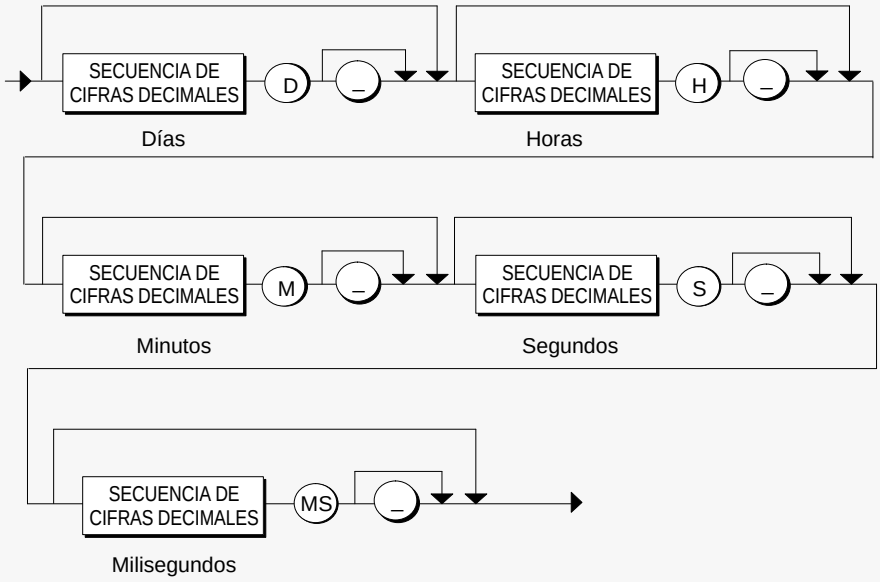
Regla	Diagrama sintáctico
Indicación de fecha	
Indicación de hora del día	

Tabla B-2 Literales, continuación

Regla	Diagrama sintáctico
Representación decimal	 Sólo es posible acceder a la representación decimal con unidades de tiempo sin definir.
Representación escalonada	 Es necesario al menos un dato

B.1.2 Direccionamiento absoluto

Tabla B-3 Direccionamiento absoluto

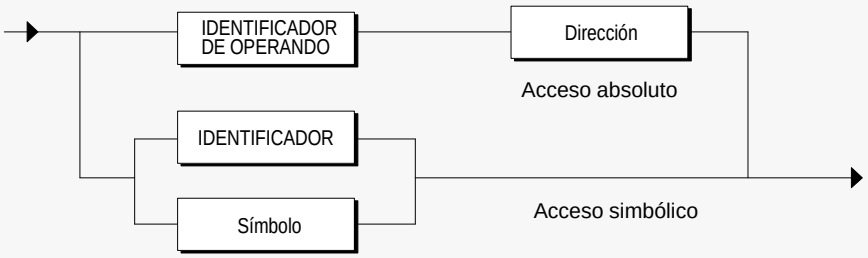
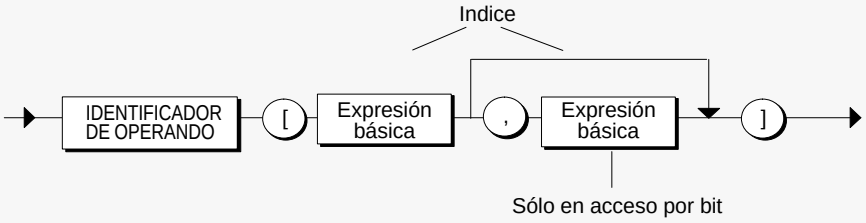
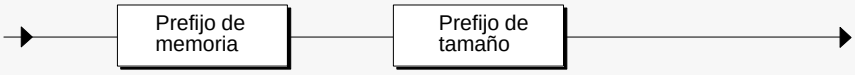
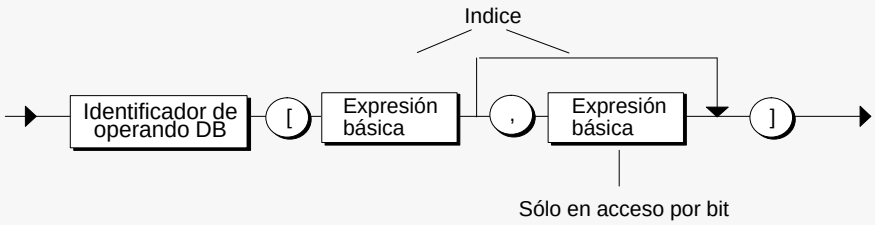
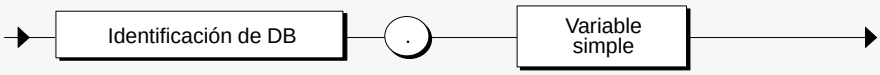
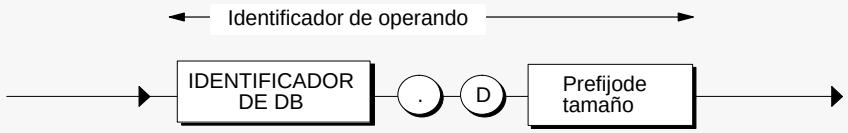
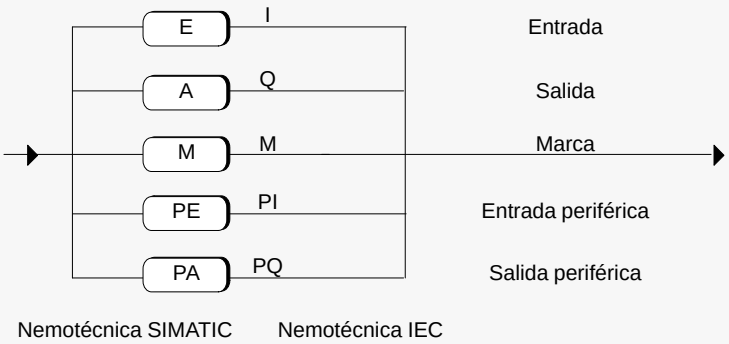
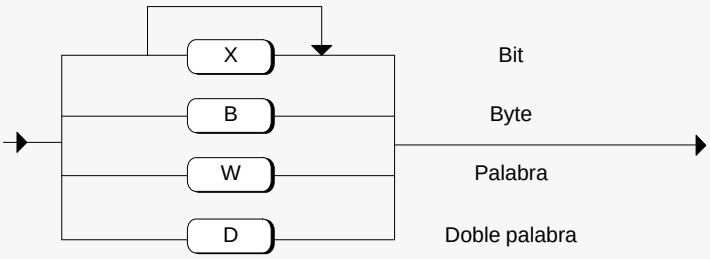
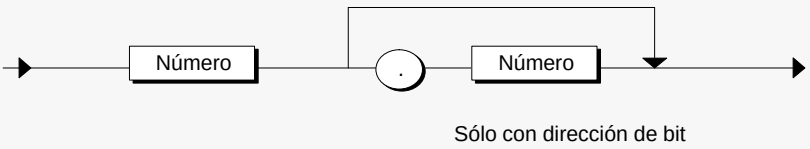
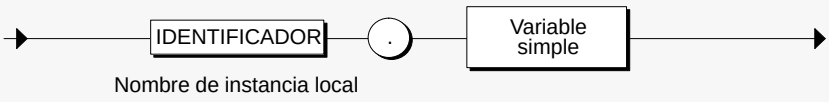
Regla	Diagrama sintáctico
Acceso simple a memoria	
Acceso indizado a memoria	
Identificador de operando para memoria	
Acceso absoluto a DB	
Acceso indizado a DB	
Acceso estructurado a DB	

Tabla B-3 Direccionamiento absoluto, continuación

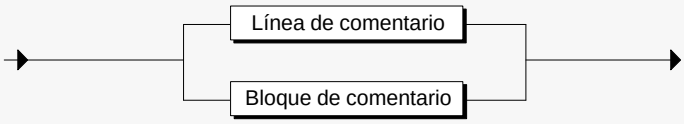
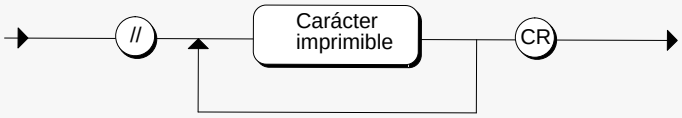
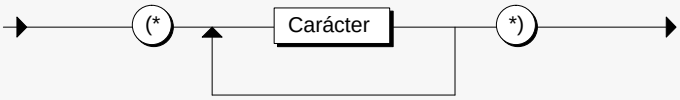
Regla	Diagrama sintáctico
Identificador de operando DB	
Prefijo de memoria	
Prefijo de tamaño para memoria y DB	
Dirección de memoria y DB	
Acceso a instancia local	

B.2 Comentarios

Tenga en cuenta Al insertar comentarios debe tener en cuenta los dos puntos siguientes:

- No está permitido el anidamiento de comentarios
- Es posible insertarlos en cualquier punto de las reglas sintácticas, pero no en las reglas léxicas.

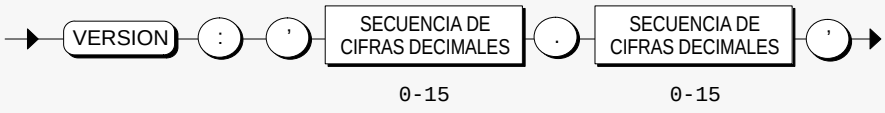
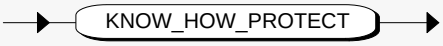
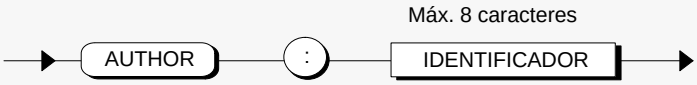
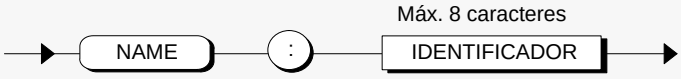
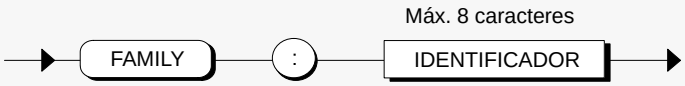
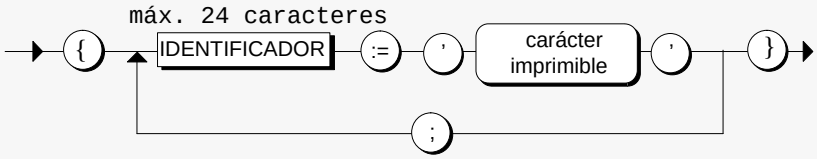
Tabla B-4 Comentarios

Regla	Diagrama sintáctico
Comentario	
Línea de comentario	
Bloque de comentario	

B.3 Atributos de bloque

Tenga en cuenta Los atributos de bloque pueden estar situados después de la IDENTIFICACIÓN DE BLOQUE y delante de la declaración del primer bloque de variables o de parámetros, siguiendo la sintaxis que se muestra a continuación.

Tabla B-5 Atributos

Regla	Diagrama sintáctico
Título	
Versión	
Protección de bloque	
Autor	
Nombre	
Familia de bloque	
Atributos del sistema para bloques	

Reglas sintácticas

C

Definición de reglas sintácticas

En las reglas sintácticas se describe la estructura de SCL basándose en las reglas léxicas. Dentro de estas reglas dispone de libertad de formato para crear su programa en SCL.

Indice del capítulo

Consideraciones formales

Toda regla tiene un nombre antepuesto. Cuando la regla se utiliza en una regla superior, su nombre aparece encerrado en un rectángulo.

Si el nombre del rectángulo está escrito con mayúsculas, se trata de un token, descrito en las reglas léxicas.

Sobre los nombres de reglas encerrados en círculos o en rectángulos con esquinas redondeadas, puede encontrar información en el anexo A.

Tenga en cuenta

La característica de libertad de formato significa lo siguiente:

- es posible insertar caracteres de formateado en cualquier punto;
- pueden insertarse bloques de comentario y líneas de comentario (v. apt. 7.6).

C.1 Clasificación de fuentes SCL

Tabla C-1 Sintaxis de las fuentes SCL

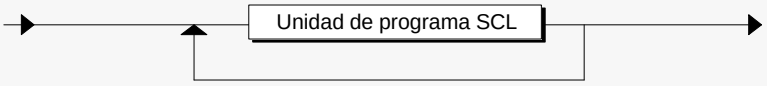
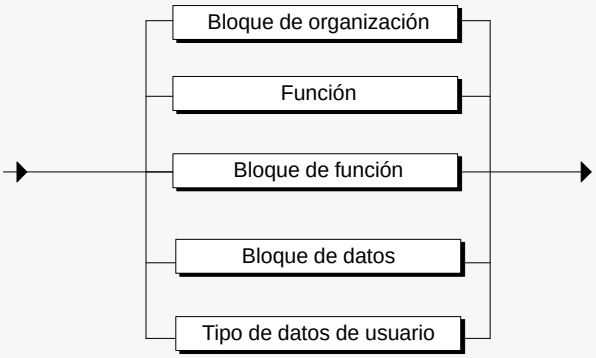
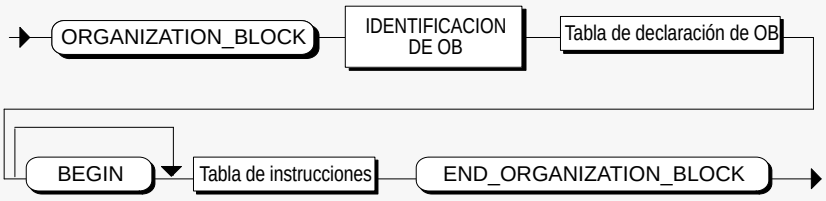
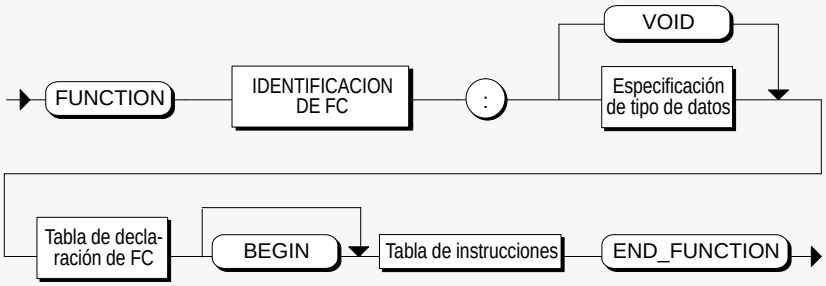
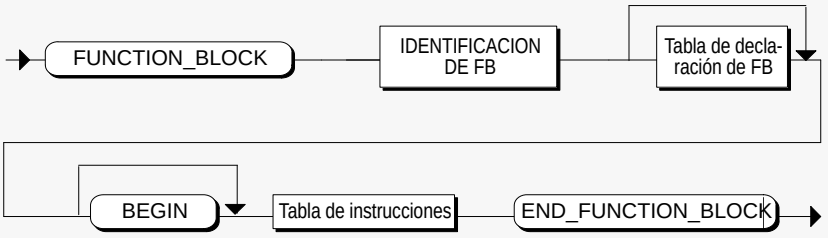
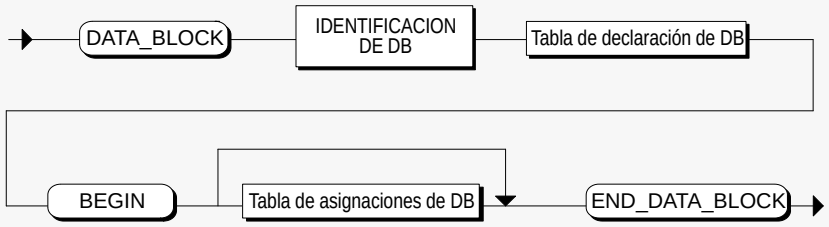
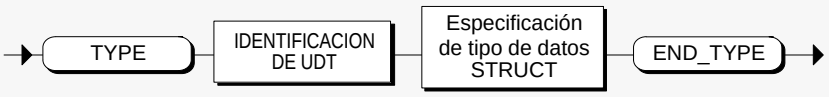
Regla	Diagrama sintáctico
Programa en SCL	
Unidad de programa SCL	
Bloque de organización	
Función Tenga en cuenta que en las funciones que no tienen VOID en la tabla de instrucciones, el valor de respuesta debe asignarse al nombre de la función.	
Bloque de función	

Tabla C-1 Sintaxis de las fuentes SCL, continuación

Regla	Diagrama sintáctico
Bloque de datos	
Tipo de datos de usuario	

C.2 Organización de las tablas de declaración

Tabla C-2 Sintaxis de la tabla de declaración

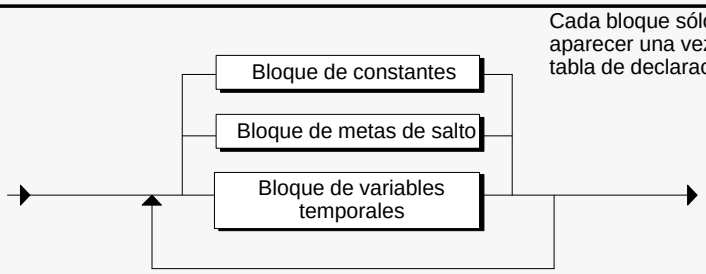
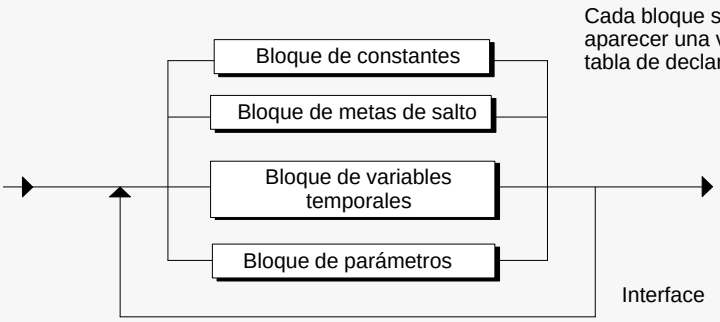
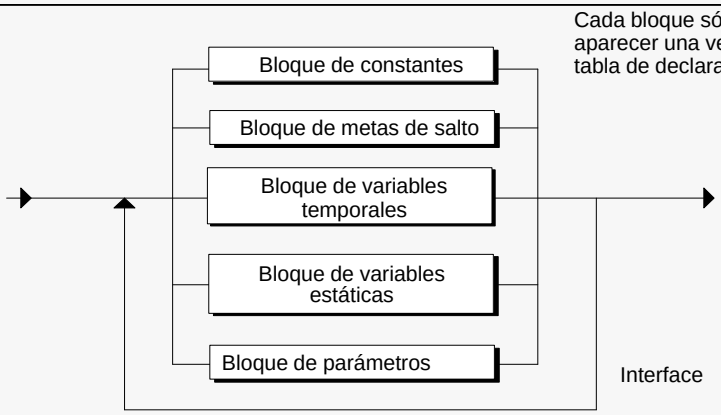
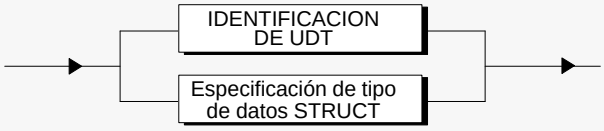
Regla	Diagrama sintáctico
Tabla de declaración de OB	
Tabla de declaración de FC	
Tabla de declaración de FB	
Tabla de declaración de FB	

Tabla C-3 Sintaxis de los bloques de declaración

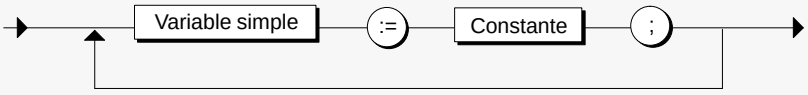
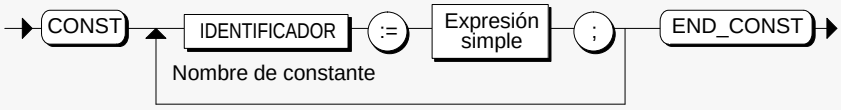
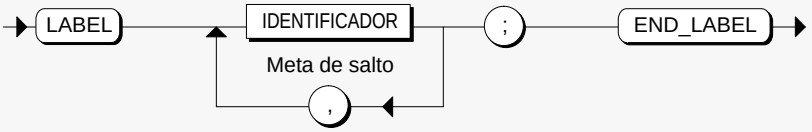
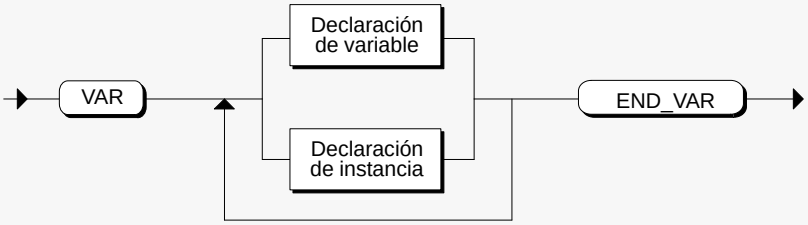
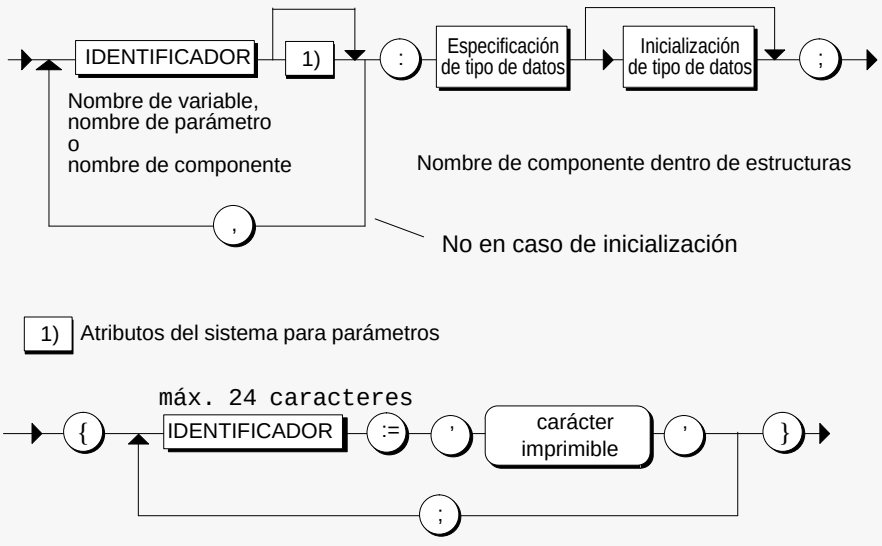
Regla	Diagrama sintáctico
Tabla de asignación de DB	
Bloque de constantes	
Bloque de metas de salto	
Bloque de variables estáticas	
Declaración de variable	

Tabla C-3 Sintaxis de los bloques de declaración, continuación

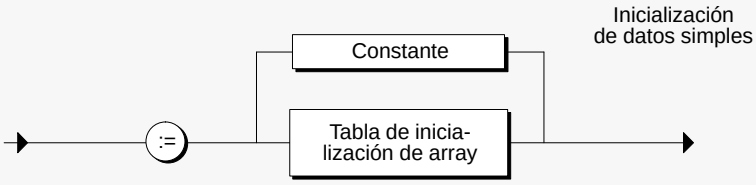
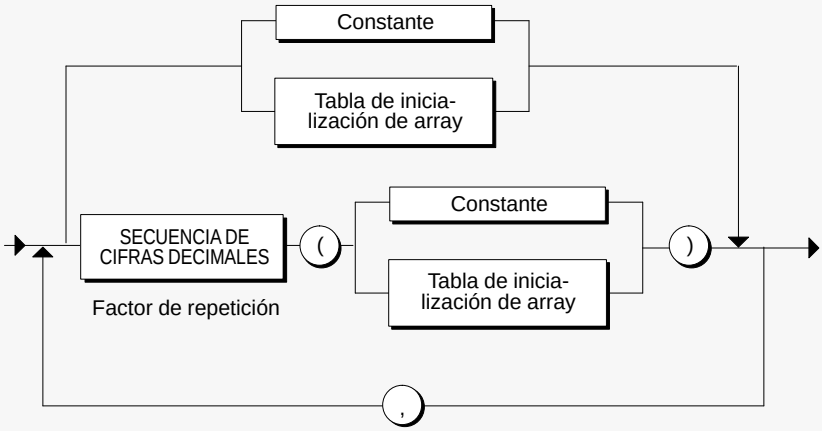
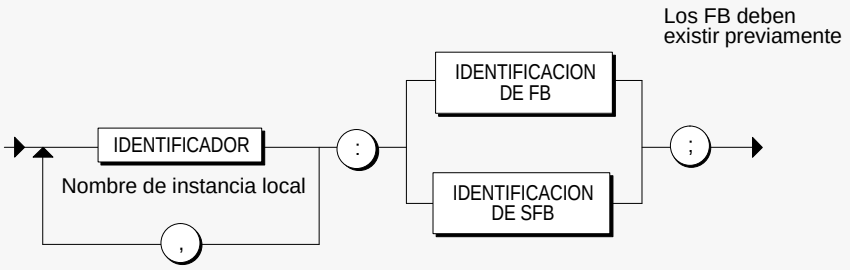
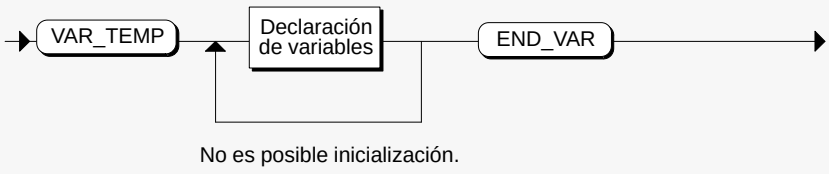
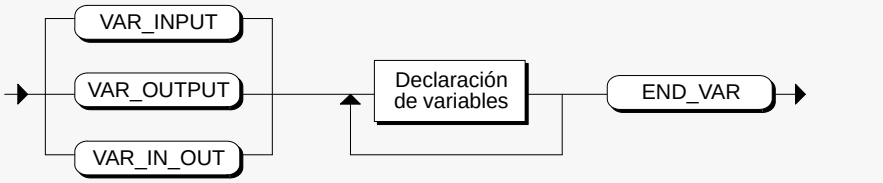
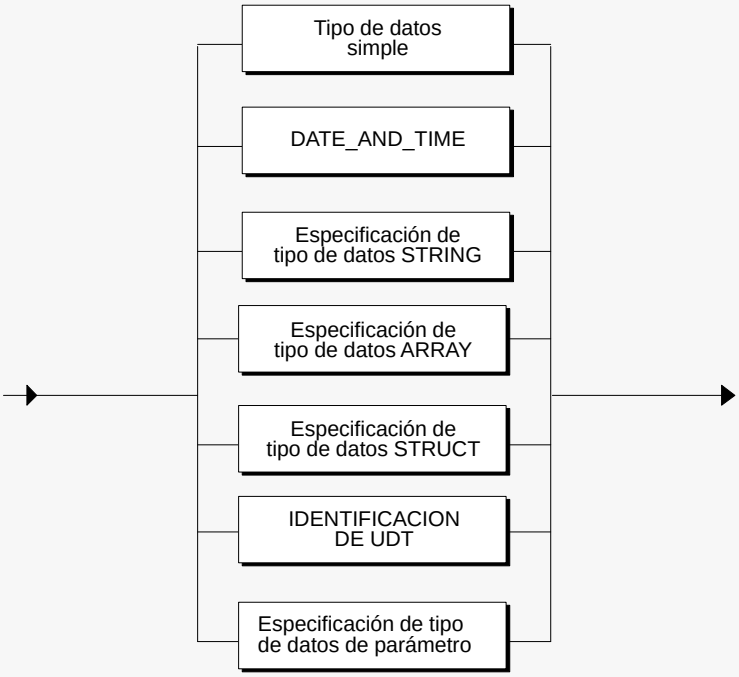
Regla	Diagrama sintáctico
Inicialización de tipo de dato	
Tabla de inicialización de array	
Declaración de instancia	
Bloque de variables temporales	

Tabla C-3 Sintaxis de los bloques de declaración, continuación

Regla	Diagrama sintáctico
Bloque de paráme-tros	 La inicialización sólo es posible para VAR_INPUT y VAR_OUTPUT.
Especificación de tipo de datos	

C.3 Tipos de datos en SCL

Tabla C-4 Sintaxis de los tipos de datos en la tabla de declaración

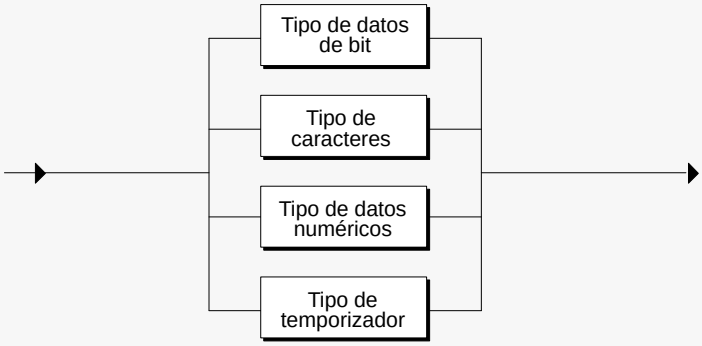
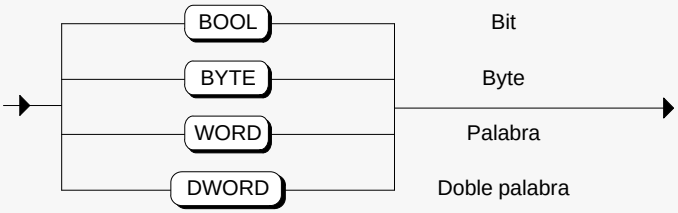
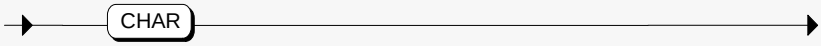
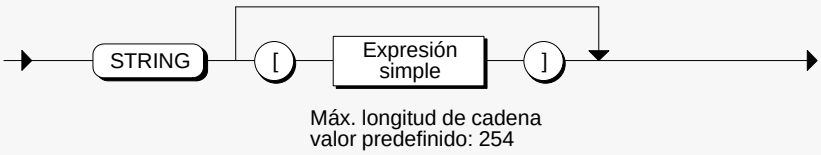
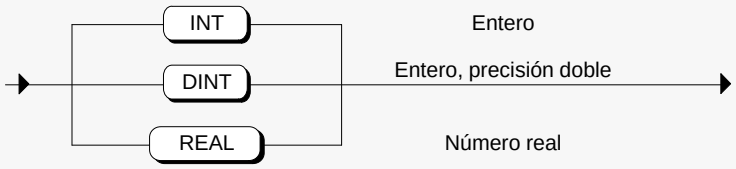
Regla	Diagrama sintáctico
Tipo de datos simple	
Tipo de datos de bits	
Tipo de caracteres	
Especificación de tipo de datos STRING	
Tipo de datos numérico	

Tabla C-4 Sintaxis de los tipos de datos en la tabla de declaración, continuación

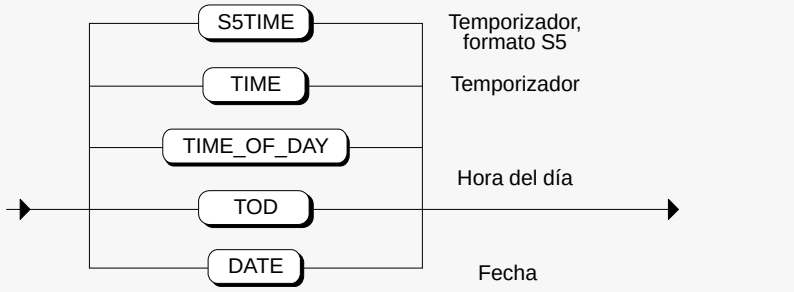
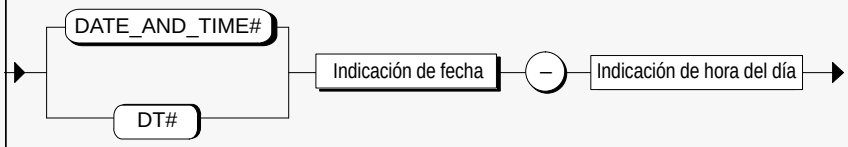
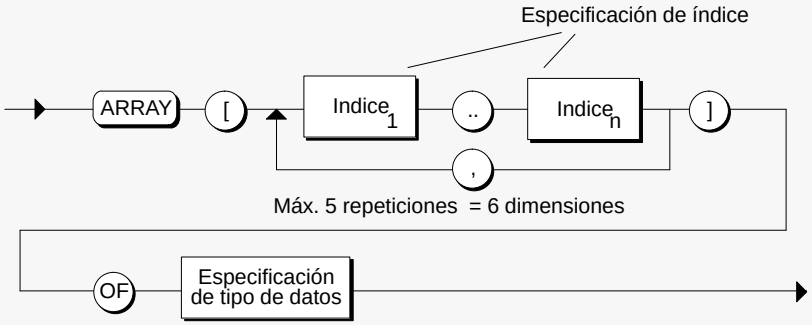
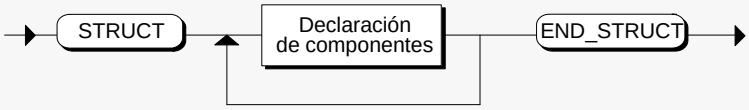
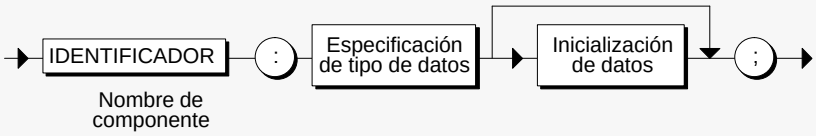
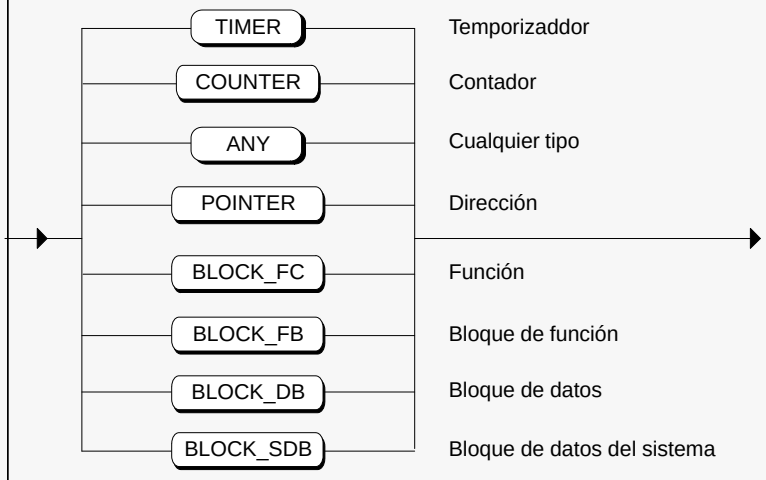
Regla	Diagrama sintáctico
Tipo de temporizador	 También v. apt. B.1.1
DATE_AND_TIME	
Especificación de tipo de datos ARRAY	
Especificación de tipo de datos STRUCT No olvide terminar la palabra clave END_STRUCT con punto y coma.	

Tabla C-4 Sintaxis de los tipos de datos en la tabla de declaración, continuación

Regla	Diagrama sintáctico
Declaración de componentes	
Especificación de tipo de parámetro	

C.4 Tabla de instrucciones

Tabla C-5 Sintaxis de la tabla de instrucciones

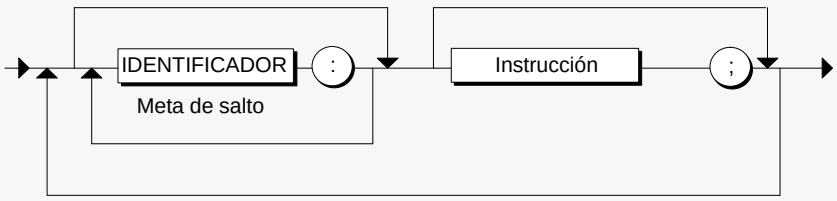
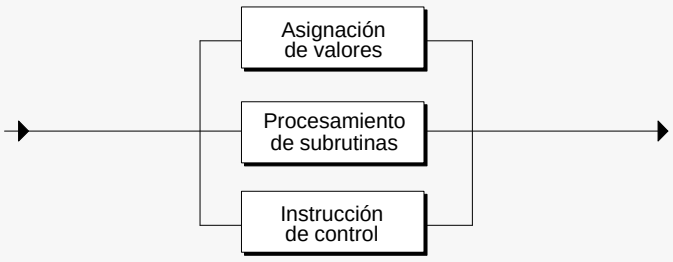
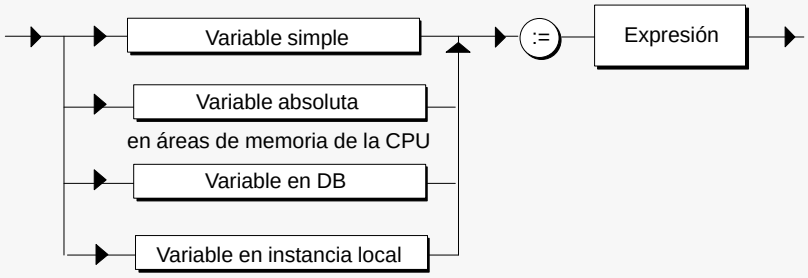
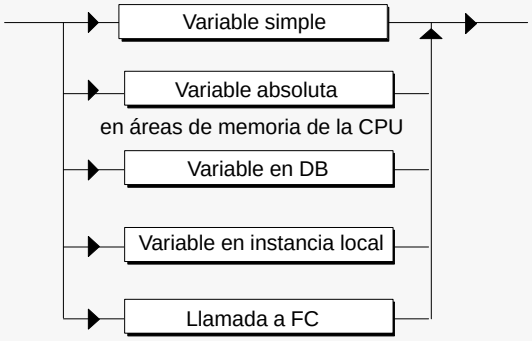
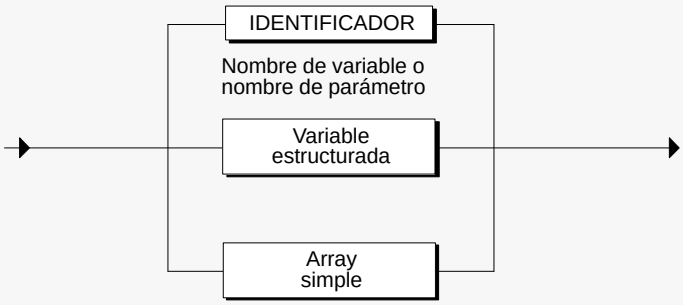
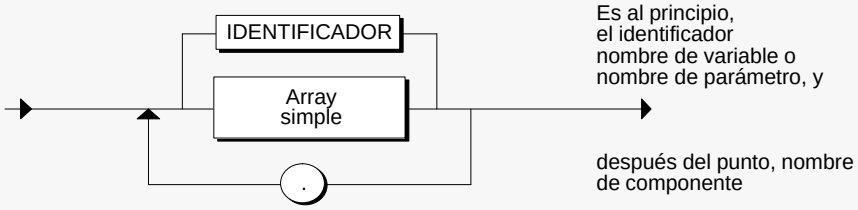
Regla	Diagrama sintáctico
Area de instrucciones	
INSTRUCCION	
Asignación de valor	
Variable ampliada	

Tabla C-5 Sintaxis de la tabla de instrucciones, continuación

Regla	Diagrama sintáctico
Variable simple	
Variable estructurada	

C.5 Asignación de valores

Tabla C-6 Sintaxis de las asignaciones de valores

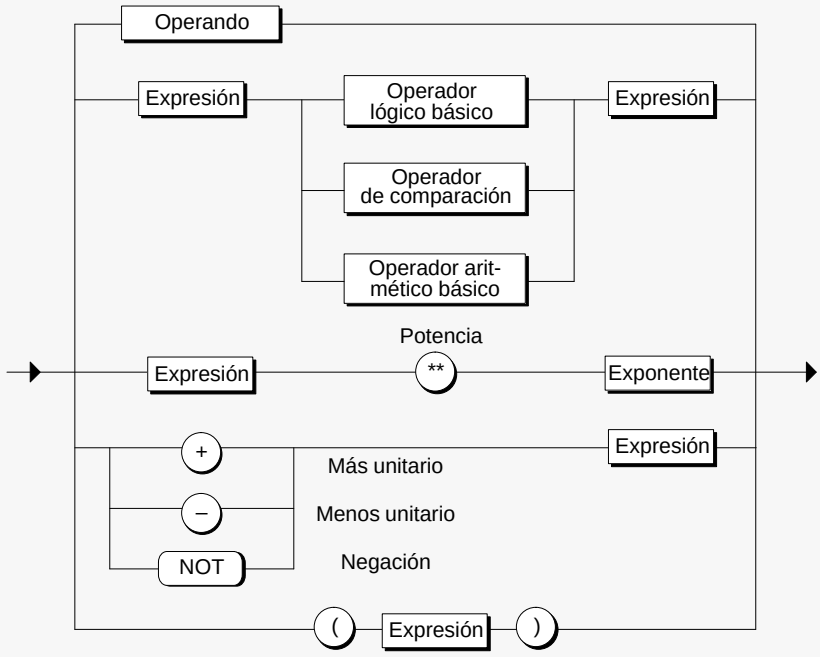
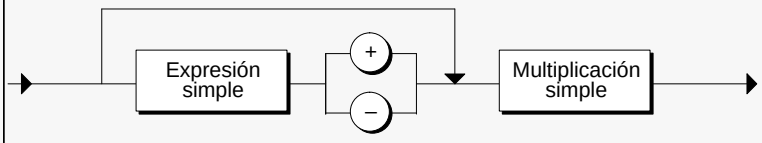
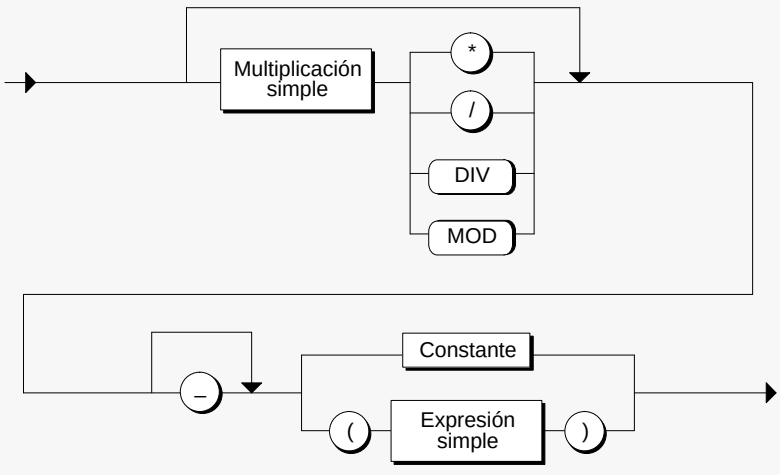
Regla	Diagrama sintáctico
Expresión	
Expresión simple	
Multiplicación simple	

Tabla C-6 Sintaxis de las asignaciones de valores, continuación

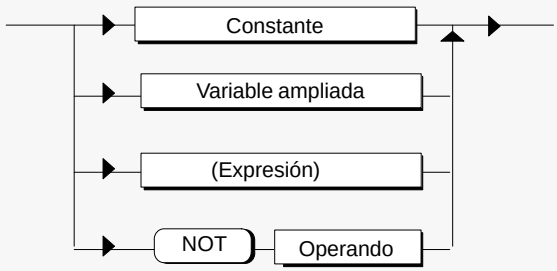
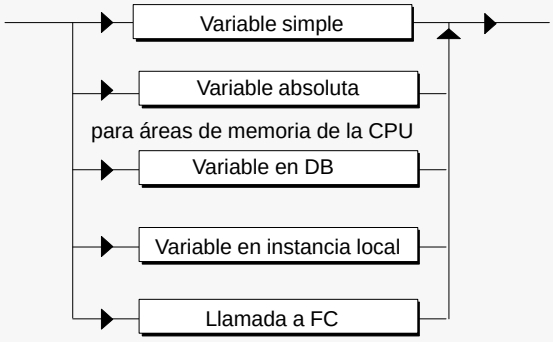
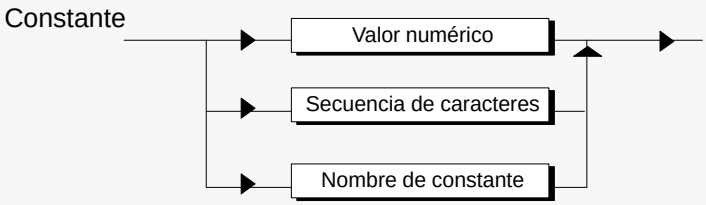
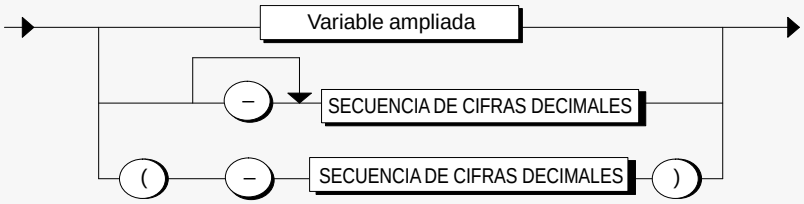
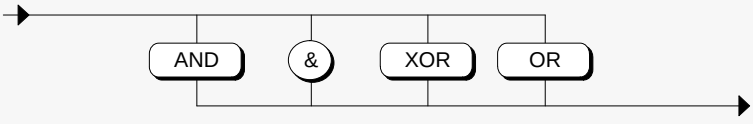
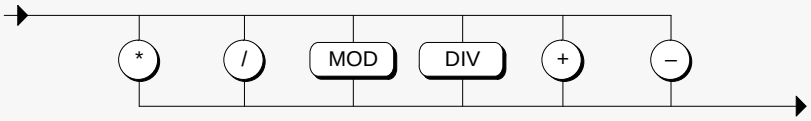
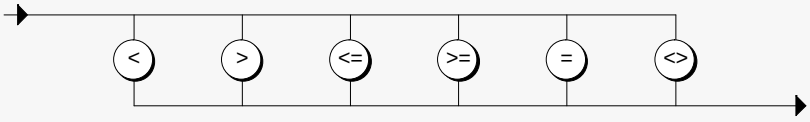
Regla	Diagrama sintáctico
Operando	
Variable ampliada	
Constante	
Exponente	
Operador lógico básico	

Tabla C-6 Sintaxis de las asignaciones de valores, continuación

Regla	Diagrama sintáctico
Operador aritmético básico	
Operador de comparación	

C.6 Llamada a funciones y bloques de función

Tabla C-7 Sintaxis de las llamadas

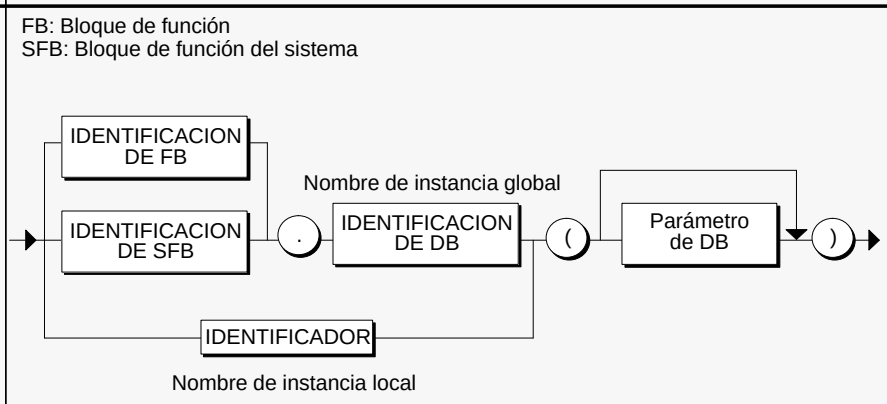
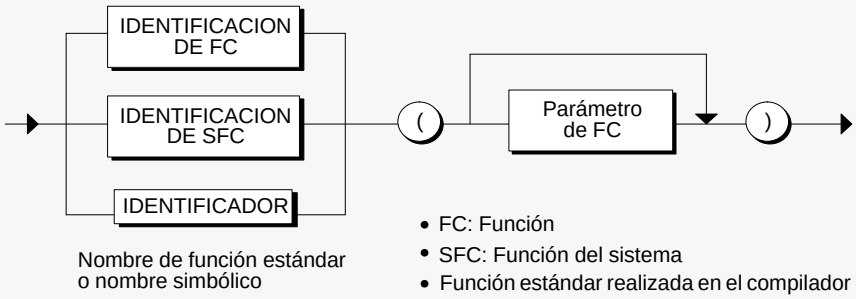
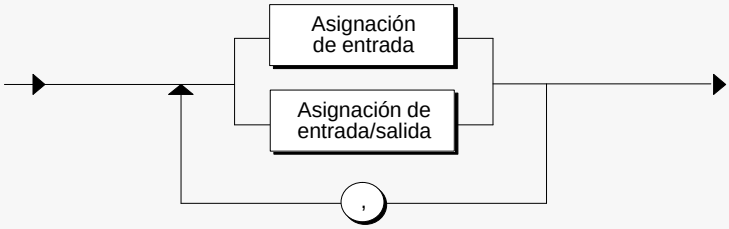
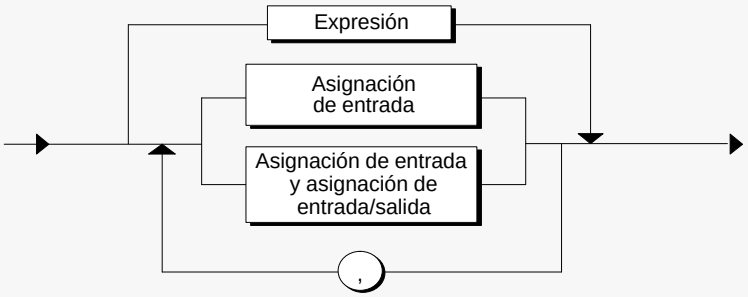
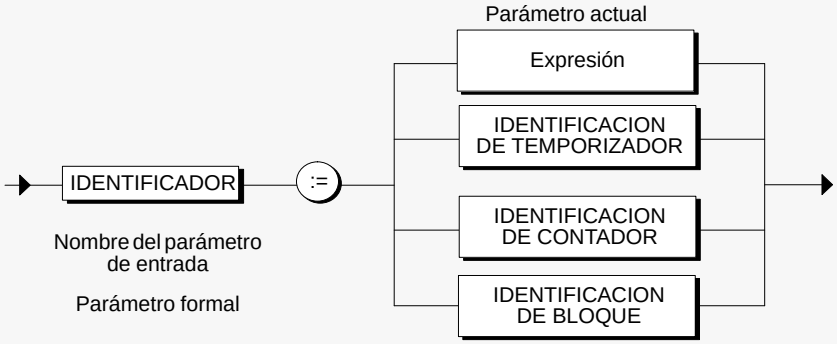
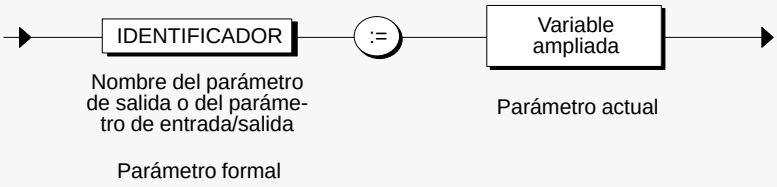
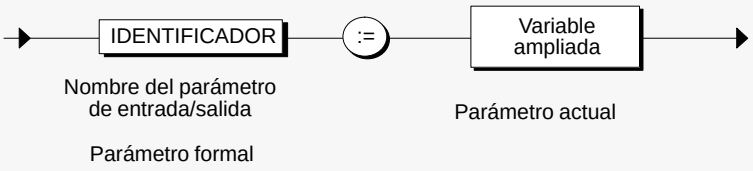
Regla	Diagrama sintáctico
Llamada a FB	
Llamada a función	
Parámetro de FB	
Parámetro de FC	

Tabla C-7 Sintaxis de las llamadas, continuación

Regla	Diagrama sintáctico
Asignación de entrada	
Asignación de salida y de entrada/salida	
Asignación de entrada/salida	

C.7 Instrucciones de control

Tabla C-8 Sintaxis de las instrucciones de control

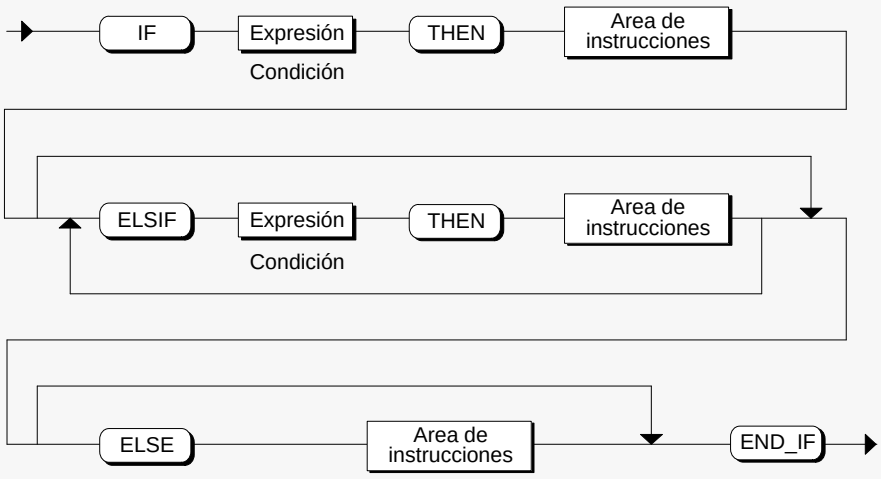
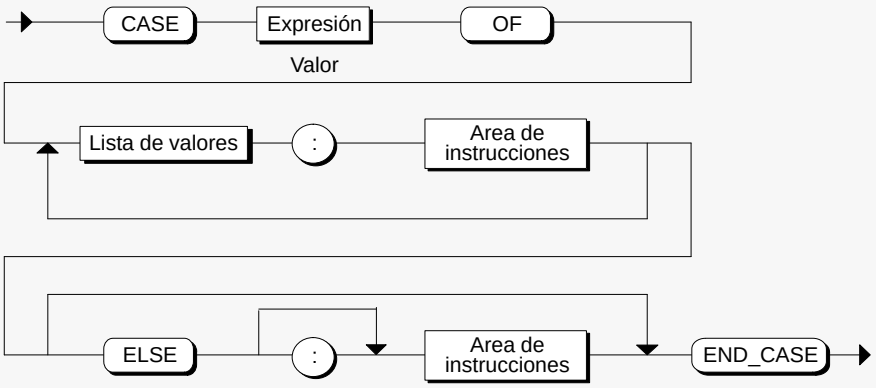
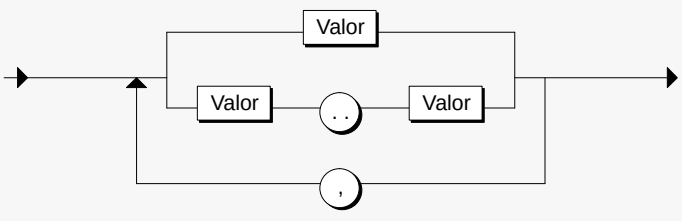
Regla	Diagrama sintáctico
Instrucción IF No olvide terminar la palabra clave END_IF con punto y coma.	
Instrucción CASE No olvide terminar la palabra clave END_CASE con punto y coma.	
Lista de valores	

Tabla C-8 Sintaxis de las instrucciones de control, continuación

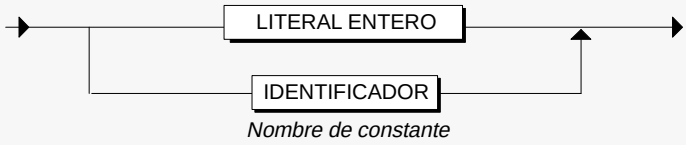
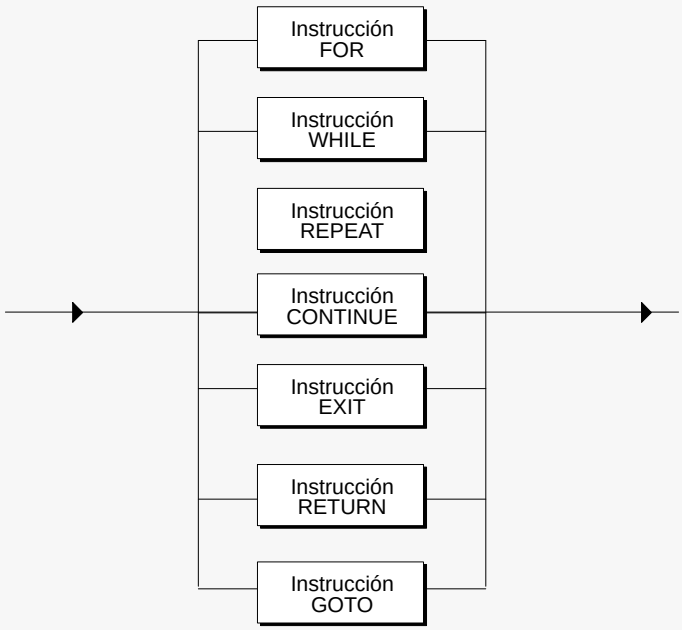
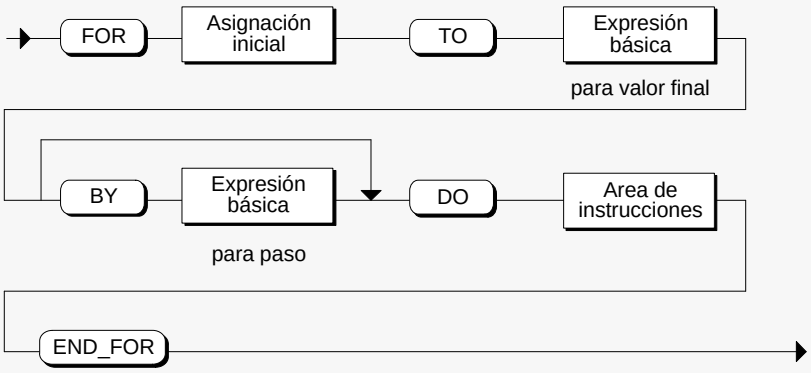
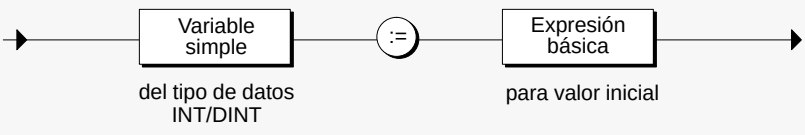
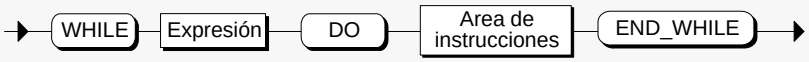
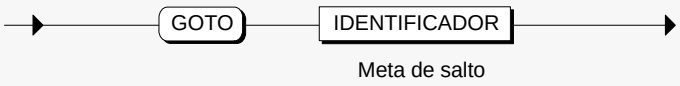
Regla	Diagrama sintáctico
Valor	
Instrucciones de repetición e instrucciones de salto	
Instrucción FOR No olvide terminar la palabra clave END_FOR con punto y coma.	
Asignación inicial	

Tabla C-8 Sintaxis de las instrucciones de control, continuación

Regla	Diagrama sintáctico
Instrucción WHILE No olvide terminar la palabra clave END_WHILE con punto y coma.	
Instrucción REPEAT No olvide terminar la palabra clave END_REPEAT con punto y coma.	
Instrucción CONTI-NUE	
Instrucción RETURN	
Instrucción EXIT	
Salto en el programa	

Bibliografía

D

/12/ Resumen técnico: *Autómata programable S7-300,*
Configuración y aplicación

/13/ Resumen técnico: *Autómata programable S7-400,*
Configuración y aplicación

/20/ Resumen técnico: *Sistemas de automatización SIMATIC,*
Programación con STEP 7

/30/ ABC de la programación: *Autómata programable S7-300,*
Iniciación a la configuración y programación

/70/ Manual: *Autómata programable S7-300,*
Configuración, instalación y datos de las CPU

/71/ Manual de referencia: *Sistemas de automatización S7-300 y M7-300*
Datos de los módulos

/72/ Lista de operaciones: *Autómata programable S7-300*

/100/ Manual: *Sistemas de automatización S7-400, M7-400,*
Configuración e instalación

/101/ Manual de referencia: *Sistemas de automatización S7-400, M7-400*
Datos de los módulos

/102/ Lista de operaciones: *Autómata programable S7-400,*
CPU 412, 413, 414, 416

/230/ Guia: *STEP 7,*
Guia para facilitar la transición de S5 a S7

/231/ Manual del usuario: *Software estándar para SIMATIC S7 y M7,*
STEP 7

/232/ Manual: *AWL para SIMATIC S7-300/400,*
Programación de bloques

/233/ Manual: *KOP para SIMATIC S7-300/400,*
Programación de bloques

/234/ Manual de programación: *Software de sistema para SIMATIC S7-300/400*
Diseño de programas

/235/ Manual de referencia: *Software de sistema para SIMATIC S7-300/400*
Funciones estándar y funciones de sistema

/236/ Manual: *FUP para S7-300/400,*
Programación de bloques

/237/ *Indice general,* STEP 7

/249/ Manual: *CFC para S7,*
Tomo 2

/251/ Manual: *GRAPH para S7-300/400,*
Programación de controles secuenciales

/254/ Manual: *CFC para S7,*
Tomo 1

/800/ *DOCPRO,*
Confección de documentación normalizada (sólo en CD)

/803/ Manual de referencia: *Software de sistema para S7-300/400,*
STEP 7 Funciones estándar, 2ª parte (sólo en CD)

Glosario

A

Archivo fuente

Un archivo fuente contiene códigos fuente (texto ASCII) que pueden crearse con cualquier editor de textos. Un archivo fuente se compila con un compilador (AWL, SCL) para generar un archivo de programa de usuario. Una fuente se deposita en el contenedor ”Fuentes” (SO) dentro del programa S7 o del programa M7.

Area de memoria

El SIMATIC S7, una unidad central de procesamiento tiene tres áreas de memoria: la memoria de carga, la memoria de trabajo y la memoria del sistema.

Array

Un array es un tipo de datos compuestos que está constituido por elementos de datos del mismo tipo. A su vez estos elementos de datos pueden ser simples o compuestos.

Atributo

Un atributo es una propiedad que puede asignarse, por ejemplo, a un identificador de bloque o a un nombre de variable. En SCL hay atributos para: título de bloque, estado de salida, protección de bloque, autor, nombre de bloque, familia de bloques.

Ayuda online

STEP 7 le ofrece la posibilidad de visualizar en la pantalla textos de ayuda sobre temas concretos mientras se está trabajando con el software de programación.

B

Bloque

Los bloques son secciones del programa de usuario delimitados por su función, por su estructura o por el uso para el que están destinados. En STEP 7 hay bloques lógicos (FB, FC, OB, SFC; SFB), bloques de datos (DB, SDB) y tipos de datos de usuario (UDT).

Bloque de datos (DB)

Los bloques de datos (DB) son áreas de datos del programa de usuario que contienen datos de usuario. Puede accederse a ellos desde todos los bloques lógicos. Los bloques de datos que están asignados a una determinada llamada de FB se denominan bloques de datos de instancia.

Bloques de datos de instancia (DB de instancia)

Un bloque de datos de instancia guarda los parámetros formales y los datos estáticos de los bloques de función. Un bloque de datos de instancia puede estar asignado a una llamada de FB o a una jerarquía de llamada de bloques de función. En SCL se genera automáticamente.

Bloque de datos del sistema (SDB)

Los bloques de datos del sistema son áreas de datos localizadas dentro de la unidad central de procesamiento, las cuales contienen ajustes del sistema y parámetros de módulos. Los bloques de datos del sistema se crean y modifican con la herramienta *Configuración S7*.

Bloque de función (FB)

Conforme a la norma IEC 1131-3, un bloque de función (FB) es un bloque lógico que contiene datos estáticos. Un FB ofrece la posibilidad de transferir parámetros dentro del programa de usuario. Por esta razón los bloques de función son idóneos para programar funciones complejas que se repiten con frecuencia, como p.ej. regulaciones o selección de modos de operación. Como un FB dispone de una memoria (bloques de datos de instancia) se puede acceder a sus parámetros (p.ej., salidas) en cualquier momento y en cualquier punto del programa de usuario.

Bloque de función del sistema (SFB)

Un bloque de función del sistema (SFB) es un bloque de función que está integrado dentro del sistema operativo de la CPU, y que se puede llamar en el programa de usuario de STEP 7 cuando se necesite.

Bloque de organización

Los bloques de organización constituyen el interface entre el sistema operativo de la CPU y el programa de usuario. En los bloques de organización se determina el orden de ejecución que sigue el programa de usuario.

Bloque lógico

En SIMATIC S7 un bloque lógico es un bloque que contiene una sección del programa de usuario STEP 7. Por el contrario, un bloque de datos sólo contiene datos. Existen los siguientes bloques lógicos: bloques de organización (OB), bloques de función (FB), funciones (FC), bloques de función del sistema (SFB) y funciones del sistema (SFC).

C

Carga en el sistema de destino

Carga de objetos cargables (p.ej., bloques lógicos) desde la unidad de programación en la memoria de carga de un módulo programable. Puede realizarse tanto a través de una unidad de programación conectada directamente como a través de PROFIBUS.

Carga en la PG

Carga de objetos cargables (p.ej., bloques lógicos) desde la memoria de carga de un módulo programable en la unidad de programación. Puede realizarse tanto a través de una unidad de programación conectada directamente como a través de PROFIBUS.

Categoría de bloque

En función de su contenido, los bloques se dividen en dos categorías: bloques lógicos y bloques de datos. El tipo de datos de usuario (UDT) puede considerarse como bloque de datos.

Comentario de bloque

Información complementaria sobre un bloque (p.ej. explicaciones acerca del proceso de automatizado) que no se carga en la memoria de trabajo de los autómatas programables SIMATIC S7.

Compilación orientada a la fuente

Al introducir datos orientados a la fuente, sólo se verifica la existencia de posibles errores de introducción durante la compilación. Sólo se genera un código ejecutable cuando no existe ningún error.

Compilador SCL

El compilador SCL es un compilador por lotes con el que puede compilarse a código máquina MC7 el programa previamente editado (fuente SCL). Los bloques generados se depositan en el contenedor ”Bloques” dentro del programa de usuario S7.

Compilar

Compilar es crear un programa de usuario ejecutable a partir de una fuente.

Constantes (simbólicas)

Las constantes con nombres simbólicos son comodines para valores constantes en los bloques lógicos. Se utilizan para aumentar la legibilidad de un programa.

Constantes (literales)

Las constantes literales son constantes cuyo valor y tipo depende de la forma de escritura. Se distinguen literales numéricos, literales de caracteres y literales de indicación de tiempos.

Contadores

Los contadores son componentes de la memoria del sistema de la CPU. El sistema operativo puede actualizar el contenido de estos contadores de forma desincronizada respecto al programa de usuario. Mediante instrucciones STEP 7 se puede definir la función exacta de los contadores (p.ej., contador incremental) e iniciar su ejecución (arranque).

Contenedor

Carpeta que se encuentra en el interface de usuario del Administrador SIMATIC. Esta carpeta se puede abrir y puede contener a su vez otros contenedores y objetos.

Convertir explícitamente

Convertir explícitamente significa insertar en el programa fuente una función de conversión. Al relacionar dos operandos de diferente tipo de datos, el usuario debe efectuar una conversión explícita: cambiando a otra categoría de tipos de datos (p.ej., de un tipo de datos de bit a un tipo de datos numérico) y, cuando el tipo de datos destino es de menor rango que el tipo de datos fuente, efectuando un cambio dentro de una misma clase de tipos de datos.

Convertir implícitamente

Convertir implícitamente significa que el compilador inserta automáticamente una función de conversión. Al enlazar dos operandos de diferentes tipos de datos se produce una conversión: cuando no se produce ningún cambio a otra categoría de tipos de datos y cuando el tipo de datos destino no es de menor rango que el tipo de datos fuente.

D

Datos estáticos

Los datos estáticos son datos locales de un bloque de función que se guardan en el bloque de datos de instancia, por lo que se conservan hasta que se vuelve a ejecutar el bloque de función.

Datos globales

Los datos globales son áreas de memoria de la CPU a los que se puede acceder desde cualquier punto del programa (p.ej., marcas).
Los datos globales son accesibles desde cualquier bloque lógico (FC, FB, OB). Dentro de los datos globales se distinguen: marcas (M), entradas (E), salidas (A), temporizadores, contadores y elementos de bloques de datos (DB). A los datos globales se puede acceder con direccionamiento simbólico o absoluto.

Datos locales

Los datos locales son datos asignados a un bloque lógico que están definidos en su tabla de declaración. Dependiendo del bloque en cuestión, abarcan a los parámetros formales, los datos estáticos y los datos temporales.

Datos de usuario

El intercambio de datos de usuario entre una unidad central de procesamiento y un módulo de señales, un módulo de función o módulos de comunicaciones se realiza por medio de la imagen del proceso o mediante accesos directos. Datos de usuario pueden ser los siguientes: señales digitales y analógicas de entrada/salida de los módulos de señales, informaciones de control y de estado de los módulos de función.

Datos temporales

Los datos temporales pertenecen localmente a un bloque lógico y **no** ocupan área estática de memoria, puesto que se depositan en la pila de la CPU. Su valor sólo se conserva durante una ejecución del bloque.

Declaración del tipo de datos

En la declaración del tipo de datos el usuario puede declarar tipos de datos de usuario.

Declaración de variables

La declaración de variables incluye la especificación de un nombre simbólico, de un tipo de datos y, en algunos casos, del valor predefinido, la dirección y el comentario.

Decompilación

Mediante la compilación en AWL es posible cargar y visualizar con cualquier PG/PC el módulo cargado en la CPU. Pueden faltar determinadas partes del bloque (p.ej., símbolos y comentarios).

Depurador SCL

El depurador SCL es un depurador de lenguaje de alto nivel que permite localizar los errores lógicos de programación en el programa de usuario creado con SCL.

Direccionamiento absoluto

En el direccionamiento absoluto se indica la dirección del operando a ejecutar. Ejemplo: con la dirección A 4.0 se denomina al bit 0 del byte 4 de la imagen de proceso de las salidas.

Direccionamiento simbólico

En el direccionamiento simbólico el operando a ejecutar se indica simbólicamente (en vez de utilizar una dirección).

E

Editor SCL

El editor SCL es un editor hecho a la medida del lenguaje SCL que permite crear la fuente SCL.

Enable (EN)

En STEP 7 cada bloque tiene una entrada ”Enable” (EN). Con esta entrada se llama al bloque. Si en EN existe una señal 1, se llama al bloque; si la señal es 0 no se efectúa la llamada al bloque.

Enable out (ENO)

En STEP 7 cada bloque tiene una salida ”Enable Output” (ENO). Dentro del bloque el usuario puede relacionar la entrada ”Enable” con un valor interno (Y). El resultado se asigna automáticamente a la salida ENO. En el caso de llamadas encadenadas a bloques, ENO permite que la edición del bloque consecutivo dependa de que se haya editado correctamente el bloque precedente.

Entero (INT)

Entero (INT) es uno de los tipos de datos simples. Se representa por medio de un número entero de 16 bits.

Entrada orientada a la fuente

En la programación con SCL es posible la entrada de un programa STEP 7 orientada a la fuente. La entrada de un programa puede realizarse con cualquier editor de textos. El auténtico código del programa no se genera hasta que se ejecuta la compilación. Entonces también se detectan posibles errores. Este tipo de entrada es idónea para la representación simbólica de programas estándar.

Al efectuar una entrada orientada a la fuente se editan en un archivo de texto los bloques o la totalidad del programa de usuario. La sintaxis no se comprueba hasta que se realiza la compilación. La entrada orientada a la fuente se utiliza en SCL.

Estado del bloque

⇒ *Observación continua*

Estructura (STRUCT)

Una estructura es un tipo de datos compuestos formado de elementos de datos de distintos tipos. Estos elementos de datos pueden ser simples y compuestos.

Expresión

En SCL una expresión sirve para procesamiento de datos. Se distinguen expresiones aritméticas, expresiones lógicas y expresiones de comparación.

F

Fuente

Una fuente (archivo de texto) incluye un código fuente (texto en ASCII) que puede crearse con cualquier editor de texto. Una fuente se compila con un compilador (AWL, SCL) para generar un programa de usuario ejecutable. Una fuente se deposita en el contenedor ”Fuentes” dentro del programa S7.

Función (FC)

De acuerdo con la norma IEC 1131-3, una función (FC) es un bloque lógico **sin** datos estáticos. Una función ofrece la posibilidad de transferir parámetros dentro del programa de usuario. Por esta razón las funciones resultan ser idóneas para programar funciones complejas que se repiten con frecuencia; p.ej., los cálculos.

Función de sistema (SFC)

Una función de sistema (SFC) es una función integrada dentro del sistema operativo de la CPU que se puede llamar en el programa de usuario de STEP 7 cuando se necesite.

I

Identificador

Se denominan identificadores a objetos de lenguaje de SCL. Existen las siguientes categorías: identificadores estándar, nombres predefinidos y palabras clave, identificadores absolutos (o identificadores de operando), nombres de libre elección (p.ej. para variables y metas de salto), y nombres simbólicos generados con otras herramientas (editor de símbolos).

Identificador de operando

Un identificador de operando es una parte del operando de una operación que contiene informaciones (p.ej., el área de memoria en el que la operación tiene un valor (objeto de datos)), con el que el usuario ejecuta una relación lógica, o la magnitud de un valor (objeto de datos) con la que el usuario ejecuta una relación lógica. En la instrucción ”Valor:= EB10”, ”EB” es el indicador de operando (”E” representa el rango de entrada de la memoria, ”B” representa un byte dentro de este área).

Imagen del proceso

Los estados de señal de los módulos digitales de entradas y salidas se depositan dentro de la CPU en una imagen de proceso. Se distingue entre la imagen del proceso de las entradas (PAE) y la imagen del proceso de las salidas (PAA).

Imagen del proceso de entradas (PAE)

Antes de la ejecución del programa de usuario el sistema operativo lee en los módulos de entradas la imagen de proceso de las entradas.

Imagen del proceso de salidas (PAA)

Al final del programa de usuario el sistema operativo transfiere la imagen del proceso de las salidas a los módulos de salidas.

Instancia

Se denomina ”Instancia” a la llamada de un bloque de función, asignándole a éste un bloque de datos de instancia o una instancia local. Si en el programa de usuario STEP 7 se llama n veces a un bloque de función, cada vez con un parámetro diferente y nombre de bloque de datos de instancia diferente, existen n instancias.

```
FB13.DB3 (P3:=...), FB13.DB4 (P4:=...),
FB13.DB5 (P5:=...), .............FB13.DBn (Pn:=...)
```

Instancia local

Una instancia local se define en la parte de la variable estática de un bloque de función. En lugar de utilizar todo el bloque de datos de instancia como área de datos para el bloque de función, sólo se utiliza una parte local, a la que se llama con el nombre de instancia local.

Instrucción

Una instrucción es la menor unidad independiente dentro de un programa de usuario que haya sido creado en un lenguaje de texto. Constituye una norma para que el procesador opere de un modo determinado.

Instrucción CASE

Esta instrucción es una instrucción de ramificación. Dependiendo del valor de una expresión de selección, sirve para seleccionar una sección del programa dentro de n opciones posibles.

Instrucción CONTINUE

Termina un bloque de ejecución y comienza de nuevo con el siguiente valor de la variable en ejecución.

Instrucción EXIT

Interrupción de un bucle en ejecución.

Instrucción FOR

Una instrucción FOR sirve para repetir una secuencia de instrucciones hasta que la variable en ejecución se sitúe dentro de un valor especificado.

Instrucción GOTO

Una instrucción GOTO provoca el salto inmediato a una marca especificada.

```
FB13.DB1 (P1:=...), FB13.DB3(P3:=...),
FB13.DB2 (P2:=...), .....FB13.DBn (Pn:=...).
```

Instrucción REPEAT

Una instrucción REPEAT sirve para repetir una secuencia de instrucciones hasta llegar a una condición de interrupción.

Instrucción RETURN

Esta instrucción provoca el abandono del bloque actual.

Interface de llamada

Se define por los parámetros de entrada, de salida y de entrada/salida (parámetros formales) de un bloque en el programa de usuario STEP 7. Al llamar al bloque estos parámetros son sustituidos por los parámetros actuales.

J

Jerarquía de llamada

Para poder editar cualquier bloque primero hay que llamarlo. El orden y el anidamiento de estas llamadas se denomina jerarquía de llamada.

L

Lista de instrucciones (AWL)

La lista de instrucciones (AWL) es un lenguaje de programación textual orientado hacia la máquina.

LL

Llamada a un bloque

Arranque de un bloque en el programa de usuario de STEP 7; los bloques de organización son llamados básicamente por el sistema operativo; todos los demás bloques son llamados por el programa de usuario de STEP 7.

M

Marca (M)

Area de memoria localizada en la memoria del sistema de una CPU SIMATIC S7. A esta memoria se tiene acceso de escritura y de lectura (con bits, bytes, palabras y palabras dobles). El usuario puede utilizar el área de marcas para guardar resultados intermedios.

Memoria del sistema (Área del sistema)

La memoria del sistema está integrada en la unidad central de procesamiento y ha sido diseñada como memoria RAM. En la memoria del sistema están depositadas las áreas de operandos (p.ej., temporizadores, contadores, marcas) y las áreas de datos que el sistema operativo necesita a nivel interno (p.ej. búfer de comunicación).

Multiinstancia

Al utilizar multiinstancias, el bloque de datos de instancias contiene los datos de varios bloques de función de una jerarquía de llamada.

N

Nemotécnica

La nemotécnica es una representación abreviada de los operandos y de las operaciones de programación dentro del programa (p.ej., ”E” significa ”entrada”). STEP 7 soporta la representación IEC (basada en el idioma inglés) y la representación SIMATIC (basada en la expresión alemana de las operaciones y de las convenciones para el direccionamiento en SIMATIC).

Nonterminal

Un nonterminal es un elemento compuesto descrito por otras reglas léxicas o sintácticas.

Número real Un número real, también denominado número en coma flotante, es un número positivo o negativo que contiene un valor decimal (p.ej., 0.339 o –11.1).

O

Observación continua Modo de test de SCL. Al observar continuamente un programa puede efectuar un test sobre un grupo de instrucciones. Este grupo de instrucciones también se llama área de observación.

Offline ”Offline” designa el estado operativo en el que la unidad de programación no tiene comunicación (física, lógica) con el autómata programable.

Online ”Online” designa el estado operativo en el que la unidad de programación está unida (físicamente, lógicamente) con el autómata programable.

Operación Una operación es parte de una instrucción que especifica lo que tiene que hacer el procesador.

Operando Un operando es una parte de una instrucción que especifica con qué debe hacer algo el procesador. Puede direccionarse tanto absoluta como simbólicamente.

P

Palabra clave En SCL las palabras clave se utilizan para identificar el comienzo de un bloque, para marcar secciones dentro de la tabla de declaración, para identificar instrucciones y para comentarios y atributos.

Palabra de estado La palabra de estado es un componente de la ficha de la unidad central de procesamiento. En la palabra de estado se encuentran informaciones de estado y de error, producidas en relación con la ejecución de comandos de STEP 7. El usuario puede leer y escribir los bits de estado; los bits de error sólo permiten lectura.

PARADA El estado operativo PARADA se alcanza a partir del estado operativo RUN cuando lo solicita la unidad de programación. En este estado operativo es posible efectuar funciones de test especiales.

Parámetro En SCL: Variable de un bloque lógico (parámetro actual, parámetro formal)

Parámetros actuales Los parámetros actuales sustituyen a los parámetros formales cuando se llama a un bloque de función (FB) o a una función (FC).
Ejemplo: el parámetro formal ”Start” será sustituido por el parámetro formal ”E 3.6”.

Parámetros de entrada

Solamente las funciones y los bloques de función tienen parámetros de entrada. Con ayuda de los parámetros de entrada se transfieren datos al bloque llamado, para el procesamiento de los mismos.

Parámetros de entrada/salida

Las funciones y los bloques de función tienen parámetros de entrada/salida. Con estos parámetros se transfieren datos al bloque llamado; allí son procesados, y los resultados se vuelven a depositar a continuación en la misma variable.

Parámetros de salida

Con el parámetro de salida de un bloque, en el programa de usuario se transfieren los resultados al bloque que efectúa la llamada.

Parámetros formales

Un parámetro formal es un comodín para el parámetro ”real” (parámetro actual) que utilizan los bloques lógicos parametrizables. En el caso de los bloques de función (FB) y de las funciones (FC), es el usuario quien define los parámetros formales; en el caso de los bloques de función del sistema (SFB) y de las funciones del sistema (SFC), los parámetros formales ya han sido establecidos. Al llamar a un bloque se asigna al parámetro formal un parámetro actual, de modo que el bloque al que se ha llamado operará con el valor actual de este parámetro. Los parámetros formales forman parte de los datos locales del bloque, y se subdividen en parámetros de entrada, parámetros de salida y parámetros de entrada/salida.

Parte de declaración

Aquí se declaran los datos locales de un bloque lógico.

Paso individual

El paso individual es una función de test del depurador SCL que permite ejecutar el programa instrucción por instrucción, y observarlo en la ventana de resultado.

Programa de usuario

El programa de usuario contiene todas las instrucciones y declaraciones necesarias para realizar el procesamiento de señales con el que poder controlar una instalación o un proceso. Está asignado a un módulo programable (p.ej. CPU, FM) y se puede estructurar en unidades menores (bloques). ⇒ *Bloque*

Programa de usuario S7

El programa de usuario S7 se encuentra en el contenedor ”Bloques”. Contiene bloques que se cargan en un módulo S7 programable (p.ej., CPU), donde son ejecutables, con el fin de controlar una instalación o un proceso.

Programación estructurada

Para solucionar tareas de automatización complejas, el programa de usuario se subdivide en secciones de programa cerradas individuales (bloques). El programa de usuario se organiza siguiendo la estructura funcional o tecnológica de las instalaciones.

Programación simbólica

El lenguaje de programación SCL permite utilizar secuencias de caracteres simbólicos en lugar de operandos. Por ejemplo, el operando A1.1 puede sustituirse por ”Válvula 17”. La lista de símbolos de STEP 7 constituye la relación entre el operando y la secuencia de caracteres simbólicos asignada.

Protección de bloque

Se denomina protección de bloque al medio por el cual se puede proteger un bloque contra una posible decompilación en aquellos casos en los que la fuente del bloque haya sido compilada con la palabra clave ”KNOW_HOW_PROTECTED”.

Proyecto

Un proyecto es un contenedor para todos los objetos de una solución a una tarea de automatización, independientemente de la cantidad de estaciones, de módulos y de la interconexión de los mismos en una red.

Punto de parada

Con esta función se logra que la unidad central de procesamiento (CPU) pase al estado operativo PARADA en puntos definidos del programa. Al llegar a un punto de parada se pueden realizar funciones de test como p.ej., la ejecución paso a paso de un comando o de una variable de estado.

R

Reglas léxicas

El nivel inferior de las reglas formales que describen el lenguaje SCL está compuesto por las reglas léxicas. Al aplicarlas no existe libertad de formato, es decir, no está permitida la complementación (p.ej., con blancos y caracteres de control).

Reglas sintácticas

El nivel superior de las reglas formales que describen el lenguaje SCL está formado por las reglas sintácticas. Pueden usarse con libertad de formato, es decir: p.ej. pueden incluirse blancos y caracteres de control.

Representación BCD

En STEP 7, a nivel interno de la CPU los temporizadores y contadores se especifican sólo en formato BCD. BCD es el acrónimo de ”*Binär–Code für Dezimaalzhalen* = código binario para números decimales”.

RUN

En estado operativo RUN se ejecuta el programa de usuario, y la imagen del proceso se actualiza cíclicamente. Todas las salidas digitales están habilitadas.

RUN-P

El estado operativo RUN-P es equivalente al sistema operativo RUN, con la diferencia de que en el estado operativo RUN-P están permitidas todas las funciones de la unidad de programación, sin limitación alguna.

S

SCL

Lenguaje de alto nivel similar al Pascal, conforme a la norma DIN EN–61131–3 (int. IEC 1131–3) que se utiliza para la programación de tareas complejas en un PLC, p.ej. algoritmos y tareas de procesamiento de datos. Es abreviatura de ”Structured Control Language”.

Símbolo

Un símbolo es un nombre que el usuario define atendiendo a determinadas normas sintácticas. Una vez que se ha determinado lo que simboliza este nombre (p.ej., variable, tipo de datos, bloque), podrá utilizarse en la programación y en el manejo y visualización. Ejemplo: operando: E 5.0; tipo de datos: BOOL; símbolo: pulsador paro de emergencia.

Single Step

⇒ *Paso individual*

T

Tabla de símbolos

Tabla para asignar símbolos (nombres) a direcciones de datos globales y bloques. Ejemplos: paro de emergencia (símbolo); E1.7 (dirección), o regulador (símbolo) SFB 24 (bloque).

Tabla de variables

En la tabla de variables se recompilan las variables que se desea observar y controlar, incluyendo las especificaciones correspondientes de formato.

Temporizadores

Los temporizadores son componentes de la memoria del sistema de la CPU. El sistema operativo actualiza el contenido de estos temporizadores de forma asíncrona con respecto al programa de usuario. Por medio de instrucciones de STEP 7 se determina exactamente la función que tendrá un temporizador (p.ej., retardo a la conexión) y se impulsa su ejecución (arranque).

Terminal

Un terminal es un elemento básico de una regla léxica o sintáctica que no se explica mediante otra regla, sino verbalmente. Un terminal puede ser, p.ej,. una palabra clave o un carácter aislado.

Tiempo de ciclo

El tiempo de ciclo es el tiempo que necesita la CPU para ejecutar una vez el programa de usuario.

Tiempo de vigilancia del ciclo

Si el tiempo de ejecución del programa de usuario sobrepasa el tiempo de vigilancia del ciclo predeterminado, el sistema operativo genera un mensaje de error y la CPU pasa a estado operativo STOP.

Tipo de bloque

La arquitectura de bloques de STEP 7 reconoce los siguientes tipos de bloques: bloque de organización, bloque de función, funciones, bloque de datos y bloque de función del sistema, funciones del sistema y bloque de datos del sistema. ⇒ *Bloque*

Tipo de datos

Con ayuda de un tipo de datos se determina cómo debe utilizarse el valor de una variable o de una constante en el programa de usuario.
En SCL el usuario dispone de tres tipos de datos:

- tipos de datos simples,
- tipos de datos estructurados, y
- tipos de datos de usuario (UDT).

Tipo de datos estructurados

Se distingue entre estructuras y arrays. Las ”estructuras” están compuestas por diferentes tipos de datos compuestos diferentes (p.ej. tipos de datos simples). Los ”arrays” están formados por varios elementos del mismo tipo de datos. Los tipos de datos `STRING` y `DATE_AND_TIME` también son tipos de datos estructurados.

Tipo de datos de usuario

Los tipos de datos de usuario los define el propio usuario mediante la declaración del tipo de datos. Estos datos tienen su propio nombre distintivo, por lo que son de uso múltiple. Por ejemplo, un tipo de datos de usuario se puede aprovechar para crear varios bloques de datos que tengan la misma estructura (p.ej., reguladores).

Tipo de datos simples

Los tipos de datos simples son tipos de datos predefinidos conforme a la norma IEC 1131-3. Ejemplos: el tipo de datos ”`BOOL`” define una variable binaria (”`Bit`”); el tipo de datos ”`INT`” define una variable fija de 16 bits.

Tipo de declaración

El tipo de declaración especifica la forma en que el bloque deberá utilizar un parámetro o una variable local. Hay parámetros de entrada, parámetros de salida y parámetros de entrada/salida, además de variables estáticas y temporales.
⇒ *Variables estáticas* ⇒ *Variables temporales*

Tipo de parámetro

Un tipo de parámetro es un tipo de datos específico para temporizadores, contadores y bloques. Puede utilizarse en parámetros de entrada de bloques de función **y** de funciones, pero sólo en parámetros de entrada/salida en el caso de bloques de función, para transferir temporizadores, contadores y bloques al bloque llamado.

U

UDT

⇒ *Tipo de datos de usuario*

V

Variable

Una variable define un dato de contenido variable que puede utilizarse en el programa de usuario de STEP 7. Una variable consta de un operando (p.ej. M 3.1) y un tipo de datos (p.ej. Bool) y puede identificarse con un símbolo (p.ej. BAND_EIN). Se declara en la tabla de declaración.

Variable OK

La variable OK sirve para indicar si la ejecución de un bloque es correcta o incorrecta. Es un tipo de datos BOOL global.

Indice alfabético

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

Siemens AG
A&D AS E46

Östliche Rheinbrückenstr. 50
D-76181 Karlsruhe
R.F.A.

Remitente:

Nombre: _

Cargo: _

Empresa: _

Calle: _

Código postal: _

Población: _

País: _

Teléfono: _

Indique el ramo de la industria al que pertenece:

- ❒ Industria del automóvil
- ❒ Industria química
- ❒ Industria eléctrica
- ❒ Industria alimentaria
- ❒ Control e instrumentación
- ❒ Industria mecánica
- ❒ Industria petroquímica
- ❒ Industria farmacéutica
- ❒ Industria del plástico
- ❒ Industria papelera
- ❒ Industria textil
- ❒ Transportes
- ❒ Otros _ _ _ _ _ _ _ _ _ _ _

Observaciones/sugerencias

Sus observaciones y sugerencias nos permiten mejorar la calidad y utilidad de nuestra documentación. Por ello le rogamos que rellene el presente formulario y lo envie a Siemens.

Responda por favor a las siguientes preguntas dando una puntuación comprendida entre 1 = muy bien y 5 = muy mal

1. ¿Corresponde el contenido del manual a sus exigencias? ☐
2. ¿Resulta fácil localizar las informaciones requeridas? ☐
3. ¿Es comprensible el texto? ☐
4. ¿Corresponde el nivel de los detalles técnicos a sus exigencias? ☐
5. ¿Qué opina de la calidad de las ilustraciones y tablas? ☐

En las líneas siguientes puede exponer los problemas concretos que se le hayan planteado al manejar el manual:

_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _