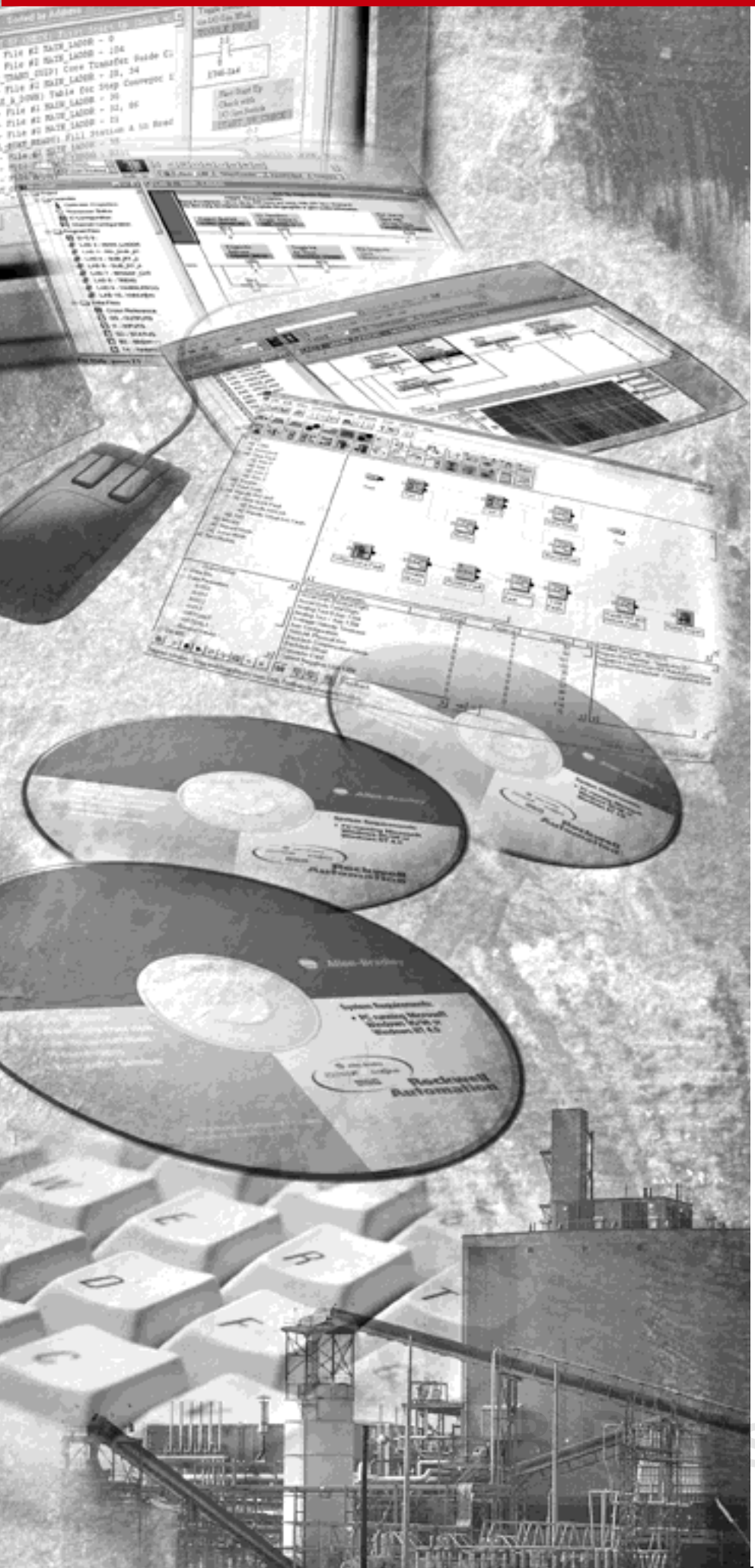


Allen-Bradley

Utilidad de importación/ exportación del controlador Logix5550

1756-L1, -L1M1, -L1M2, -L1M3

Manual de referencia

Rockwell Automation

Información importante para el usuario

Debido a la variedad de usos de los productos descritos en esta publicación, las personas responsables de la aplicación y uso de este equipo deben asegurarse de que se hayan seguido todos los pasos necesarios para que cada aplicación y uso cumpla con todos los requisitos de rendimiento y seguridad, incluyendo leyes, reglamentos, códigos y normas aplicables.

Los ejemplos de ilustraciones, gráficos, programas y esquemas mostrados en esta guía tienen la única intención de ilustrar el texto. Debido a las muchas variables y requisitos asociados con cualquier instalación particular, Allen-Bradley no puede asumir responsabilidad u obligación (incluyendo responsabilidad de propiedad intelectual) por el uso real basado en los ejemplos mostrados en esta publicación.

La publicación SGI-1.1 de Allen-Bradley *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (disponible a través de la oficina regional de Allen-Bradley), describe algunas diferencias importantes entre dispositivos de estado sólido y dispositivos electromecánicos, las cuales deben tenerse en consideración al usar productos tales como los descritos en esta publicación.

En este manual hacemos anotaciones para informarle de consideraciones de seguridad:

ATENCION

Identifica información sobre prácticas o circunstancias que pueden conducir a lesiones personales o la muerte, o a daños materiales o pérdidas económicas.

Las notas de “Atención” le ayudan a:

- Identificar un peligro
- Evitar un peligro
- Reconocer las consecuencias

IMPORTANTE

Identifica información importante para la aplicación y entendimiento correctos del producto.

Sírvase tomar nota de que en esta publicación se usa el punto decimal para separar la parte entera de la decimal de todos los números.

Resumen de los cambios

Este documento describe cómo usar la versión 1.25 de la utilidad de importación/exportación que se incluye con el software de programación RSLogix5000, versión 2.25.00 y posteriores.

IMPORTANTE

La versión 2.25 y posteriores del software de programación RSLogix son compatibles con los archivos .L5K creados anteriormente a la versión 1.25 de la utilidad de importación/exportación. La utilidad de importación/exportación convierte archivos anteriores a la versión 1.25.

Las versiones del software de programación RSLogix5000 anteriores a la versión 2.25 no son compatibles con los archivos .L5K creados con la versión 1.25 de la utilidad de importación/exportación.

Los cambios incorporados en este documento incluyen:

- Los ítems CommMethod y ConfigMethod en el componente MODULE ahora almacenan cadenas numéricas en lugar de cadenas de texto

- Estas nuevas instrucciones están disponibles:
  - Absolute Value (ABS)
  - Modulo (MOD)
  - Truncate (TRN)
  - Motion Axis Postion Cam (MAPC)
  - Motion Axis Time Cam (MATC)
  - Motion Calculate Cam Profile (MCCP)

Notas:

Capítulo 1

Importación y exportación de archivos

Capítulo 2

Estructuración de un formato de archivo de importación/exportación (.L5K) completo

Capítulo 3

Creación de un archivo de importación/exportación completo

Capítulo 4

Introducción de la lógica de escalera

Capítulo 5

Estructuración del formato de archivo de importación/ exportación (.CSV) de tags

Capítulo 6

Ejemplo de importación/ exportación

Apéndice A

Consideraciones para usar Microsoft Excel para editar un archivo .CSV

Notas:

Importación y exportación de archivos

Introducción

Este documento describe cómo usar la versión 1.25 de la utilidad de importación/exportación que se incluye con el software de programación RSLogix5000, versión 2.25.00 y posteriores.

IMPORTANTE

La versión 2.25 y posteriores del software de programación RSLogix es compatible con los archivos .L5K creados anteriormente a la versión 1.25 de la utilidad de importación/exportación. La utilidad de importación/exportación convierte archivos anteriores a la versión 1.25.

Las versiones del software de programación RSLogix5000 anteriores a la versión 2.25 no son compatibles con los archivos .L5K creados con la versión 1.25 de la utilidad de importación/exportación.

Con un controlador Logix5550, se puede importar/exportar un proyecto entero o se puede importar/exportar tags dentro de un proyecto.

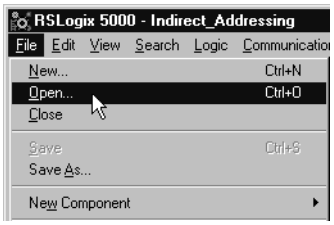
Cuando se importa o se exporta un proyecto, se usa el proyecto completo. El archivo de texto es un archivo de importación/exportación que incluye definiciones de tags, datos, lógica de escalera, información de configuración de E/S e información de configuración del controlador. Si usted importa o exporta tags, el archivo de texto es un archivo de importación/exportación parcial que incluye sólo definiciones de tags y comentarios de tags.

La estructura del archivo de importación/exportación depende de si usted realiza una operación de importación/exportación completa o parcial. Además, hay consideraciones diferentes para operaciones de importación/exportación parcial. Este capítulo muestra cómo realizar las operaciones de importación/exportación y describe las consideraciones.

Importación de un archivo de texto a un proyecto

Usted puede importar la información del controlador desde un archivo de texto guardado (que tenga una extensión .L5K). Esto le permite usar cualquier editor de texto para crear un proyecto.

1. **Seleccione File → Open.**



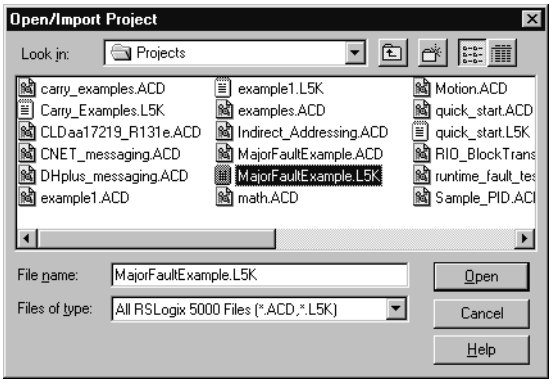
2. **Seleccione el archivo de texto.** El archivo de texto debe tener una extensión .L5K.

Seleccione el archivo a importar.

Como opción predeterminada, el software señala la carpeta \RSLogix5000\Project. Usted puede cambiar la opción predeterminada mediante Tool → Options.

Especifique el nombre del archivo a importar.

Haga clic en Open.

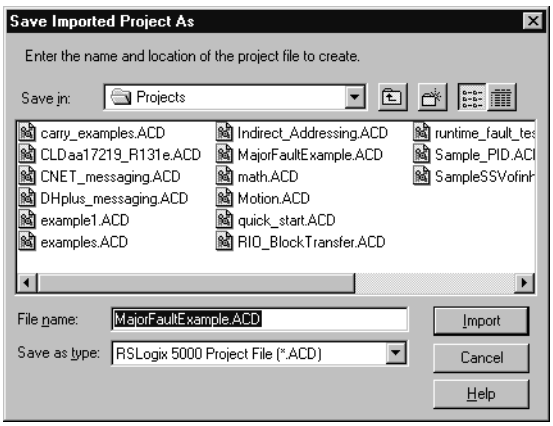


3. **Especifique el nombre y ubicación del proyecto**

Especifique la ubicación del proyecto.

Especifique el nombre del proyecto.

Haga clic en Import.



Si importó un proyecto que tiene forzados, el proyecto establece de manera predeterminada Forces Disabled (Forzados inhabilitados), aunque el proyecto haya sido exportado con los forzados habilitados.

Para obtener más información acerca de la estructura del archivo de importación/exportación completo, vea:

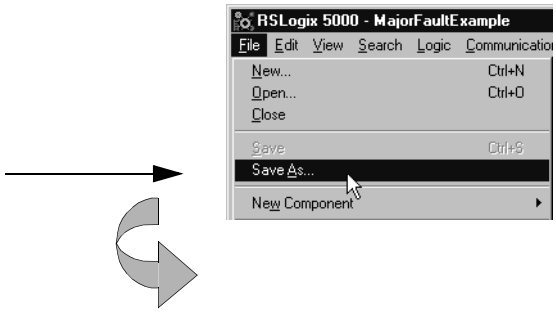
Para obtener información acerca de:	Vea el capítulo:
estructurar un archivo de importación/ exportación completo	2
crear un archivo de importación/exportación completo	3
introducir la lógica	4

Exportación de un proyecto completo a un archivo de texto

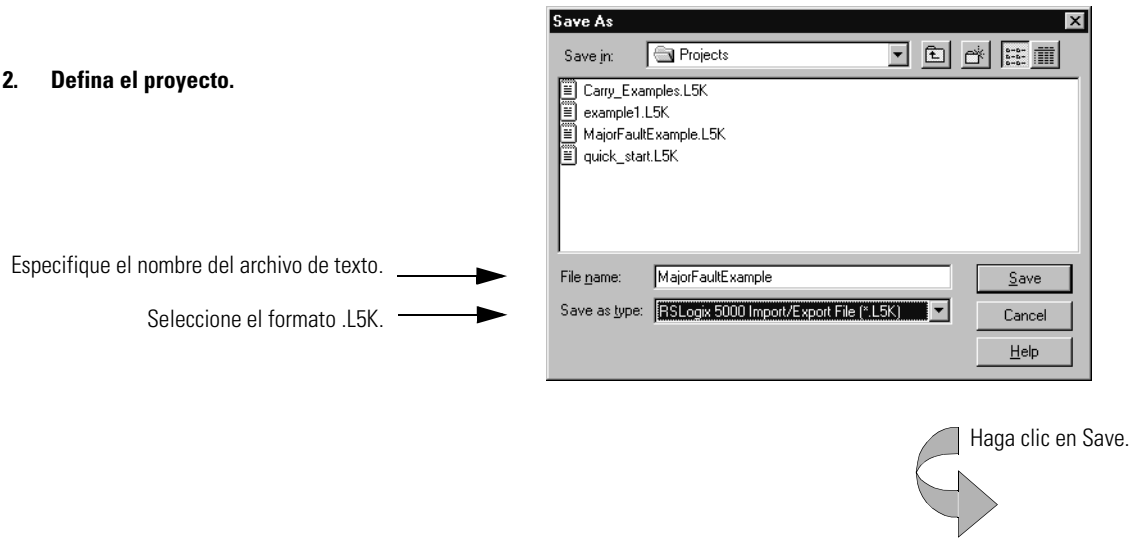
Se puede exportar el proyecto a un archivo de texto. Luego se puede usar cualquier editor de texto para modificar el proyecto.

Asegúrese de que el proyecto que desea exportar ya está abierto.

1. Seleccione File → Save As.



2. Defina el proyecto.

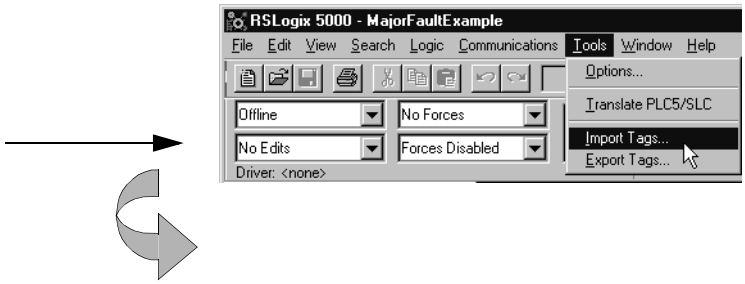


IMPORTANTE Las ediciones no guardadas se guardan automáticamente cuando usted acepta (mediante el botón OK) la operación de exportación.

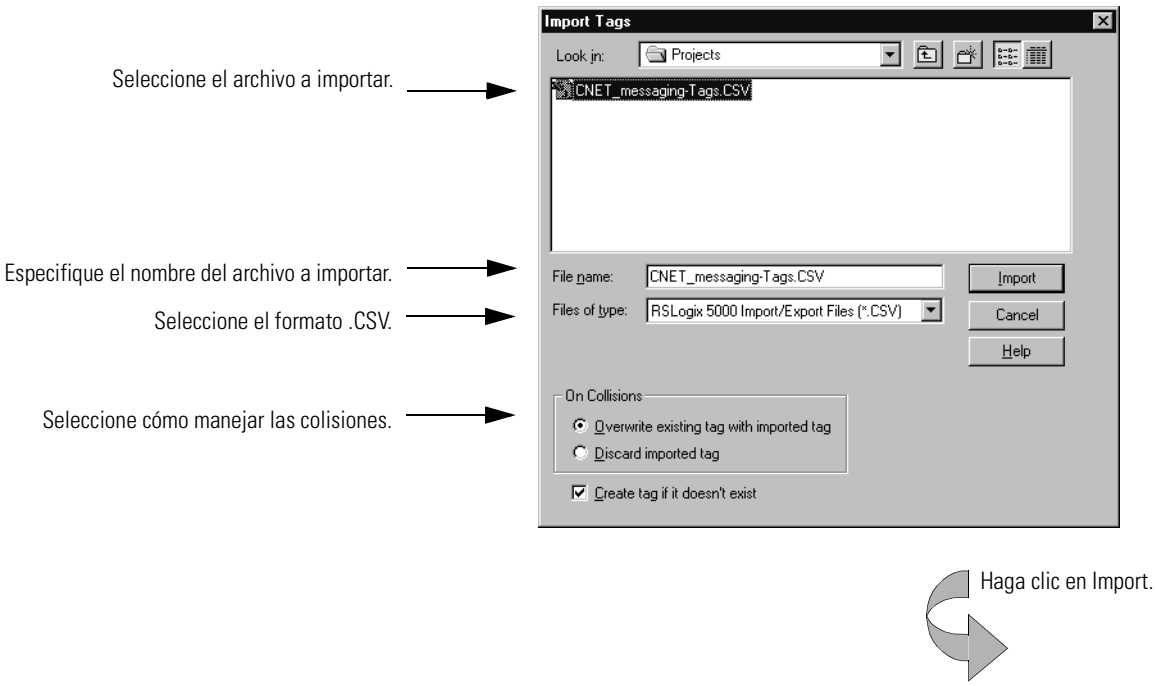
Importación de un archivo de texto de tags a un proyecto

Cuando usted está fuera de línea y tiene un proyecto abierto, puede importar tags desde un archivo de texto guardado (que tenga la extensión .CSV). Esto le permite usar un programa de base de datos (como Microsoft® Access®) para crear y editar los tags.

1. **Seleccione Tools → Import Tags.**



2. **Seleccione el archivo de texto.** El archivo de texto debe tener una extensión .CSV.



Al importar tags, existe la posibilidad de que los tags en el archivo de importación tengan el mismo nombre que los tags que ya están en el proyecto abierto. Esta condición es una colisión. Usted especifica cómo manejar una colisión al seleccionar el archivo de tags a importar:

Si desea:	Seleccione:
reemplazar el tag en el proyecto con el tag del archivo de importación	Overwrite (esta es la selección predeterminada)
mantener el tag que está en el proyecto y desechar el tag que está en el archivo de importación	Discard

También es posible tener tags en el archivo de importación que no existen en el proyecto abierto. Puede seleccionar si va a crear estos tags en el proyecto.

Si elimina los tags de un archivo de importación/exportación y luego importa el archivo, los tags no se eliminan del proyecto del controlador. Deberá usar el software de programación para eliminar los tags de la lista de tags.

Para obtener más información acerca de la estructura del archivo de importación/exportación parcial, vea:

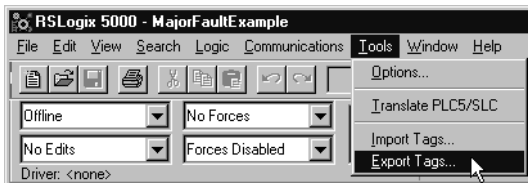
Para obtener información acerca de:	Vea el capítulo:
estructurar un archivo de importación/exportación parcial	5

Exportación de tags a un archivo de texto

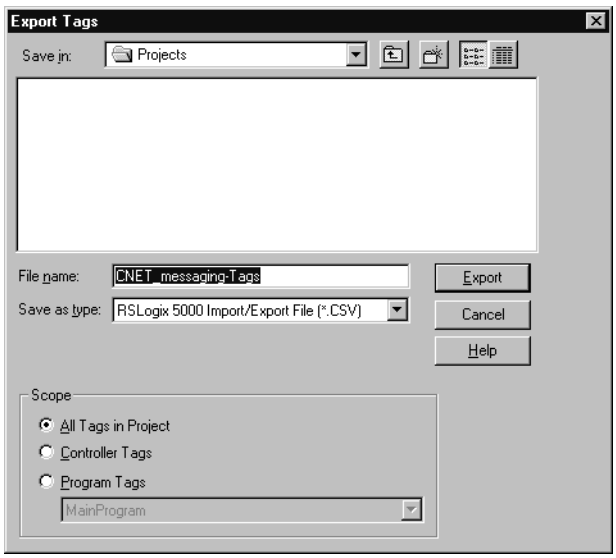
Cuando un proyecto está abierto, se pueden importar tags a un archivo de texto. Luego se puede usar un programa de base de datos (como Microsoft Access) para editar los tags.

Asegúrese de que el proyecto del cual desea exportar tags ya esté abierto.

1. Seleccione Tools → Export Tags.



2. Defina el proyecto.



Especifique el nombre del archivo de tags.

Seleccione el formato .CSV.

Seleccione el grupo de tags a exportar.

Haga clic en Export.

Seleccionar el grupo de tags a exportar

Al exportar tags, usted tiene las siguientes opciones referentes a cuáles tags en el proyecto desea exportar.

Grupo:	Esta opción de exportación:
Todos los tags en el proyecto	todos los tags (tags del controlador cubiertos y tags del programa cubiertos) del proyecto a un archivo de texto.
Tags del controlador	los tags del controlador cubiertos del proyecto a un archivo de texto.
Tags del programa	los tags del programa cubiertos del programa especificado. Use la flecha desplegable para ver en pantalla los programas disponibles en el proyecto actual.

Estructuración de un formato de archivo de importación/exportación (.L5K) completo

Introducción

Este capítulo explica la estructura general de un archivo de importación/exportación completo. La extensión de archivo para un archivo de importación/exportación completo es .L5K.

Para obtener información específica de cada componente de un archivo de importación/exportación, vea el capítulo “Creación de un archivo de importación/exportación”. Para obtener información sobre la introducción de la lógica, vea el capítulo “Introducción de la lógica”.

Convenciones

La utilidad de importación/exportación se basa en los formatos especificados en la norma IEC 1131-3. Los ejemplos siguen estas convenciones:

Convención:	Significado:
< >	los ítems mostrados en paréntesis angulares son necesarios
[ ]	los ítems mostrados en corchetes cuadrados son opcionales
user_value	los ítems en letra cursiva indican información proporcionada por el usuario
LITERAL	los ítems en mayúsculas indican una palabra clave o símbolo requerido que debe introducirse tal como se muestra
“[“	los ítems entre signos de dos comillas son caracteres requeridos

Los caracteres de espacio en blanco incluyen espacios, tabulaciones, retorno de carro, nueva línea y alimentación de impresora. Estos caracteres pueden estar en cualquier lugar de un archivo de importación/exportación, excepto en palabras claves o nombres. Si hubiera espacios en blanco fuera de descripciones, éstos se ignoran.

Comentarios internos de archivo

Se pueden introducir comentarios para documentar los archivos de importación. El proceso de importación ignora estos comentarios. Se pueden colocar comentarios en cualquier lugar de un archivo de importación/exportación, excepto en palabras clave, nombres y descripciones de componentes.

Se pueden introducir comentarios usando cualquiera de estos métodos:

- Empiece el comentario con dos caracteres de porcentaje (%%) y deténgase al final de la línea
- Empiece el comentario con "(*" y termine con los correspondientes "*)".

Colocación de información en un archivo de importación/exportación

El archivo de importación/exportación contiene diferentes componentes de información. Estos componentes son:

Componente:	Identifica:
CONTROLLER	nombre del archivo del proyecto
DATATYPE	estructuras de datos de E/S y definidas por el usuario
MODULE	módulos en el organizador del controlador
TAG	tags del controlador cubiertos
PROGRAM	archivos del programa
ROUTINE	archivos de rutinas
TASK	tareas del controlador
CONFIG	información de configuración

Todos los componentes en un archivo de importación siguen esta estructura:

```
Component_Type < component_name > [ Attributes ]
        [ body ]
END_Component_Type
```

Donde:

Ítem:	Identifica:
Component_Type	el componente (según lo definido en la tabla anterior)
component_name	una instancia específica del componente
Attributes	los atributos del componente también puede contener una descripción del componente separe cada atributo con una coma (,)
body	los subcomponentes (secundarios) de este componente
END_*Component_Type*	fin de la información del componente

Estilo de pantalla

Los tags y tipos de datos aceptan un atributo de base que le permite especificar cómo mostrar en pantalla la información numérica asociada. Las opciones de base son

- Binario (usa 2# como prefijo)
- Octal (usa 8# como prefijo)
- Decimal
- Hex (usa 16# como prefijo)
- Exponencial
- Valor con punto flotante (coma flotante)

Descripciones de componentes

Las descripciones de los componentes son opcionales. A diferencia de los comentarios internos, las descripciones son importadas. Coloque la descripción entre signos de dos comillas. Por ejemplo:

```
TASK Task1 (Description := “Hello World”, Rate := 10000,
Priority := 10 )
END_TASK
```

Para introducir caracteres de control en la descripción, preceda el carácter con un signo de dólar ($). La siguiente tabla muestra cómo introducir los caracteres de control aceptados en una descripción:

Para este carácter:	**Introduzca:**
$	$$
‘	$‘
“	$”
10 (salto de línea)	$L o $l
13,10 (retorno de carro, salto de línea)	$N o $n
12 (alimentación de impresora)	$P o $p
13 (retorno de carro)	$R o $r
9 (tabulación)	$T o $t
xxxx (código de carácter de 4 dígitos que representa un valor hexadecimal)	$xxxx

Definición de un controlador

El componente CONTROLLER es la estructura general de un proyecto que va a ser ejecutado en un controlador. Contiene la información de configuración y la lógica que se descarga a un controlador. Un componente CONTROLLER sigue la siguiente estructura:

```
IE_VER := 2.0
CONTROLLER < controller_name > [ Attributes ]
       [ < DATATYPE declaration > ]
       [ < MODULE declaration > ]
       [ < TAG declaration > ]
       [ < PROGRAM declaration > ]
       [ < TASK declaration > ]
       [ < CONFIG controller objects declaration > ]
END_CONTROLLER
```

Donde:

Ítem:	Identifica:
controller_name	el controlador del proyecto
Attributes	atributos del controlador también puede contener una descripción del controlador separe cada atributo con una coma (,)
DATATYPE	estructuras de datos de E/S y definidas por el usuario Vea la página 3-1.
MODULE	dispositivos en el organizador del controlador Vea la página 3-4.
TAG	tags del controlador cubiertos Vea la página 3-9.
PROGRAM	organización de rutinas Vea la página 3-13.
TASK	organización de programas Vea la página 3-15.
CONFIG	características de objetos del controlador (información de estado) Vea la página 3-18.

Especificar los atributos del CONTROLLER

Se pueden especificar los siguientes atributos de un CONTROLLER:

Atributo:	Descripción:
`Description`	Proporciona información acerca del controlador. Especifique: `Description := "text"`
`TimeSlice`	Porcentaje de tiempo CPU disponible (10-90) que se asigna a las comunicaciones. Especifique: `TimeSlice := value`
PowerLossProgram	Nombre del programa a ser ejecutado al momento de la reinicialización después de una interrupción de la alimentación eléctrica. Especifique: `PowerLossProgram := name`
MajorFaultProgram	Nombre del programa que se ejecutará cuando ocurra un fallo mayor. Especifique: `MajorFaultLossProgram := name`
CommDriver	Nombre del controlador de comunicaciones. Especifique: `CommDriver := name`
CommPath	Nombre de la ruta de acceso al controlador de comunicaciones. Especifique: `CommPath := name`

Pautas del CONTROLLER

Tenga en cuenta estas pautas al definir un tipo de datos:

- Todas las declaraciones deben ordenarse explícitamente tal como se muestra en la sintaxis anterior.
- Pueden haber 32 tareas como máximo.
- Sólo puede haber una tarea continua.
- Los programas sólo pueden programarse debajo de una tarea
- Los programas programados deben definirse, es decir, deben existir

Ejemplo de CONTROLLER

```
CONTROLLER TestImportExport (Description := “Example",
TimeSlice := 11, MajorFaultProgram := Prg2)
END_CONTROLLER
```

Notas:

Creación de un archivo de importación/ exportación completo

Introducción

Este capítulo explica cómo introducir información del proyecto y de la configuración en un archivo de importación/exportación.

Para obtener información sobre cómo introducir la lógica, vea el siguiente capítulo.

Definición de un tipo de datos

Un componente DATATYPE sigue la siguiente estructura:

```
DATATYPE < DataType_name > [ Attributes ]
       [ member_definition ]
END_DATATYPE
```

Donde:

Ítem:	Identifica:
`DataType_name`	la estructura de datos
`Attributes`	los atributos de la estructura de datos también puede contener una descripción del componente separe cada atributo con una coma (,)
`member_definition`	cada miembro de la estructura de datos

Especificar atributos del DATATYPE

Se pueden especificar los siguientes atributos de un DATATYPE:

Atributo:	Descripción:
`Description`	Proporcione información acerca del tipo de datos. Especifique: `Description := "text"`
`ProductDefined`	Defina la estructura como una estructura de E/S. No usar para estructuras definidas por el usuario. Especifique: `ProductDefined := 1`
`Radix`	Especifique el formato como decimal, hexadecimal, octal, binario, exponencial o punto flotante (coma flotante) Especifique: `Radix := value`
`Hidden`	Haga del miembro un miembro oculto de la estructura. Especifique: `Hidden := 1`

Especificar un miembro de DATATYPE

Hay dos tipos de miembros de tipo de datos. Un miembro bit es un miembro en el cual sólo se va a tener acceso a un solo bit de información. Un miembro no bit es un miembro que se define como otro tipo de datos (tal como SINT, INT, DINT, COUNTER, etc.).

La definición de un miembro no bit sigue esta estructura:

```
< TypeName > < MemberName > < Attributes >;
```

Todos los tipos de datos son asignados en límites de 8 bits. Un solo bit de almacenamiento no está permitido, por lo tanto un miembro no puede ser un tipo de datos BOOL. Para acceder a un solo bit, use la declaración BIT. BIT permite el acceso a un solo bit dentro de un miembro principal (un miembro no bit).

Un miembro bit usa la siguiente sintaxis:

```
BIT < BitName > < HostMemberName > : < BitPosition > < Attributes >
```

Por ejemplo, para tener acceso a dos bits definidos como MyBit0 y MyBit1, defina a un miembro principal como MySint y haga referencia a los dos bits dentro de ese miembro principal. Generalmente, el miembro principal es un miembro oculto porque sólo las referencias de bit son visibles. La sintaxis para este ejemplo es:

```
SINT MySint (Hidden := 1)
BIT MyBit0 MySint : 0
BIT MyBit1 MySint : 1
```

IMPORTANTE **Debe** haber un espacio entre el nombre del miembro principal y el signo de dos puntos y entre el signo de dos puntos y la posición de bit porque los nombres de tipo pueden contener un signo de dos puntos (por ejemplo, las estructuras de E/S) y sin el espacio no sabríamos dónde termina el nombre del tipo.

Los miembros bit no pueden definirse antes que sus miembros principales. Observe que BitPosition cero es el bit menos significativo.

Especificar atributos de miembro de DATATYPE

Se pueden especificar los siguientes atributos de miembro de un DATATYPE:

Atributo:	Descripción:
`Description`	Proporcione información acerca del miembro del tipo de datos. Especifique: `Description := "text"`
`Radix`	Especifique decimal, hexadecimal, octal, binario, exponencial o punto flotante (coma flotante) Especifique: `Radix := value`
`Hidden`	Haga del miembro un miembro oculto de la estructura. Especifique: `Hidden := 1`

Pautas de DATATYPE

Tenga en cuenta estas pautas al definir un tipo de datos:

- Los tipos de datos deben definirse primero dentro del cuerpo del controlador.
- Los tipos de datos pueden definirse fuera de orden. Por ejemplo, si el Type1 depende de Type2, Type2 puede definirse primero.
- Los tipos de datos pueden ser no verificados. Por ejemplo, si Type1 depende de Type2 y Type2 no se ha definido, entonces Type1 será accesible como un tipo no verificado. El Type2 será un type sin tipo. Se podrán crear tags de Type1 pero no de Type2.
- Los miembros de tipos de datos pueden ser arreglos, pero sólo se permite una dimensión.
- Los siguientes tipos de datos no pueden usarse en un tipo de datos definido por el usuario: AXIS, MOTION_GROUP y MESSAGE.

Ejemplo de DATATYPE

```
DATATYPE MyStruct
       DINT x (Hidden := 1);
       DINT y (Radix := Hex);
       TIMER y[3];
       BIT aBit x : 3;
END_DATATYPE
```

Definición de un módulo

Un componente MODULE sigue la siguiente estructura:

```
MODULE < device_name > < Attributes >
       ConfigData := < initial_value >;
< connection_list >
END_MODULE
```

Donde:

Ítem:	Identifica:
`device_name`	el módulo
`Attributes`	atributos del módulo también puede contener una descripción del módulo separe cada atributo con una coma (,)
`ConfigData`	características de operación del módulo
`connection_list`	características de conexión del módulo vea la página 3-6

Especificar atributos del MODULE

Se pueden especificar los siguientes atributos de un MODULE:

Atributo:	Descripción:
`Description`	Proporcione información acerca del módulo. Especifique: `Description := "text"`
`Parent`	Si el módulo es secundario a otro módulo, especifique el nombre del módulo primario. El módulo primario debe definirse antes que cualquier módulo secundario. Especifique: `Parent := name`
`CatalogNumber`	Especifique el número de catálogo del módulo. Especifique: `CatalogNumber := number`
`Major`	Especifique el número de revisión mayor (1 – 127) del módulo. Especifique: `Major := number`

Atributo:	**Descripción:**
Minor	Especifique el número de revisión menor (1 – 255) del módulo. Especifique: `Minor := number`
PortLabel	Si éste es un módulo secundario, especifique el puerto usado para llegar a este módulo desde su módulo primario. La etiqueta de puerto es RxBACKPLANE para módulos en un chasis o una cadena de texto para módulos en una red. Especifique: `PortLabel := label`
ChassisSize	Especifique el número de ranuras en el chasis (1 – 32) Especifique: `ChassisSize := number`
Slot	Especifique el número de ranura (1 – 31) donde está el módulo en el chasis. Especifique: `Slot := number`
NodeAddress	Especifique la dirección de nodo ControlNet (1 – 99) o la dirección de rack de E/S remotas (0 – 63) del módulo. Especifique: `NodeAddress := number`
Group	Si el módulo es un módulo de E/S remotas, especifica el grupo inicial (0 – 7). Para un módulo de transferencia en bloques, éste es el número de grupo de módulos bajo el adaptador de E/S remotas. Especifique: `Group := number`
CommMethod	Especifique el método de hacer conexión al módulo. Especifique: `CommMethod := number`
ConfigMethod	Especifique el método de configuración del módulo. Especifique: `ConfigMethod := number`
Mode	Seleccione un modo específico estableciendo el bit apropiado. **Establezca: Para:** 0 fallo en el módulo causa un fallo mayor en el controlador 2 inhibe el módulo Especifique: `Mode := number`
KeyMask	Especifique si se hará conexión al módulo exacto que coincide con la información de codificación electrónica (proveedor, código del producto, tipo del producto, revisión mayor, revisión menor). La falta de codificación hará conexión a cualquier módulo. Especifique: `KeyMask := hex_string`
CompatibleModule	Especifique si se hará conexión a un módulo compatible en base a la revisión menor (*value* = 1) o a un dispositivo exactamente igual al módulo (*value* = 0). Especifique: `CompatibleModule := value`
ChABaud	Para un módulo 1756-DHRIO, especifique la velocidad en baudios para el canal A. Introduzca 57.5, 115.2 ó 230.4. Especifique: `ChABaud := baud`
ChBBaud	Para un módulo 1756-DHRIO, especifique la velocidad en baudios para el canal B. Introduzca 57.5, 115.2 ó 230.4. Especifique: `ChBBaud := baud`

Especificación de una lista de conexión de un MODULE

Se pueden especificar los siguientes atributos de una lista de conexión:

```
CONNECTION < connection_name > < Attributes >
       InputData := < value_list >;
       InputForceData := < value_list >;
       OutputData := < value_list >;
       OutputForceData := < value_list >;
END_CONNECTION
```

Donde:

Ítem:	Identifica:
`connection_name`	la conexión
InputData	datos del canal de entrada
InputForceData	información de forzados para el canal de entrada
OutputData	datos del canal de salida
OutputForceData	información de forzados para el canal de salida
`Attributes`	atributos de la conexión también puede contener una descripción del módulo separe cada atributo con una coma (,)

Para obtener detalles sobre los datos en la lista de conexión, vea el manual del usuario del módulo de E/S. Los datos de la lista de conexión dependen del módulo de E/S y de la configuración para dicho módulo.

Los forzados aparecen como arreglos de bytes bajo los atributos InputForceData y OutputForceData de la lista de conexión. No modifique los forzados en el archivo de importación/exportación. Use el software de programación para introducir y habilitar los forzados.

Especificar los atributos de la lista de conexión del MODULE

Se pueden especificar los siguientes atributos de una lista de conexión para un MODULE:

Atributo:	Descripción:
`Description`	Proporcione información acerca de la lista de conexión. Especifique: `Description := "text"`
`Rate`	Especifique el régimen del intervalo de paquete solicitado (RPI) en microsegundos. Especifique: `Rate := microseconds`
EventID	Existe para uso futuro. Por ahora: Especifique: `EventID := NA`

Pautas del MODULE

Tenga en cuenta estas pautas al definir un módulo:

- Los atributos deben ordenarse explícitamente tal como se muestra en la tabla anterior.
- Un módulo primario debe definirse antes que las definiciones de sus módulos secundarios.

Ejemplo de MODULE

```
MODULE Local (Parent := Local,
       CatalogNumber := 1756-L1,
       Major := 1,
       PortLabel := RxBACKPLANE,
       ChassisSize := 10,
       Slot := 3,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)

END_MODULE
```

```
MODULE DHRIO_Module (Parent := Local,
       CatalogNumber := 1756-DHRIO,
       Major := 2,
       PortLabel := RxBACKPLANE,
       Slot := 8,
       CommMethod := Standard,
       ConfigMethod := ChannelA RIO ChannelB DH,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111,
       ChABaud := 115.2,
       ChBBaud := 57.6)

       CONNECTION Standard (Rate := 500000,
              EventID := NA)
       END_CONNECTION
END_MODULE
MODULE Diagnostic_Module_1 (Parent := Local, CatalogNumber :=
1756-OB16D,
       Major := 1,
       PortLabel := RxBACKPLANE,
       Slot := 5,
       CommMethod := Full Diagnostics - Output Data,
       ConfigMethod := Diagnostic,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)
       ConfigData :=
[44,19,1,0,0,0,0,0,0,0,65535,65535,65535,0];

       CONNECTION Diagnostic (Rate := 5000,
              EventID := NA)
       END_CONNECTION

END_MODULE
```

Definición de un tag

Se pueden definir tags del controlador cubiertos y tags del programa cubiertos. Los tags del controlador cubiertos se definen en el componente TAG dentro del componente CONTROLLER; los tags del programa cubiertos se definen en un componente TAG dentro del componente PROGRAM dentro de un componente CONTROLLER. Un componente TAG sigue la siguiente estructura:

```
TAG
      [ tag_declaration ]
END_TAG
```

Dentro de una lista de tags, los tags de movimiento y mensajes deben seguir a todos los tags que no son de movimiento y los tags de eje deben seguir a los tags de grupo de movimiento.

IMPORTANTE Para obtener información detallada sobre los tags atómicos y de estructura y sus atributos y rangos, vea el *Manual del usuario del controlador Logix5550*, publicación 1756-6.5.12ES.

Definir una declaración de TAG para un tag que no es alias

Una declaración de tag para un tag que no es alias sigue esta estructura:

```
< tag_name > : < type:array > < Attributes > < initial_value >;
```

Donde:

Ítem:	Identifica:
`tag_name`	nombre del tag
type	tipo de tag tipos de tag aceptados: BOOL, SINT, INT, DINT, REAL tipos predefinidos: AXIS, CONTROL, COUNTER, MESSAGE, MOTION_GROUP, MOTION_INSTRUCTION, PID, TIMER
array	límites dimensionales para tags de arreglo vea la página 3-10
`Attributes`	atributos del tag también puede contener una descripción del tag separe cada atributo con una coma (,) vea la página 3-11
`initial_value`	valor inicial del tag vea la página 3-12

No puede **haber** ningún espacio en blanco entre el tipo y la definición del arreglo. **Debe** haber un espacio en blanco entre el nombre del tag y el signo de dos puntos y otro espacio entre el mismo signo de dos puntos y el nombre del tipo.

Esto es porque los nombres de tipo pueden tener un signo de dos puntos y sin el espacio sería imposible detectar dónde empieza el nombre del tipo.

Definir una declaración de TAG para un tag de alias

Una declaración de tag para un tag que no es alias sigue esta estructura:

```
< tag_name > OF < alias > < Attributes >;
```

Donde:

Ítem:	Identifica:
`tag_name`	nombre del tag de alias
alias	nombre del tag base al cual se refiere el tag de alias Especifique: *`alias<specifier>`* Donde *specifier* es: un bit (*.bitnumber*), elemento de arreglo ([*element*]), o miembro de estructura (*.membername*) del tag.
`Attributes`	atributos del tag también puede contener una descripción del tag separe cada atributo con una coma (,)

Definir de una especificación de arreglo dentro de una declaración de TAG

Una especificación de arreglo sigue esta estructura:

```
"[" < element > [ , < element > [ , < element > ] ] "]"
```

Donde:

Ítem:	Identifica:
`element`	el número de elementos dentro de la dimensión del arreglo por ejemplo: [5, 10, 2]

Especificación de atributos de TAG

Se pueden especificar los siguientes atributos de un TAG:

Atributo:	**Descripción:**
`Description`	Proporcione información acerca del tag. Especifique: `Description := "text"`
`Radix`	Especifique el formato como decimal, hexadecimal, octal, binario, exponencial o punto flotante (coma flotante) Especifique: `Radix := value`
`ProduceCount`	Especifique el número de consumidores permitido (0 – 255). Especifique: `ProduceCount := value`
`Producer`	Si el controlador produce este tag, especifique el nombre del controlador remoto que consume este tag. También debe especificar un atributo RemoteTag y RPI. Especifique: `Producer:= name`
RemoteTag	Si el controlador produce este tag para un controlador que acepta nombres de tag, especifique el nombre del tag en el controlador remoto. También debe especificar un atributo Producer y RPI. Especifique: `RemoteTag := name`
RPI	Si el controlador produce este tag, especifique el valor RPI en milisegundos. También debe especificar un atributo Producer y RemoteTag. Especifique: `RPI := milliseconds`
RemoteFile	Si el controlador produce este tag para un controlador PLC-5, especifique el número del archivo PLC-5 (1 – 999) en el controlador PLC-5. También debe especificar un atributo Producer u uno RPI. Especifique: `RemoteFile := number`
PLCMappingFile	Si este tag se asigna a un controlador PLC, especifique el número de archivo (0 – 65535). Especifique: `PLCMappingFile := number`
PLC2Mapping	Si este tag se asigna a un archivo PLC-2, establezca este atributo en 1. Especifique: `PLC2Mapping := 1`
Comment	Proporcione información acerca de un componente de tag. Especifique: `Comment<specifier> := "text"` Donde *specifier* es: *.bitnumber* para un bit en el tag [*element*] para un elemento de arreglo del tag *.membername* para un miembro de estructura del tag

IMPORTANTE Si se proporciona información de consumo en un tag de alias, el tag de alias se convierte en un tag base antes de que pueda consumir datos.

Definición de valores iniciales de TAG

El formato `initial_value` sigue la sintaxis de inicialización del lenguaje C, excepto que se usan corchetes cuadrados en lugar de llaves. La siguiente tabla muestra algunos ejemplos de introducción de valores iniciales.

Si el tag es:	Introduzca:
valor simple, atómico	[*Value*]
estructura con tres miembros	[*Value1*, *Value2*, *Value3*]
estructura con una estructura anidada	[*Value1*, [*Value2*, *Value3*], *Value4*]
estructura con un arreglo anidado	[*Value1*, [*ArrayValue1*, *ArrayValue2*], *Value3*]

Definir un comentario para un componente de TAG

El atributo comentario de una declaración de tag le permite proporcionar información acerca de un componente del tag, tal como un bit específico, elemento de arreglo o miembro de estructura. Por ejemplo:

Para añadir un comentario a este operando:	Introduzca:
bit 3 de un tag	COMMENT.3 := "*description*"
elemento 8 de un tag de arreglo	COMMENT[8] := "*description*"
valor preseleccionado de un tag	COMMENT.PRE := "*description*"

Pautas de un TAG

Tenga en cuenta estas pautas al definir un tag:

- Los tags deben definirse después de los dispositivos (si no hay dispositivos, entonces después de los tipos de datos) dentro del cuerpo del controlador.
- Los tags de base y alias pueden definirse fuera de orden dentro de un bloque de tags.
- No se puede definir una 2ª dimensión sin una 1ª dimensión ni una 3ª dimensión sin una 2ª dimensión.
- Los valores iniciales deben cumplir con las especificaciones de tipo de tag y dimensiones.
- No puede haber espacios en blanco dentro de valores iniciales, ni dentro del especificador de tipo/dimensión.

Ejemplos de TAG

```
TAG
a1 OF t1.3 (Description := “Tag Alias Description”);
t3 : TIMER[5] (Description := “Array”
       Comment.EN := “monitor this bit”) := [[1,2,3],[2,2,2]];
t15 : REAL (Radix := Hex) := 3.000000e+000;
t10 : MESSAGE
       (LocalTag := a,
       RemoteElement := kk,
       ProduceCount := 0,
       MessageType := CIP Data Table Write,
       RequestedLength := 1,
       ConnectionPath := "1,2");
END_TAG
```

Definición de un Programa

Un componente PROGRAM sigue la siguiente estructura:

```
PROGRAM < program_name > < Attributes >
       < TAG declaration >
       < ROUTINE declaration >
END_PROGRAM
```

Donde:

Ítem:	Identifica:
program_name	el programa
Attributes	los atributos del programa (tales como MAIN o FAULT) también puede contener una descripción del programa separe cada atributo con una coma (,)
TAG	los tags del programa cubiertos sigue el mismo formato que los tags del controlador cubiertos vea la página 3-9
ROUTINE	las rutinas adicionales para este programa vea la página 3-15

Especificar atributos del PROGRAM

Se pueden especificar los siguientes atributos de una PROGRAM:

Atributo:	Descripción:
`Description`	Proporcione información acerca del programa. Especifique: `Description := "text"`
Main	Nombre de la rutina principal del programa Especifique: `Main := name`
Fault	Nombre de la rutina de fallo del programa, si la hubiera Especifique: `Fault := name`

Pautas del PROGRAM

Tenga en cuenta estas pautas al definir un programa:

- Los atributos MAIN y FAULT se pueden definir en cualquier orden.
- El bloque de declaración TAG debe ocurrir antes que el bloque de rutina.
- Todos los bloques de declaración de recolección de tags que ocurren en un bloque de definición de programa se importan como tags de un programa dado y sólo pueden ser vistos por las rutinas bajo dicho programa. Por el contrario, los tags del controlador pueden ser vistos por rutinas en cualquier programa.

Ejemplo de PROGRAM

```
PROGRAM Prg1 (Main := RoutineB, Description := "I $'am$'" "
$0034 a $"program$"")
       TAG
       st11 : DINT (RADIX := Decimal, ProduceCount := 0) := 2;
       st12 : BOOL (RADIX := Binary, ProduceCount := 0) :=
2#00000000;
       END_TAG

       ROUTINE RoutineA
       JSR(_2_LADDER, 0);
       END_ROUTINE

       ROUTINE RoutineB
              RC: "$L ** ;MORE $";STUFF" do not include "more";
              xic(st11) ote(st12);
       END_ROUTINE
END_PROGRAM
```

Definición de una rutina

Un componente ROUTINE sigue la siguiente estructura:

```
ROUTINE < routine_name > < Attributes >
       < rungs >
END_ROUTINE
```

Donde:

Ítem:	Identifica:
`routine_name`	la rutina
`Attributes`	los atributos de la rutina también puede contener una descripción de la rutina separe cada atributo con una coma (,)
`rungs`	la lógica de escalera para obtener más información sobre cómo introducir la lógica, vea el capítulo 3

Para obtener información sobre la sintaxis de la introducción de renglones e instrucciones, vea el siguiente capítulo.

Especificar atributos de la ROUTINE

Se pueden especificar los siguientes atributos de una ROUTINE:

Atributo:	Descripción:
`Description`	Proporcione información acerca de la rutina. Especifique: `Description := "text"`

Ejemplo de ROUTINE

```
ROUTINE RoutineB
RC: "This is a description for the 1st rung";
       xic(st11) ote(st12);
       xic(st11) ote(st12);
RC: "This is a description for the 2nd rung";
       xic(st11) ote(st12);

END_ROUTINE
```

Definición de una tarea

Un componente TASK sigue la siguiente estructura:

```
TASK < task_name > < Attributes >
       < program_name > ;
END_TASK
```

Donde:

Ítem:	Identifica:
`task_name`	la tarea
`Attributes`	los atributos de la tarea también puede contener una descripción de la tarea separe cada atributo con una coma (,)
`program_name`	cada programa dentro de la tarea

Especificar atributos de la TASK

Se pueden especificar los siguientes atributos de una TASK:

Atributo:	**Descripción:**
`Description`	Proporcione información acerca de la tarea. Especifique: `Description := "text"`
Type	Especifique el tipo de tarea (PERIODIC o CONTINUOUS). Sólo puede haber una tarea continua. Especifique: `Type := type`
Priority	Especifique la prioridad de una tarea periódica (1 – 15) Especifique: `Priority:=number`
Rate	Si la tarea es una tarea periódica, especifique la frecuencia de ejecución de la tarea (1000 – 2,000,000,000 us). Especifique: `Rate:=number`
Watchdog	Introduzca el tiempo de espera del temporizador de control (watchdog) para la tarea (1000 – 2,000,000,000 us). Especifique: `Watchdog :=number`

Pautas de la TASK

Tenga en cuenta estas pautas al definir una tarea:

- Las tareas se deben definir después de los programas y antes de los objetos del controlador.
- Pueden haber 32 tareas como máximo.
- Sólo puede haber una tarea continua.
- Un programa sólo puede programarse debajo de una tarea.
- Los programas programados deben definirse, es decir, deben existir.

Ejemplo de TASK

```
TASK joe (TYPE := PERIODIC, PRIORITY := 8, RATE := 10000)
       sue;
       betty;
END_TASK
```

Los atributos de la tarea (Type, Priority, Rate y Watchdog) se pueden definir en cualquier orden. La lista de programas priorizados para una tarea se listan en el bloque de declaraciones de la tarea, tal como se muestra anteriormente. Los programas se ejecutan en el orden en que se especificaron.

Definición de un objeto del controlador

Un componente CONFIG define objetos del controlador y sigue esta estructura.

```
CONFIG < object_name > < Attributes >
END_CONFIG
```

Donde:

Ítem:	Identifica:
`object_name`	el objeto del controlador Para obtener información detallada sobre los objetos del controlador disponibles y sus atributos, vea el *Manual de referencia del conjunto de instrucciones del controlador Logix5550,* publicación 1756-6.4.1ES. Este manual describe cada objeto y su rango válido de valores.
`Attributes`	los atributos del objeto del controlador también puede contener una descripción de la tarea separe cada atributo con una coma (,)

Los objetos del controlador son opcionales. Sólo puede existir uno de cada objeto de controlador que se decida definir. Los objetos del controlador aparecen al final del archivo de importación/exportación.

Ejemplos de CONFIG

Los dos ejemplos siguientes muestran un objeto de controlador DF1 y un objeto de controlador SerialPort.

```
CONFIG DF1
       DuplicateDetection := -1,
       ErrorDetection := BCC Error,
       EmbeddedResponseEnable := -1,
       DF1Mode := Pt to Pt,
       ACKTimeout := 50,
       NAKReceiveValue := 3,
       DF1ENQs := 3,
       DF1Retries := 3,
       StationAddress := 0,
       ReplyMessageWait := 50,
       PollingMode := 0,
       MasterMessageTransmit := 0,
       NormalPollNodeFile := NA,
       NormalPollGroupSize := 0,
       PriorityPollNodeFile := NA,
       ActiveStationFile := NA)
END_CONFIG


CONFIG SerialPort
       (BaudRate := 19200,
       Parity := No Parity,
       DataBits := 8 Bits of Data,
       StopBits := 1 Stop Bit,
       ComDriverId := DF1,
       RTSOffDelay := 0,
       RTSSendDelay := 0,
       ControlLine := No Handshake,
       RemoteModeChangeFlag := 0,
       ModeChangeAttentionChar := 27,
       SystemModeCharacter := 83,
       UserModeCharacter := 85)
END_CONFIG
```

Notas:

Introducción de la lógica de escalera

Introducción

Este capítulo explica cómo introducir la lógica de escalera en un archivo de importación/exportación completo.

Para obtener información acerca de:	Vea la página:
Introducir la lógica de renglón	4-1
Introducir comentarios	4-3
Introducir instrucciones en texto neutro	4-3

Introducción de lógica de renglón

La lógica de renglón se introduce dentro de un componente ROUTINE en un archivo de importación/exportación. Cada renglón sigue la siguiente estructura:

```
< RungType > : < RungNeutralText > ;
```

Donde:

Ítem:	Identifica:
RungType	el renglón
RungNeutralText	la lógica

Los siguientes tipos de renglón están disponibles:

Tipo de renglón:	Descripción:
N	normal
I	insertar
D	eliminar
IR	insertar con un reemplazo
rR	reemplazar IR pendiente
R	reemplazar
rI	reemplazar I pendiente
rN	reemplazar N pendiente
e	renglón de insertar pendiente
er	renglón de reemplazar pendiente

Pautas de los renglones

- Los renglones se especifican usando lenguaje neutro. Consulte el resto de este capítulo para ver el formato de texto neutro para las instrucciones aceptadas.
- Cada renglón termina con un punto y coma (;).

Ejemplo de renglón

```
N: OTE(TagX) OTE(TagY);
```

Introducción de bifurcaciones

Se puede introducir una sola bifurcación o bifurcaciones simultáneas en un renglón. Una bifurcación sigue la siguiente estructura:

[,*BranchNeutralText*]

Donde:

Ítem:	Identifica:
[ ]	la bifurcación
,	el comienzo de cada bifurcación dentro de la bifurcación, para tener en cuenta bifurcaciones simultáneas
space	el final de cada bifurcación dentro de la bifurcación, para tener en cuenta bifurcaciones simultáneas
BranchNeutralText	la lógica

Ejemplo con una sola bifurcación

```
N: XIC(conveyor_a)[,XIC(input_1) XIO(input_2) ]OTE(light_1);
```

Ejemplo con dos bifurcaciones simultáneas

```
N: XIC(conveyor_b)[,XIC(input_1) XIO(input_2) ,XIC(input_a) XIO(input_b) ]OTE(light_2);
```

Introducción de comentarios de renglón

Los comentarios de renglones son similares a los de los componentes, excepto que la sintaxis es un poco diferente. La sintaxis del comentario de renglón es:

```
RC: “comment” “more” “etc”;
```

Un comentario de renglón debe ser seguido por un renglón.

Introducción de texto neutro para instrucciones

La siguiente tabla lista cada instrucción y su formato de texto neutro.

IMPORTANTE Para obtener información detallada sobre los parámetros y las instrucciones y sus valores aceptados, vea el *Manual de referencia del conjunto de instrucciones del controlador Logix5550*, publicación 1756-6.4.1ES.

Instrucción:	Formato de texto neutro:
ABS	`ABS(source,destination);`
ACS	`ACS(source,destination);`
ADD	`ADD(source_A,source_B,destination);`
AFI	`AFI();`
AND	`AND(source_A,source_B,destination);`
ASN	`ASN(source,destination);`
ATN	`ATN(source,destination);`
AVE	`AVE(array,dim_to_vary,destination,control,length,position);`
BRK	`BRK();`
BSL	`BSL(array,control,source_bit,length);`
BSR	`BSR(array,control,source_bit,length);`
BTD	`BTD(source,source_bit,destination,destination_bit,length);`
CLR	`CLR(destination);`
CMP	`CMP(expression);`
COP	`COP(source,destination,length);`
COS	`COS(source,destination);`
CPT	`CPT(destination,expression);`
CTD	`CTD(counter,preset,accum);`
CTU	`CTU(counter,preset,accum);`
DDT	`DDT(source,reference,result,cmp_control,length,position,result_control,length,position);`
DEG	`DEG(source,destination);`
DIV	`DIV(source_A,source_B,destination);`

Instrucción:	Formato de texto neutro:
DTR	DTR(*source,mask,reference*);
EQU	EQU(*source_A,source_B*);
FAL	FAL(*control,length,position,mode,destination,expression*);
FBC	FBC(*source,reference,result,cmp_control,length,position,result_control, length,position*);
FFL	FFL(*source,FIFO,control,length,position*);
FFU	FFU(*FIFO,destination,control,length,position*);
FLL	FLL(*source,destination,length*);
FOR	FOR(*routine_name,index,initial_value,terminal_value,step_size*);
FRD	FRD(*source,destination*);
FSC	FSC(*control,length,position,mode,expression*);
GEQ	GEQ(*source_A,source_B*);
GRT	GRT(*source_A,source_B*);
GSV	GSV(*object_class,object_name,attribute_name,destination*);
JMP	JMP(*label_name*);
JSR	JSR(*routine_name,input_1,...input_n,return_1,..return_n*);
LBL	LBL(*label_name*);
LEQ	LEQ(*source_A,source_B*);
LES	LES(*source_A,source_B*);
LFL	LFL(*source,LIFO,control,length,position*);
LFU	LFU(*LIFO,destination,control,length,position*);
LIM	LIM(*low_limit,test,high_limit*);
LN	LN(*source,destination*);
LOG	LOG(*source,destination*);
MAAT	MAAT(*axis,motion_control*);
MAFR	MAFR(*axis,motion_control*);
MAG	MAG(slave*_axis,master_axis,motion_control,direction,ratio,slave_counts,mas ter_counts,master_reference,ratio_format,clutch,accel_rate,units*);
MAH	MAH(*axis,motion_control*);
MAHD	MAHD(*axis,motion_control,test,direction*);
MAJ	MAJ(*axis,motion_control,direction,speed,units,accel_rate,units, decel_rate,units,profile,merge,merge_speed*);
MAM	MAM(*axis,motion_control,move_type,position,speed,units,accel_rate units,decel_rate,units,profile,merge,merge_speed*);
MAPC	MAPC(*slave_axis*,*master_axis*,*motion_control*,*direction*,*cam_profile*, *slave_scaling*,*master_scaling*,*execution_mode*,*execution_schedule*, *master_lock_position*,*cam_lock_position*,*master_reference*, *master_direction*);
MAR	MAR(*axis,motion_control,trigger,registration,minimum,maximum*);

Instrucción:	Formato de texto neutro:
MAS	MAS(*axis,motion_control,stop_type,change_decel,rate_units*);
MASD	MASD(*axis,motion_control*);
MASR	MASR(*axis,motion_control*);
MATC	MATC(*axis*,*motion_control*,*direction*,*cam_profile*,*distance_scaling*, *time_scaling*,*execution_mode*,*execution_schedule*);
MAW	MAW(*axis,motion_control,trigger,position*);
MCCP	MCCP(*motion_control*,*cam*,*length*,*start_slope*,*end_slope*,*cam_profile*);
MCD	MCD(a*xis,motion_control,motion_type,change_speed,speed,change_accel, accel_rate,change_decel,decel_rate,speed_units,accel_units, decel_units*);
MCR	MCR();
MDF	MDF(*axis,motion_control*);
MDO	MDO(*axis,motion_control,drive_output,drive_units*);
MDR	MDR(*axis,motion_control*);
MDW	MDW(*axis,motion_control*);
MEQ	MEQ(*source,mask,compare*);
MGPS	MGPS(*group,motion_control*);
MGS	MGS(*group,motion_control,inhibit*);
MGSD	MGSD(*group,motion_control*);
MGSP	MGSP(*group,motion_control*);
MGSR	MGSR(*group,motion_control*);
MOD	MOV(*source_A,source_B,destination*);
MOV	MOV(*source,destination*);
MRAT	MRAT(*axis,motion_control*);
MRHD	MRHD(*axis,motion_control,test*);
MRP	MRP(a*xis,motion_control,type,position_select,position*);
MSF	MSF(*axis,motion_control*);
MSG	MSG(*message_control*);
MSO	MSO(*axis,motion_control*);
MUL	MUL(*source_A,source_B,destination*);
MVM	MVM(*source,mask,destination*);
NEG	NEG(*source,destination*);
NEQ	NEQ(*source_A,source_B*);
NOP	NOP();
NOT	NOT(*source,destination*);
ONS	ONS(*storage_bit*);
OR	OR(*source_A,source_B,destination*);
OSF	OSF(*storage_bit,output_bit*);

Instrucción:	Formato de texto neutro:
OSR	OSR(*storage_bit,output_bit*);
OTE	OTE(*data_bit*);
OTL	OTL(*data_bit*);
OTU	OTU(*data_bit*);
PID	PID(*pv,pv_type,tieback,cv,cv_type,master,inhold_bit,inhold_value*);
RAD	RAD(*source,destination*);
RES	RES(*structure*);
RET	RET(*return_1,...return_n*);
RTO	RTO(*timer,preset,accum*);
SBR	SBR(*routine_name,input_1,...input_n*);
SIN	SIN(*source,destination*);
SSV	SSV(*object_class,object_name,attribute_name,destination*);
SQI	SQI(*array,mask,source,control,length,position*);
SQL	SQL(*array,source,control,length,position*);
SQO	SQO(*array,mask,destination,control,length,position*);
SQR	SQR(*source,destination*);
SRT	SRT(*array,dim_to_vary,control,length,position*);
STD	STD(*array,dim_to_vary,destination,control,length,position*);
SUB	SUB(*source_A,source_B,destination*);
TAN	TAN(*source,destination*);
TND	TND();
TOD	TOD(*source,destination*);
TOF	TOF(*timer,preset,accum*);
TON	TON(*timer,preset,accum*);
TRN	TRN(*source,destination*);
UID	UID();
UIE	UIE();
XIC	XIC(*data_bit*);
XIO	XIO(*data_bit*);
XOR	XOR(*source_A,source_B,destination*);
XPY	XPY(*source_A,source_B,destination*);

Estructuración del formato de archivo de importación/exportación (.CSV) de tags

Introducción

Este capítulo explica la estructura general de un archivo de importación/exportación de tags. La extensión de archivo para un archivo de importación/exportación de tags es .CSV.

IMPORTANTE Para editar el archivo .CSV, recomendamos usar una herramienta de programa de base de datos, tal como Microsoft Access, o un editor de texto general. Muchas otras herramientas de escritorio, tales como Microsoft Word o Excel, pueden cambiar la estructura del archivo .CSV y causar el fallo de una importación del archivo. Para obtener más información sobre el uso de Excel para editar el archivo .CSV exportado, vea el Apéndice A.

Convenciones

La utilidad de importación/exportación de tags se basa en el formato CSV usado para programas de hojas de cálculo. Los ejemplos siguen estas convenciones:

Convención:	Significado:
`user_value`	los ítems en letra cursiva indican información proporcionada por el usuario
`LITERAL`	los ítems en mayúsculas indican una palabra clave o símbolo requerido que debe introducirse tal como se muestra.
`" "`	las comillas dobles deben encerrar algunos valores, tal como se muestra en los ejemplos

El formato CSV (valores separados por coma) usa comas para identificar información separada. Éste es un formato estándar usado por programas de hojas de cálculo.

Los caracteres de espacio en blanco incluyen espacios, tabulaciones, retorno de carro, nueva línea y alimentación de impresora. Estos caracteres pueden estar en cualquier lugar de un archivo de importación/exportación, excepto en palabras claves o nombres. Si hubiera espacios en blanco fuera de descripciones, éstos se ignoran.

Comentarios internos de archivo

Se pueden introducir comentarios para documentar los archivos de importación. El proceso de importación ignora estos comentarios. Se pueden colocar comentarios en cualquier lugar de un archivo de importación/exportación, excepto en nombres y descripciones. Los comentarios se introducen empezando la línea (registro) con REMARK y una coma.

Colocar información en un archivo de importación/ exportación (.CSV) de tags

El archivo de importación/exportación contiene dos componentes de información. Estos componentes son:

Ítem:	Identifica:
header information	el contenido del archivo de importación/exportación de tags cada línea es una línea de comentario
record	cada tag es un registro individual

El formato general es:

```
remark,CSV-Import-Export
remark,Date =
remark,Version = RSLogix 5000-2.10.00.00
remark,Owner =
remark,Company =
0.1
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
remark Controller Tags
TAG
.
.
ALIAS
.
.
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
remark 1st program
TAG
.
.
remark 1st program
ALIAS
.
.
remark last program
```

```
TAG
.
.
remark last program
ALIAS
.
.
```

Los tags globales preceden a los tags del programa.

Todos los registros en un archivo de importación de tags siguen esta estructura:

```
Type,Scope,Name,"Description","Datatype","Specifier"
```

Donde:

Ítem:	**Identifica:**
`Type`	el tipo de tag los tipos válidos son: TAG tag ALIAS tag de alias COMMENT componente de operando de tag
`Scope`	qué parte del proyecto es propietaria del tag si no se especifica un grupo, el grupo es controlador si se especifica un grupo, éste identifica al programa
`Name`	nombre del tag
`Description`	la descripción del tag (opcional) encierre entre signos de dos comillas
Datatype	el tipo de datos del tag – usar cualquier nombre de tipo de dato válido
Specifier	opcional • en el caso de un alias, especifica el tag de base • en el caso de un comentario de tag, especifica el nombre del tag y el miembro o bit

Especificar un registro de tag

Cada registro de TAG define un tag dentro de un proyecto del controlador. Un registro de TAG sigue este formato:

```
TAG,Scope,Name,"Description","Datatype","Specifier"
```

Especificar dimensiones

Las dimensiones del tag se especifican de la misma forma en que se introduce el tag en la lógica.

Para especificar:	Introduzca:
1 dimensión	número
2 dimensiones	número,número
3 dimensiones	número,número,número

Ejemplos de TAG

Los siguientes ejemplos muestran registros de TAG.

Ejemplo:	Descripción:
TAG,,timer_1,"this is the first timer","TIMER",""	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El tag tiene el nombre timer_1. La descripción es "this is the first timer." El tipo de datos es TIMER No hay especificadores adicionales para este tag.
TAG,,fault_record,"",fault_structure",""	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El nombre del tag es fault_record. No hay descripción. El tipo de dato es fault_structure definida por el usuario. No hay especificadores adicionales para este tag.
TAG,recipe_b,int_array,"",INT[10,10],""	Este tag está agrupado al programa con el nombre recipe_b. El tag tiene el nombre int_array. No hay descripción. El tipo de dato es INT[10,10] – un arreglo INT con dos dimensiones No hay especificadores adicionales para este tag.

Especificar un registro de alias

Cada registro de alias define un alias dentro de un proyecto del controlador. Un registro de ALIAS sigue este formato:

```
ALIAS,Scope,Name,"Description","Datatype","Specifier"
```

Los siguientes ejemplos muestran registros de ALIAS.

Ejemplo:	Descripción:
ALIAS,,hot,"","","temp"	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El tag de alias tiene el nombre hot. No hay descripción. No hay un tipo de datos para un alias, éste es el mismo que el tag de base. El especificador es el nombre del tag de base.
ALIAS,recipe_b,start_value,"have this much at first","","int_a"	Este tag está agrupado al programa con el nombre recipe_b. El nombre del tag de alias es start_value. La descripción es "have this much at first." No hay un tipo de datos para un alias, éste es el mismo que el tag de base. El especificador es el nombre del tag base.

Especificar de un registro de comentario

Cada registro de COMMENT define un comentario acerca de un componente de un registro de TAG. Un registro de COMMENT sigue este formato:

```
COMMENT,Scope,Name,"Description","Datatype","Specifier"
```

Los siguientes ejemplos muestran registros de COMMENT.

Ejemplo:	Descripción:
COMMENT,,timer_1,"this is the enable bit",,"timer_1.en"	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El comentario está asociado con el "timer_1" del tag. La descripción es "this is the enable bit of the first timer." No hay tipo de datos para un comentario. El especificador es el miembro tag asociado con el comentario.
COMMENT,,ratio,"this is the bit to monitor",,"ratio.3"	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El comentario está asociado con el "régimen" del tag. La descripción es "this is the bit to monitor." No hay tipo de datos para un comentario. El especificador es el bit asociado con el comentario.
COMMENT,recipe_b,table,"look at this element",,"table[8]"	Este tag está agrupado al programa con el nombre recipe_b. El comentario está asociado con la "tabla" del tag. La descripción es "look at this element." No hay tipo de datos para un comentario. El especificador es el elemento del arreglo asociado con el comentario.
COMMENT,,mask_1,"use this mask value",,""	No hay grupo especificado, por lo tanto es un tag de controlador cubierto. El comentario está asociado con la "mask_1" del tag. La descripción es "use this mask value." No hay tipo de datos para un comentario. No hay especificador porque esto especifica una descripción de un tag, no un componente de un tag. **Importante:** Si usa la instrucción COMMENT para un tag, ésta sobrescribe la parte de descripción de la instrucción TAG para dicho tag. El tag COMMENT básicamente define una nueva descripción para el tag.

Ejemplo de escenarios de importación/exportación de tags

Los siguientes ejemplos muestran cómo usar la importación parcial de tags (el modo de colisión es sobrescritura):

Escenario:	**Resultado:**
Exportar tags Editar atributos de tag, pero no nombre Importar tags nuevamente al proyecto del controlador	Cambió atributos de tag existentes de sobrescritura atribuidos
Exportar tags (contiene tag con el nombre Joe) Abrir archivo .CSV en Excel Cambiar Joe a Joseph Cerrar archivo Importar archivo nuevamente al proyecto del controlador	Tanto Joe como Joseph están en la lista de tags La lógica referente a Joe todavía se refiere a Joe
Exportar tags Añadir tags nuevos Importar tags nuevamente al proyecto del controlador	Se añaden los nuevos tags
Exportar tags Eliminar los tags Sam y Mitch Importar tags nuevamente al proyecto del controlador	Los tags Sam y Mitch todavía existen en la lista de tags

Ejemplo de importación/exportación

Introducción

Este capítulo muestra un ejemplo de un

- archivo de importación/exportación (.L5K) completo (ver a continuación)
- archivo de importación/exportación (.CSV) de tags, parcial (ver la página 6-7).

Estos ejemplos tienen el único propósito de ilustrar los resultados de importación/exportación. Es posible que la lógica no corresponda a su aplicación.

Ejemplo de archivo de importación/exportación completo

```
(********************************************

 Import-Export
 Version     := RSLogix 5000-2.25.00.00
 Owner     :=
 Exported  := Wed Sep 01 13:07:23 1999

********************************************)
IE_VER := 1.25;

CONTROLLER CNET_messaging (TimeSlice := 10,
                           CommDriver := AB_KT-1,
                           CommPath := "0, 2, 1, 0")

    DATATYPE structure1
        AB:1756_DI:I:0 data1;
        AB:1756_DO:O:0 data2;
        DINT array1[10];
    END_DATATYPE

    MODULE Local (Parent := Local,
        CatalogNumber := 1756-L1,
        Major := 3,
        Minor := 1,
```

```
        PortLabel := RxBACKPLANE,
        ChassisSize := 7,
        Slot := 0,
        Mode := 2#0000_0000_0000_0001,
        CompatibleModule := 2#0000_0000_0000_0000_0000_0000_0000_0000,
        KeyMask := 2#0000_0000_0001_1111)
END_MODULE

MODULE bridge1 (Parent := Local,
        CatalogNumber := 1756-CNB,
        Major := 1,
        Minor := 1,
        PortLabel := RxBACKPLANE,
        Slot := 3,
        Mode := 2#0000_0000_0000_0000,
        CompatibleModule := 2#0000_0000_0000_0000_0000_0000_1000_0000,
        KeyMask := 2#0000_0000_0001_1111)
END_MODULE

MODULE input1 (Parent := Local,
        CatalogNumber := 1756-IA16,
        Major := 2,
        Minor := 1,
        PortLabel := RxBACKPLANE,
        Slot := 1,
        CommMethod := 536870913,
        ConfigMethod := 8388610,
        Mode := 2#0000_0000_0000_0000,
        CompatibleModule := 2#0000_0000_0000_0000_0000_0000_1000_0000,
        KeyMask := 2#0000_0000_0001_1111)
        ConfigData := [28,16,1,0,0,0,1,9,1,9,0,0,0,0,65535,65535];
        CONNECTION StandardInput (Rate := 5000,
                EventID := 0)
                InputData := [0,0];
                InputForceData := [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        END_CONNECTION
END_MODULE
```

```
MODULE output1 (Parent := Local,
        CatalogNumber := 1756-OA16,
        Major := 2,
        Minor := 1,
        PortLabel := RxBACKPLANE,
        Slot := 2,
        CommMethod := 536870914,
        ConfigMethod := 8388610,
        Mode := 2#0000_0000_0000_0000,
        CompatibleModule := 2#0000_0000_0000_0000_0000_0000_1000_0000,
        KeyMask := 2#0000_0000_0001_1111)
        ConfigData := [28,18,1,0,0,0,0,0,0,0];
        CONNECTION Fused (Rate := 5000,
                EventID := 0)
                InputData := [0,0,[0,0],0,0,0,0];
                InputForceData := [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0];
                OutputData := [0];
                OutputForceData := [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        END_CONNECTION
END_MODULE

TAG
        float1 : REAL (RADIX := Float) := 0.0;
        integer1 : INT (RADIX := Decimal) := 0;
        module_info : structure1  := [[0,0],[0],[0,0,0,0,0,0,0,0,0,0]];
        PLC5C_sec_clock : INT (RADIX := Decimal) := 32;
        Source_data_buffer : DINT (RADIX := Decimal) := 300;
        L1_to_L1 : MESSAGE (MessageType := CIP Data Table Write,
                RequestedLength := 1,
                ConnectionPath := "input1, 2, 6, 1, 0",
                DF1DHFlag := 0,
                LocalTag := Source_data_buffer,
                RemoteElement := destination_data_buffer,
                CacheConnections := TRUE);
```

```
        L1_to_PLC5C : MESSAGE (MessageType := PLC5 Typed Read,
                RequestedLength := 1,
                ConnectionPath := "input1, 2, 1",
                DF1DHFlag := 0,
                LocalTag := PLC5C_sec_clock,
                RemoteElement := S:23,
                CacheConnections := FALSE);
END_TAG

PROGRAM MainProgram (MAIN := MainRoutine)
        TAG
        END_TAG

        ROUTINE MainRoutine
                RC: “PURPOSE - This rung continuously sends a 1-DINTmessage from the local 1756-L1 to a remote
1756-L1 over ControlNet.$N”
                        “N”
                        “SYSTEM CONFIGURATION - The local ControlLogix chassis consists of a 1756-L1 in slot 0 and
a 1756-CNB (node 5) in slot 1. The remote ControlLogix chassis also consists of a 1756-L1 in slot 0 and a 1756-CNB (node 6) in slot 1. The
two 1756-CNB modules are on the same ControlNet network.$N”
                        “$N”
                        “THE MESSAGE INSTRUCTION - Under the $'Configuration$' Tab in the message setup screen,
$"CIP Data Table Write$" is selected as the message type, the Source Tag is created as a DINT called $"source_data_buffer$", the
Destination Element is called $"destination_data_buffer$", and the Number Of Elements is one. Make sure that the Destination Element is a
tag created in the remote controller with the same name and same data type. These tags need to be created under the controller scope.$N”
                        “$N”
                        “Under the $'Communication$' Tab in the Message setup screen, the path to the remote controller is
specified. The following describes the path (1, 1, 2, 6, 1, 0) used in this example: $”1$” indicates a connection to the backplane of the
ControlLogix chassis, $”1$” indicates a connection to the CNB module in slot 1, $”2$” indicates a connection to port 2 of the CNB module
(get on the ControlNet wire), $”6$” indicates a connection to the remote CNB module at node address 6, $”1$” indicates a connection to the
backplane of the remote ControlLogix chassis, and finally, $”0$” indicates a connection to the remote controller in slot 0.$N”
                        ““;
```

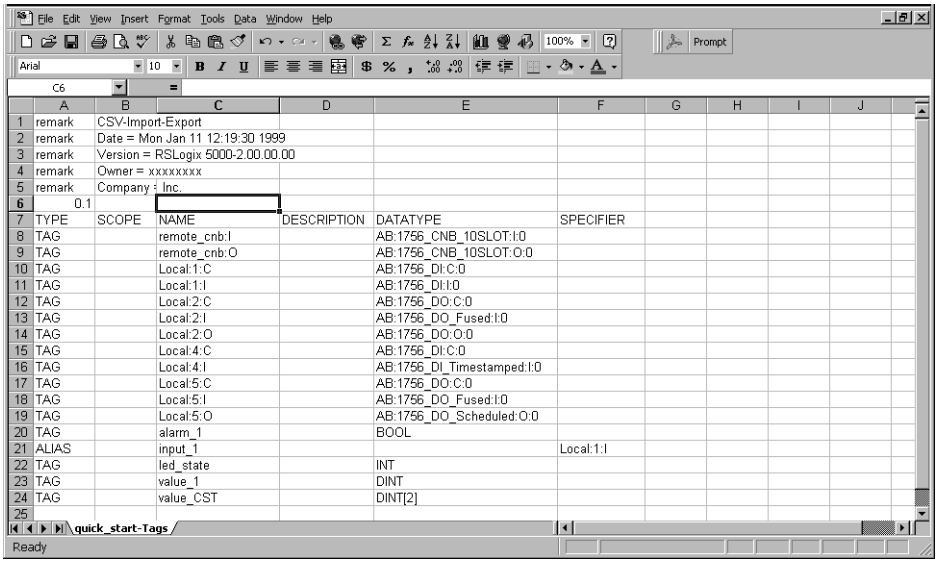
```
                N: XIO(L1_to_L1.EN)MSG(L1_to_L1);
                RC: “PURPOSE - This rung continuously reads data from location S:23 of a PLC5C at node 1.$N”
                    “$N”
                    “SYSTEM CONFIGURATION - The ControlLogix chassis consists of a 1756-L1 in slot 0 and a
1756-CNB (node 5) in slot 1. A PLC5C (node 1) is also on the same ControlNet network as the CNB module.$N”
                    “$N”
                    “THE MESSAGE INSTRUCTION - Under the $'Configuration$' Tab in the message setup screen,
$”PLC5 Typed Read$” is selected as the Message Type, the Source element is $”S:23$” of the PLC5C, the Destination Tag is named
$”PLC5C_sec_clock$”, and the Number Of Elements is one. Make sure that the destination tag is created under the controller scope and has
the same data type as the source tag (INT).$N”
                    “$N”
                    “Under the $'Communication$' Tab in the Message setup screen, the path to the PLC5C on
ControlNet is specified. The following describes the path (1, 1, 2, 1) used in this example: $”1$” indicates a connection to the backplane of
the ControlLogix chassis, $”1$” indicates a connection to the CNB module in slot 1, $”2$” indicates a connection to port 2 of the CNB
module (get on the ControlNet wire), $”1$” indicates a connection to the PLC5C at node address 1.$N”
                    ““;
                N: XIO(L1_to_PLC5C.EN)MSG(L1_to_PLC5C);
            END_ROUTINE


    END_PROGRAM


    TASK MainTask (TYPE := CONTINUOUS,
            WATCHDOG := 500.000,
            PRIORITY := 10,
            RATE := 10.000)
            MainProgram;
    END_TASK
```

```
        CONFIG CST(SystemTimeMasterID := 16#0000) END_CONFIG

        CONFIG DF1(DuplicateDetection := 1,
                ErrorDetection := BCC Error,
                EmbeddedResponseEnable := 0,
                DF1Mode := Pt to Pt,
                ACKTimeout := 50,
                NAKReceiveLimit := 3,
                ENQTransmitLimit := 3,
                TransmitRetries := 3,
                StationAddress := 0,
                ReplyMessageWait := 5,
                PollingMode := 1,
                MasterMessageTransmit := 0,
                NormalPollNodeFile := NA,
                NormalPollGroupSize := 0,
                PriorityPollNodeFile := NA,
                ActiveStationFile := NA,
                SlavePollTimeout := 3000,
                EOTSuppression := 0) END_CONFIG

        CONFIG SerialPort(BaudRate := 19200,
                Parity := No Parity,
                DataBits := 8 Bits of Data,
                StopBits := 1 Stop Bit,
                ComDriverId := DF1,
                PendingComDriverId := DF1,
                RTSOffDelay := 0,
                RTSSendDelay := 0,
                ControlLine := No Handshake,
                PendingControlLine := No Handshake,
                RemoteModeChangeFlag := 0,
                PendingRemoteModeChangeFlag := 0,
                ModeChangeAttentionChar := 27,
                PendingModeChangeAttentionChar := 27,
                SystemModeCharacter := 83,
                PendingSystemModeCharacter := 83,
                UserModeCharacter := 85,
                PendingUserModeCharacter := 85) END_CONFIG

END_CONTROLLER
```

Ejemplo de archivo de importación/exportación de tags

```
remark,CSV-Import-Export
remark,Date = Mon Jan 11 12:19:30 1999
remark,Version = RSLogix 5000-2.25.00.00
remark,Owner =
remark,Company =
1.25
TYPE,SCOPE,NAME,DESCRIPTION,DATATYPE,SPECIFIER
TAG,,remote_cnb:I,"","AB:1756_CNB_10SLOT:I:0",""
TAG,,remote_cnb:O,"","AB:1756_CNB_10SLOT:O:0",""
TAG,,Local:1:C,"","AB:1756_DI:C:0",""
TAG,,Local:1:I,"","AB:1756_DI:I:0",""
TAG,,Local:2:C,"","AB:1756_DO:C:0",""
TAG,,Local:2:I,"","AB:1756_DO_Fused:I:0",""
TAG,,Local:2:O,"","AB:1756_DO:O:0",""
TAG,,Local:4:C,"","AB:1756_DI:C:0",""
TAG,,Local:4:I,"","AB:1756_DI_Timestamped:I:0",""
TAG,,Local:5:C,"","AB:1756_DO:C:0",""
TAG,,Local:5:I,"","AB:1756_DO_Fused:I:0",""
TAG,,Local:5:O,"","AB:1756_DO_Scheduled:O:0",""
TAG,,alarm_1,"","BOOL",""
ALIAS,,input_1,"","","Local:1:I"
TAG,,led_state,"","INT",""
TAG,,value_1,"","DINT",""
TAG,,value_CST,"","DINT[2]",""
```

El ejemplo de archivo de tags anterior se parecería a éste si se abre usando un programa de hojas de cálculo:



	A	B	C	D	E	F	G	H	I	J
1	remark	CSV-Import-Export								
2	remark	Date = Mon Jan 11 12:19:30 1999								
3	remark	Version = RSLogix 5000-2.00.00.00								
4	remark	Owner = xxxxxxxx								
5	remark	Company = Inc.								
6	0.1									
7	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER				
8	TAG		remote_cnb:I		AB:1756_CNB_10SLOT:I:0					
9	TAG		remote_cnb:O		AB:1756_CNB_10SLOT:O:0					
10	TAG		Local:1:C		AB:1756_DI:C:0					
11	TAG		Local:1:I		AB:1756_DI:I:0					
12	TAG		Local:2:C		AB:1756_DO:C:0					
13	TAG		Local:2:I		AB:1756_DO_Fused:I:0					
14	TAG		Local:2:O		AB:1756_DO:O:0					
15	TAG		Local:4:C		AB:1756_DI:C:0					
16	TAG		Local:4:I		AB:1756_DI_Timestamped:I:0					
17	TAG		Local:5:C		AB:1756_DO:C:0					
18	TAG		Local:5:I		AB:1756_DO_Fused:I:0					
19	TAG		Local:5:O		AB:1756_DO_Scheduled:O:0					
20	TAG		alarm_1		BOOL					
21	ALIAS		input_1			Local:1:I				
22	TAG		led_state		INT					
23	TAG		value_1		DINT					
24	TAG		value_CST		DINT[2]					
25										

Consideraciones para usar Microsoft Excel para editar un archivo .CSV

Introducción

Este apéndice describe cómo el usar Microsoft Excel puede afectar un archivo .CSV.

IMPORTANTE Para editar el archivo .CSV, recomendamos usar una herramienta de programa de base de datos, tal como Microsoft Access, o un editor de texto general. Muchas otras herramientas de escritorio, tales como Microsoft Word o Excel, pueden cambiar la estructura del archivo .CSV y causar el fallo de una importación del archivo.

Recomendaciones

Si usa Microsoft Excel para editar su archivo de tags .CSV:

- Use signos de una comilla en lugar de signos de dos comillas dentro de las descripciones y comentarios.
- No cree descripciones o comentarios que consten sólo de números, tenga ceros precedentes o tenga un símbolo precedente que Microsoft Excel trate especialmente. Por ejemplo, no cree descripciones como:

  002
  +2
  =2
  -2
  .0

- No cree descripciones ni comentarios que empiecen con un símbolo +, – ó =. Aunque añada texto después del símbolo, Excel mostrará `#NAME?` en la celda.

Transformaciones de datos RSLogix5000

Cuando el software de programación RSLogix5000 exporta tags, realiza estas conversiones:

Contenido original:	Contenido en el archivo .CSV después de la exportación:	Detalles:
‘	$’	ningún problema con Excel
“	$”	Existe un problema general con el signo de dos comillas (") y el signo de dólar ($). Excel altera el contenido basado en estos símbolos (vea la sección Excel a continuación).
newline	NL	ningún problema con Excel
tab	$T	ningún problema con Excel
$	$$	ningún problema con Excel

Transformación de datos Microsoft Excel

Cuando usted abre el archivo .CSV exportado en Excel, estas conversiones ocurren:

Contenido original:	Contenido en el archivo .CSV después de la exportación:	Contenido después de abrir en Excel:	Contenido después de guardar desde Excel:	Detalles:
.0	“.0”	0	0	RSLogix5000 direcciona esto como el especificador de un tag. Si introdujo esto como un comentario completo, perderá cualquier punto precedente (.). Si introduce texto antes o después de esto, Excel mantiene el contenido.
=2	“=2”	2	2	Si introdujo esto como un comentario completo, perderá cualquier signo de igual precedente (=). Si introduce texto antes o después de esto, Excel mantiene el contenido.
+2	“+2”	2	2	Si introdujo esto como un comentario completo, perderá cualquier signo más precedente (+). Si introduce texto antes o después de esto, Excel mantiene el contenido.
002	“002”	2	2	Si introdujo esto como un comentario completo, perderá los ceros precedentes. Si introduce texto antes o después de esto, Excel mantiene el contenido.
cadena de prueba	“cadena de prueba”	cadena de prueba	cadena de prueba	Excel coloca comillas alrededor del contenido de la celda sólo si hay una coma incorporada. RSLogix5000 siempre coloca signos de dos comillas alrededor del texto. Pero RSLogix5000 igualmente puede manejar la descripción sin comillas.
“cadena de prueba”	“$”cadena de prueba$””	$cadena de prueba$””	“$cadena de prueba$””””	Tanto Excel como RSLogix5000 alteran el contenido cuando éste incluye un signo de dólar ($).
tiene “texto entre comillas” dentro de cadena	“tiene $”texto entre comillas$” dentro de cadena”	tiene $texto entre comillas$” dentro de cadena”	“tiene $texto entre comillas $”” dentro de cadena”””	Tanto Excel como RSLogix5000 alteran el contenido cuando éste incluye un signo de dólar ($).
esto tiene texto ‘incorporado’	esto tiene texto $’incorporado$’	esto tiene texto $’incorporado$’	esto tiene texto $’incorporado$’	Los signos de una comilla funcionan bien en ambos paquetes de software.

Contenido original:	**Contenido en el archivo .CSV después de la exportación:**	**Contenido después de abrir en Excel:**	**Contenido después de guardar desde Excel:**	**Detalles:**
+texto	"+texto"	#NAME?	#NAME?	No inicie una descripción ni comentario con un signo más (+).
-texto	"-texto"	#NAME?	#NAME?	No inicie una descripción ni comentario con un signo menos (–).
=texto	"=texto"	#NAME?	#NAME?	No inicie una descripción ni comentario con un signo de igual (=).

Notas:

E

F

I

L

M

O

P

R

T

V

Nos encontrará en www.rockwellautomation.com

En cualquier lugar en el que nos necesite, Rockwell Automation reúne las marcas líder en automatización industrial, incluyendo los controles Allen-Bradley, los productos de transmisión de potencia eléctrica Reliance Electric, los componentes de transmisión de potencia mecánica Dodge y los programas de Rockwell Software. La manera única y flexible en la que Rockwell Automation ayuda a sus clientes a lograr una ventaja competitiva está respaldada por miles de socios, distribuidores e integradores de sistemas autorizados en todo el mundo.

Sede central: 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414-382-2000, Fax: (1) 414-382-4444
Sede central europea: 46, avenue Hermann Debroux, 1160 Bruselas, Bélgica, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Sede central en España: Calle Doctor Trueta 113-119, 08005 Barcelona, España, Tel: (34) 93-295-90-00, Fax: (34) 93-295-90-01

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

Publicación 1756-6.8.4ES - Octubre 1999
Reemplaza la publicación 1756-6.8.4ES - Febrero de 1999
PN 957293-02