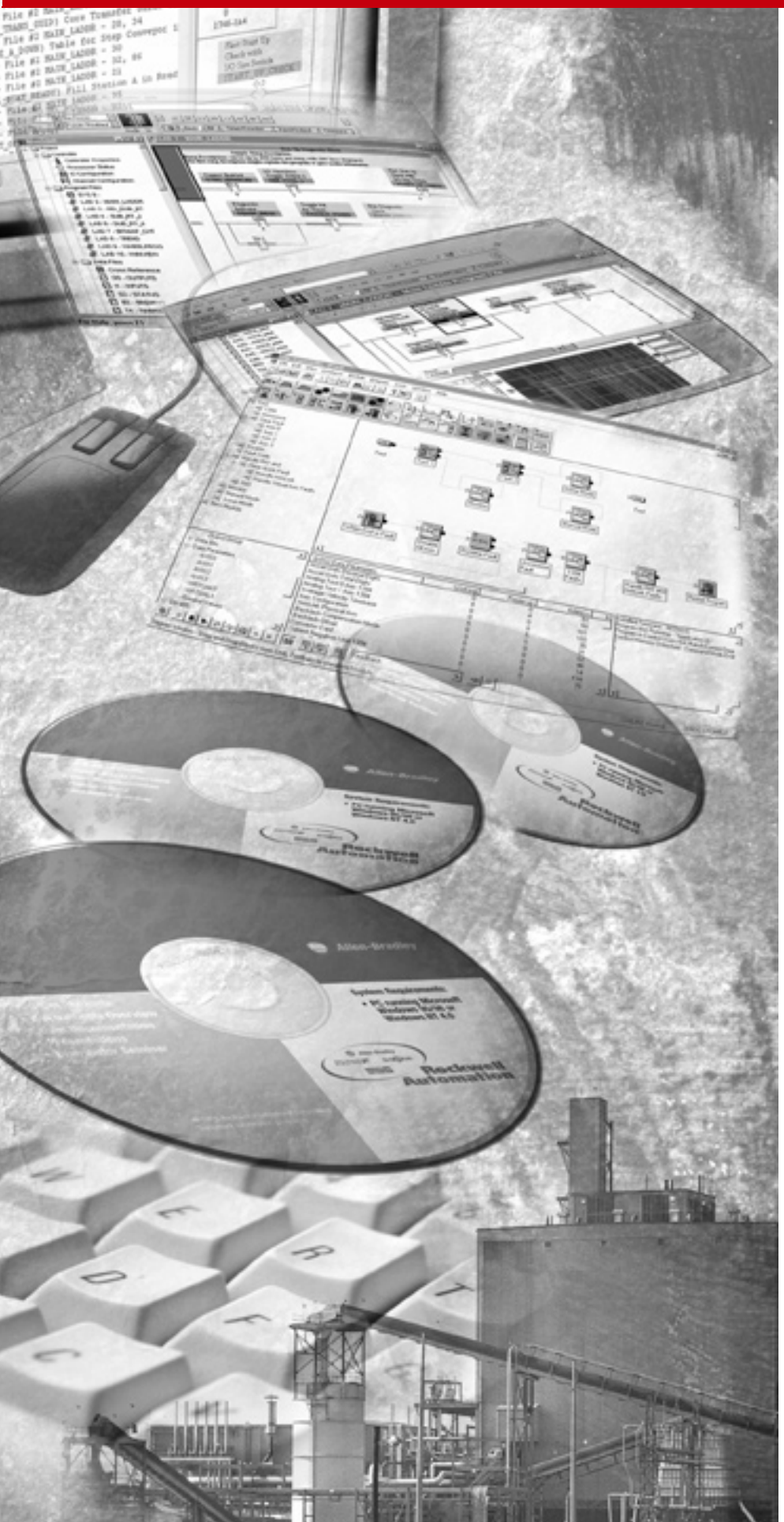


Allen-Bradley

Logix5550 Controller Import/Export

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

Reference Manual

Rockwell Automation

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of any safety considerations:

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss

Attention statements help you to:

- identify a hazard
- avoid a hazard
- recognize the consequences

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

Summary of Changes

This document describes how to use version 1.25 of the import/export utility that is included with RSLogix5000 programming software, release 2.25.00 and later.

IMPORTANT

Release 2.25 and later of RSLogix programming software supports the .L5k files created prior to version 1.25 of the import/export utility. The import/export utility converts earlier .L5K files to the version 1.25 format.

Releases of RSLogix5000 programming software prior to release 2.25 will not support .L5K files created with version 1.25 of the import/export utility.

Changes to this document include:

- The CommMethod and ConfigMethod items in the MODULE component now store numeric strings rather than text strings
- These new instructions are available:
  - Absolute Value (ABS)
  - Modulo (MOD)
  - Truncate (TRN)
  - Motion Axis Position Cam (MAPC)
  - Motion Axis Time Cam (MATC)
  - Motion Calculate Cam Profile (MCCP)

Table of Contents

Chapter 4

Entering Ladder Logic

Chapter 5

Structuring the Tag (.CSV) Import/Export File Format

Chapter 6

Import/Export Example

Appendix A

Considerations for Using Microsoft Excel to Edit a .CSV File

Importing and Exporting Files

Introduction

This document describes how to use version 1.25 of the import/export utility that is included with RSLogix5000 programming software, release 2.25.00 and later.

IMPORTANT

Release 2.25 and later of RSLogix programming software supports the .L5k files created prior to version 1.25 of the import/export utility. The import/export utility converts earlier .L5K files to the version 1.25 format.

Releases of RSLogix5000 programming software prior to release 2.25 will not support .L5K files created with version 1.25 of the import/export utility.

With a Logix5550 controller, you can import/export an entire project or you can import/export tags within a project.

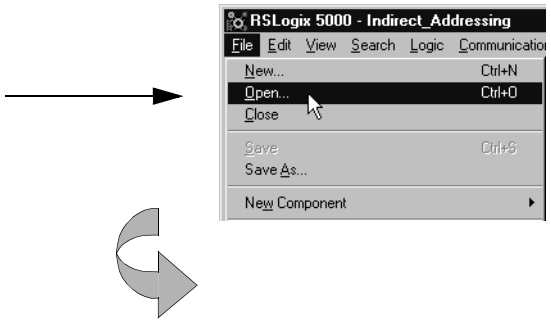
When you import or export a project, you use the entire project. The text file is a complete import/export file, that includes tag definitions, data, ladder logic, I/O configuration information, and controller configuration information. If you import or export tags, the text file is a partial import/export file, that includes only tag definitions and tag comments.

The structure of the import/export file depends on whether you perform a complete or partial import/export operation. There are also different considerations for complete and partial import/export operations. This chapter shows how to perform the import/export operations and describes any considerations.

Importing a Complete Text File into a Project

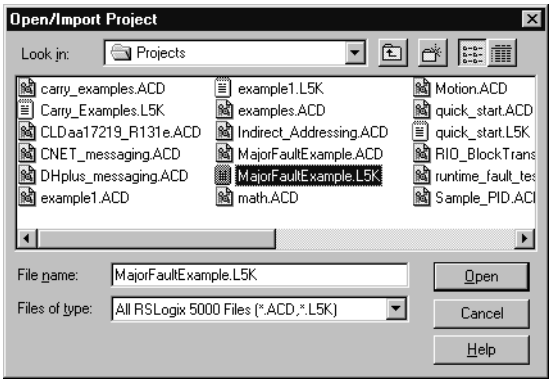
You can import controller information from a saved text file (that has a .L5K extension). This lets you use any text editor to create a project.

1. **Select File → Open.**



2. **Select the text file.**

The text file should have a .L5K extension.



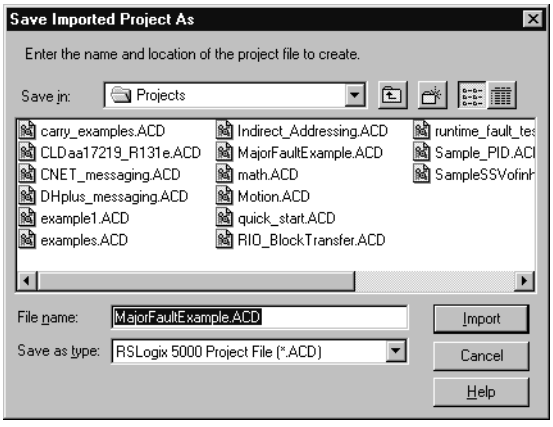
Select the file to import.

By default, the software points to the \RSLogix5000\Project folder. You can change the default via Tool → Options.

Specify the name for the file to import.

Click Open.

3. **Specify the name and location of the project**



Specify the project location.

Specify the project name.

Click Import.

If you import a project that has forces, the project defaults to Forces Disabled, even if the project was exported with Forces Enabled.

For more information about the structure of the complete import/export file, see:

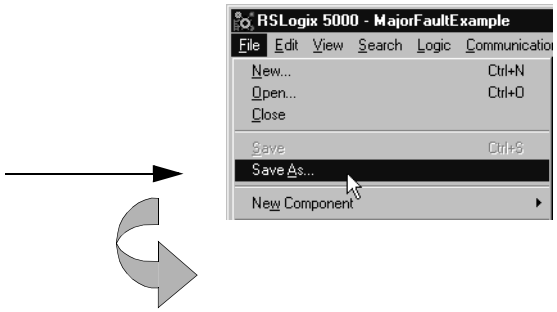
For information on:	See chapter:
structuring a complete import/export file	2
creating a complete import/export file	3
entering logic	4

Exporting a Complete Project to a Text File

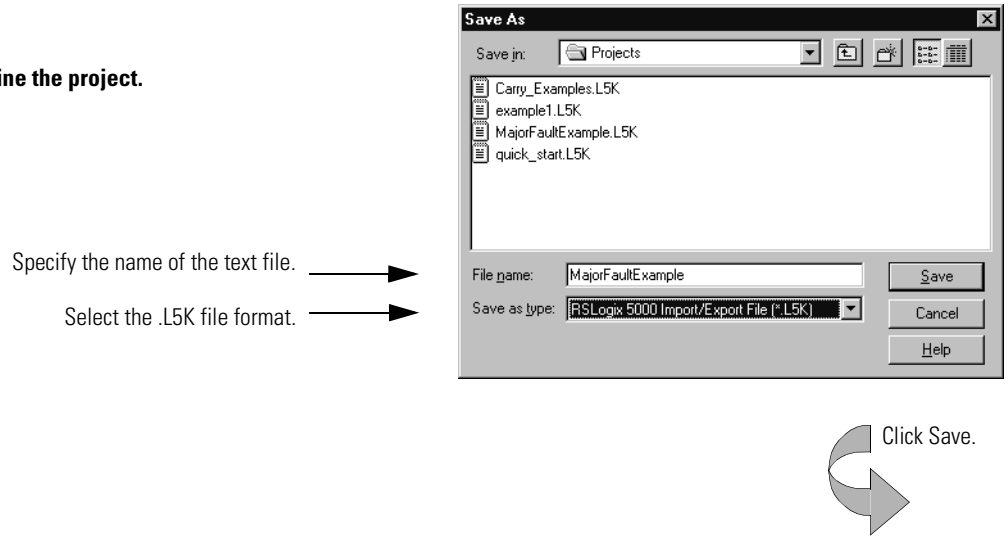
You can export the project to a text file. You can then use any text editor to modify the project.

Make sure the project you want to export is already open.

1. **Select File → Save As.**



2. **Define the project.**

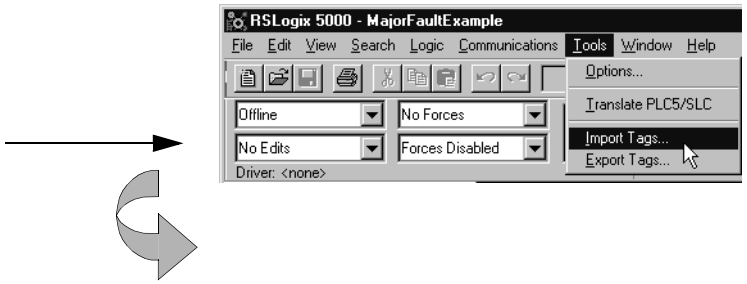


IMPORTANT Any unsaved edits are automatically saved when you OK the export operation.

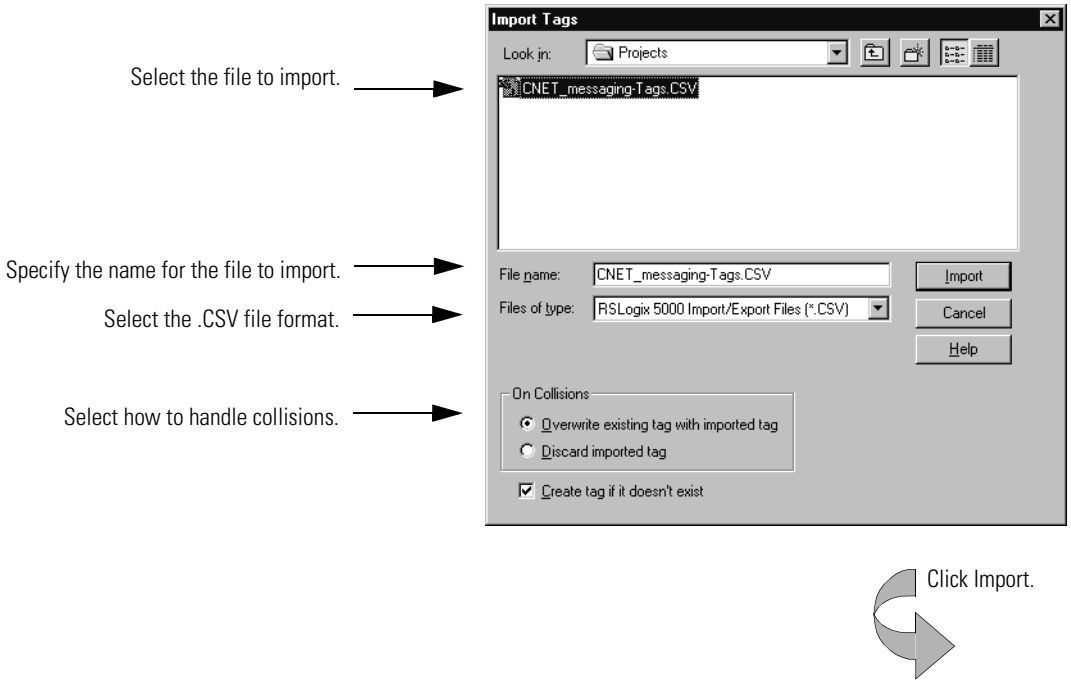
Importing a Tags Text File into a Project

When you are offline and have a project open, you can import tags from a saved text file (that has a .CSV extension). This lets you use a database program (like Microsoft® Access®) to create and edit tags.

1. **Select Tools → Import Tags.**



2. **Select the text file.** The text file must have a .CSV extension.



When you import tags, the possibility exists for tags in the import file to have the same name as tags already in the open project. This condition is a collision. You specify how to handle a collision when you select the tag file to import:

If you want:	Select:
replace the tag in the project with the tag from the import file	Overwrite (this is the default selection)
keep the tag that is in the project and discard the tag in the import file	Discard

It is also possible to have tags in the import file that do not exist in the open project. You can select whether to create these tags in the project.

If you delete tags from an import/export file and then import the file, tags are not deleted from the controller project. You have to use the programming software to delete tags from the tag list.

For more information about the structure of the partial import/export file, see:

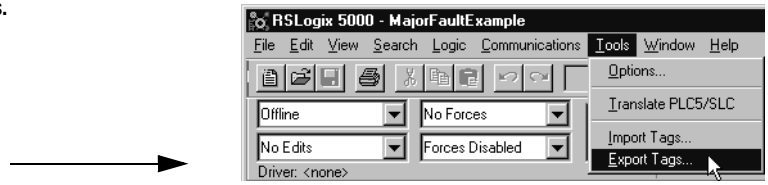
For information on:	See chapter:
structuring a partial import/export file	5

Exporting Tags to a Text File

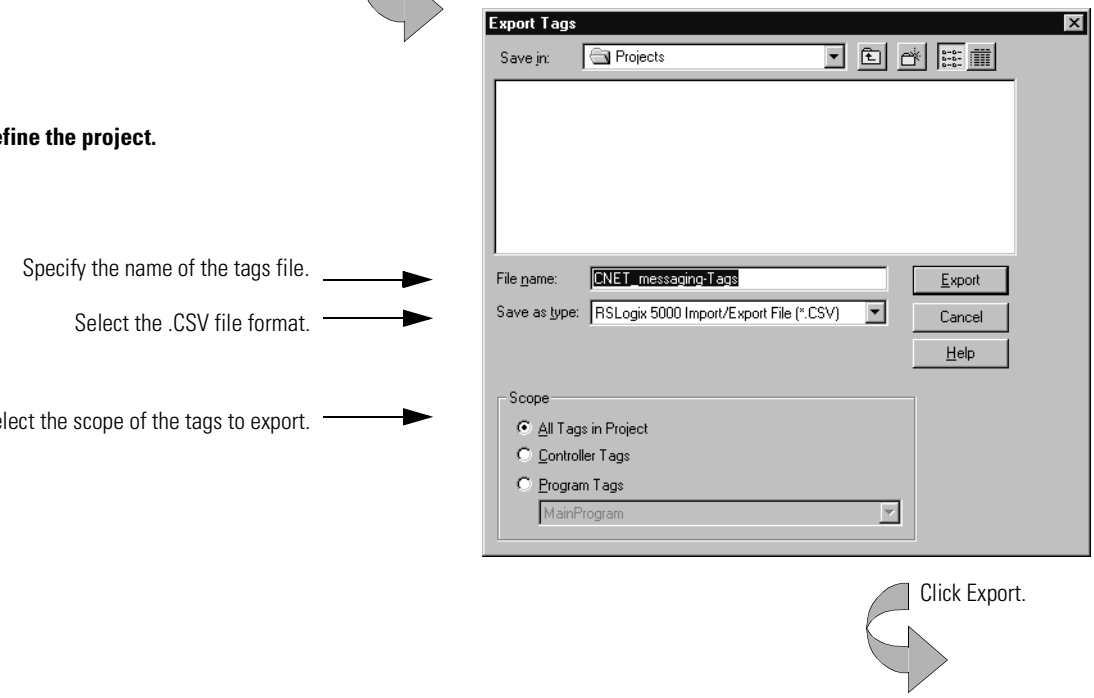
When you have a project open, you can export tags to a text file. You can then use a database program (like Microsoft Access) to edit tags.

Make sure the project you want to export tags from is already open.

1. **Select Tools → Export Tags.**



2. **Define the project.**



Selecting the scope of the tags to export

When you export tags, you have these choices as to which tags in the project you want to export.

Scope:	This option export:
All tags in project	all the tags (controller-scope and program-scope) in the project to a text file.
Controller tags	the controller-scoped tags of the project to a text file.
Program tags	the program-scoped tags of the program you specify. Use the drop-down arrow to display the available programs in the current project.

Structuring a Complete (.L5K) Import/Export File Format

Introduction

This chapter explains the overall structure of a complete import/export file. The file extension for a complete import/export file is .L5K.

For information about the specifics of each component in an import/export file, see the chapter "Creating an Import/Export File". For information on entering logic, see the chapter "Entering Logic."

Conventions

The import/export utility is based on the formats specified by the IEC 1131-3 specification. The examples follow these conventions:

Convention:	Meaning:
`< >`	items shown in angle brackets are required
`[ ]`	items shown in square brackets are optional
`user_value`	items in italics indicate user-supplied information
`LITERAL`	items in all uppercase indicate a required keyword or symbol that must be entered as shown
`"["`	items in double quotes are required characters

White space characters include spaces, tabs, carriage return, newline, and form feed. These characters can occur anywhere in an import/export file, except in keywords or names. If white space characters occur outside of descriptions, they are ignored.

Internal file comments

You can enter comments to document your import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in keywords, names, and component descriptions.

You can enter comments using either of these methods:

- Start the comment with two percent (%%) characters and stop at the end of the line
- Start the comment with a "(*" and end with a corresponding "*)".

Placing Information in an Import/Export File

The import/export file contains different components of information. These components are:

Component:	Identifies:
CONTROLLER	name of the project file
DATATYPE	user-defined and I/O data structures
MODULE	modules in the controller organizer
TAG	controller-scope tags
PROGRAM	program files
ROUTINE	routine files
TASK	controller tasks
CONFIG	configuration information

All components in an import export file follow this structure:

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

Where:

Item:	Identifies:
Component_Type	the component (as defined in above table)
component_name	a specific instance of the component
Attributes	any attributes of the component can also contain a description of the component separate each attribute with a comma (,)
body	any sub-components (children) of this component
END_Component_Type	end of the component information

Display style

Tags and data types support a radix attribute that lets you specify how to display the associated numerical information. The radix options are

- Binary (uses a 2# prefix)
- Octal (uses a 8# prefix)
- Decimal
- Hex (uses a 16# prefix)
- Exponential
- Float

Component descriptions

Descriptions of components are optional. Unlike internal comments, descriptions are imported. Place the description within double quotes. For example:

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

To enter control characters in the description, precede the character with a dollar sign ($). The following table shows how to enter the supported control characters in a description:

For this character:	Enter:
$	$$
‘	$‘
“	$”
10 (line feed)	$L or $l
13,10 (carriage return, line feed)	$N or $n
12 (form feed)	$P or $p
13 (carriage return)	$R or $r
9 (tab)	$T or $t
xxxx (4-digit character code that represents a hexadecimal value)	$xxxx

Defining a Controller

The CONTROLLER component is the overall structure of a project to be executed on one controller. It contains the configuration information and logic that you download to one controller. A CONTROLLER component follows this structure:

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

Where:

Item:	Identifies:
controller_name	the controller for the project
Attributes	attributes of the controller can also contain a description of the controller separate each attribute with a comma (,)
DATATYPE	I/O and user-defined data structures See page 3-1.
MODULE	devices in the controller organizer See page 3-4.
TAG	controller-scope tags See page 3-9.
PROGRAM	organization of routines See page 3-13.
TASK	organization of programs See page 3-15.
CONFIG	characteristics of controller objects (status information) See page 3-18.

Specifying CONTROLLER attributes

You can specify these attributes for a CONTROLLER:

Attribute:	Description:
`Description`	Provide information about the controller. Specify: `Description := "text"`
`TimeSlice`	Percentage of available CPU time (10-90) that is assigned to communications. Specify: `TimeSlice := value`
PowerLossProgram	Name of the program to be executed on reboot after a power loss. Specify: `PowerLossProgram := name`
MajorFaultProgram	Name of the program to be executed when a major fault occurs. Specify: `MajorFaultLossProgram := name`
CommDriver	Name of communications driver. Specify: `CommDriver := name`
CommPath	Name of communications driver path. Specify: `CommPath := name`

CONTROLLER guidelines

Keep these guidelines in mind when defining a data type:

- All declarations must be explicitly ordered as shown in the syntax above.
- There can only be at most 32 tasks.
- There can only be one continuous task.
- Programs can only be scheduled under one task
- Scheduled programs must be defined - i.e. must exist

CONTROLLER example

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

Creating a Complete Import/Export File

Introduction

This chapter explains how to enter project and configuration information in a complete import/export file.

For information on entering logic, see the next chapter.

Defining a Data Type

A DATATYPE component follows this structure:

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

Where:

Item:	Identifies:
`DataType_name`	the data structure
`Attributes`	attributes of the data structure can also contain a description of the component separate each attribute with a comma (,)
`member_definition`	each member of the data structure

Specifying DATATYPE attributes

You can specify these attributes for a DATATYPE:

Attribute:	Description:
`Description`	Provide information about the data type. Specify: `Description := "text"`
`ProductDefined`	Defines the structure as an I/O structure. Do not use for user-defined structures. Specify: `ProductDefined := 1`
`Radix`	Specify the display style as decimal, hex, octal, binary, exponential, or float. Specify: `Radix := value`
`Hidden`	Make the member a hidden member of the structure. Specify: `Hidden := 1`

Specifying a DATATYPE member

There are two kinds of data type members. A bit member is a member in which only a single bit of information is to be accessed. A non-bit member is a member that is defined as another data type (such as SINT, INT, DINT, COUNTER, etc.).

A non-bit member definition follows this structure:

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

All data types are allocated in 8-bit boundaries. A single bit of storage is not allowed, so a member cannot be a BOOL data type. To access a single bit, use the BIT declaration. BIT allows access to a single bit within a host member (a non-bit member).

A bit member uses the following syntax:

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

For example, to access two bits defined as MyBit0 and MyBit1, define a host member as MySint and reference the two bits within that host member. Normally, the host member is a hidden member because only the bit references are visible. The syntax for this example is:

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

IMPORTANT There **must** be a space between the host member name and the colon and the colon and the bit position because type names can contain a colon (for example, I/O structures) and without the space we could not tell where type name actually ends.

Bit members cannot be defined before their host members. Note that BitPosition zero is the least significant bit.

Specifying DATATYPE member attributes

You can specify these attributes for a member of a DATATYPE:

Attribute:	Description:
`Description`	Provide information about the data type member. Specify: `Description := "text"`
`Radix`	Specify decimal, hex, octal, binary, exponential, or float. Specify: `Radix := value`
`Hidden`	Make the member a hidden member of the structure. Specify: `Hidden := 1`

DATATYPE guidelines

Keep these guidelines in mind when defining a data type:

- Data types must be defined first within the controller body.
- Data types can be defined out of order. For example, if Type1 depends on Type2, Type2 can be defined first.
- Data types can be unverified. For example if Type1 depends on Type2 and Type2 is never defined, then Type1 will be accessible as an unverified type. Type2 will be typeless type. Tags of Type1 may be created but not of Type2.
- Data type members can be arrays but only one dimension is allowed.
- The following data types cannot be used in a user-defined data type: AXIS, MOTION_GROUP, and MESSAGE.

DATATYPE example

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

Defining a Module

A MODULE component follows this structure:

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

Where:

Item:	Identifies:
`device_name`	the module
`Attributes`	attributes of the module can also contain a description of the module separate each attribute with a comma (,)
`ConfigData`	operating characteristics of the module
`connection_list`	connection characteristics for the module see page 3-6

Specifying MODULE attributes

You can specify these attributes for a MODULE:

Attribute:	Description:
`Description`	Provide information about the module. Specify: `Description := "text"`
`Parent`	If this module is a child to another module, specify the name of the parent module. The parent module must be defined before any child module. Specify: `Parent := name`
`CatalogNumber`	Specify the catalog number of the module. Specify: `CatalogNumber := number`
`Major`	Specify the major revision number (1-127) of the module. Specify: `Major := number`

Attribute:	Description:
Minor	Specify the minor revision number (1-255) of the module. Specify: `Minor := number`
PortLabel	If this is a child module, specify the port used to reach this module from its parent. The port label is either `RxBACKPLANE` for modules in a chassis or a text string for modules on a network. Specify: `PortLabel := label`
ChassisSize	Specify the number of slots in the chassis (1-32). Specify: `ChassisSize := number`
Slot	Specify the slot number (1-31) where the module is in the chassis. Specify: `Slot := number`
NodeAddress	Specify the ControlNet node address (1-99) or the remote I/O rack address (0-63) of the module. Specify: `NodeAddress := number`
Group	If the module is a remote I/O module, specify the starting group (0-7). For a block-transfer module, this is the module group number under the remote I/O adapter. Specify: `Group := number`
CommMethod	Specify the method of connecting to the module. Specify: `CommMethod := number`
ConfigMethod	Specify the method of configuring the module. Specify: `ConfigMethod := number`
Mode	Select a specific mode by setting the appropriate bit. **Set:** **For:** 0 fault in the module causes major fault in controller 2 inhibit the module Specify: `Mode := number`
KeyMask	Specify whether to connect to the exact module that matches the electronic keying information (vendor, product code, product type, major revision, minor revision). No keying will connect to any module. Specify: `KeyMask := hex_string`
CompatibleModule	Specify whether to connect to a compatible module based on the minor revision (*value* = 1) or to an exact match of the module (*value* = 0). Specify: `CompatibleModule := value`
ChABaud	For a 1756-DHRIO module, specify the baud rate for channel A. Enter 57.5, 115.2, or 230.4. Specify: `ChABaud := baud`
ChBBaud	For a 1756-DHRIO module, specify the baud rate for channel B. Enter 57.5, 115.2, or 230.4. Specify: `ChBBaud := baud`

Specifying a MODULE connection list

You can specify these attributes for a connection list:

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

Where:

Item:	Identifies:
`connection_name`	the connection
InputData	input channel data
InputForceData	forcing information for the input channel
OutputData	output channel data
OutputForceData	forcing information for the output channel
`Attributes`	attributes of the connection can also contain a description of the module separate each attribute with a comma (,)

For details on the data in the connection list, see the user manual for the I/O module. The connection list data depends on the I/O module and the configuration for that module.

Forces appear as arrays of bytes under the InputForceData and OutputForceData attributes of the connection list. Do not modify forces in the import/export file. Use the programming software to enter and enable forces.

Specifying MODULE connection list attributes

You can specify these attributes for a MODULE connection list:

Attribute:	Description:
`Description`	Provide information about the connection list. Specify: `Description := "text"`
`Rate`	Specify the requested packet interval (RPI) rate in microseconds. Specify: `Rate := microseconds`
EventID	Exists for future support. For now: Specify: `EventID := NA`

MODULE guidelines

Keep these guidelines in mind when defining a module:

- Attributes must be explicitly ordered as shown in table above.
- A parent module must be defined before any definitions of its child modules.

MODULE example

```
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
```

Defining a Tag

You can define controller-scope tags and program-scope tags. Controller-scope tags are defined in the TAG component within the CONTROLLER component; program-scope tags are defined in a TAG component within a PROGRAM component within a CONTROLLER component. A TAG component follows this structure:

```
TAG
      [ tag_declaration ]
END_TAG
```

Within a tag list, message and motion tags must follow all non-motion tags and axis tags must follow motion group tags.

IMPORTANT For detailed information about atomic and structure tags and their supported attributes and ranges, see the *Logix5550 Controller User Manual*, publication 1756-6.5.12.

Defining a TAG declaration for a non-alias tag

A tag declaration for a non-alias tag follows this structure:

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

Where:

Item:	Identifies:
`tag_name`	name of the tag
type	type of tag supported atomic types: BOOL, SINT, INT, DINT, REAL predefined types: AXIS, CONTROL, COUNTER, MESSAGE, MOTION_GROUP, MOTION_INSTRUCTION, PID, TIMER
array	dimensional boundaries for array tags see page 3-10
`Attributes`	attributes of the tag can also contain a description of the tag separate each attribute with a comma (,) see page 3-11
`initial_value`	initial value of the tag see page 3-12

There **cannot** be any whitespace between the type and array definition. There **must** be a space between the tag name and the colon and another space between that same colon and the type name.

This is because type names can contain a colon and without the space it would be impossible to detect where the type name actually starts.

Defining a TAG declaration for an alias tag

A tag declaration for a non-alias tag follows this structure:

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

Where:

Item:	Identifies:
`tag_name`	name of the alias tag
alias	name of the base tag the alias tag references Specify: *`alias<specifier>`* Where the *specifier* is: a bit (.*bitnumber*), array element ([*element*]), or structure member (.*membername*) of the tag.
`Attributes`	attributes of the tag can also contain a description of the tag separate each attribute with a comma (,)

Defining an array specification within a TAG declaration

An array specification follows this structure:

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

Where:

Item:	Identifies:
`element`	the number of elements within the array dimension for example: [5, 10, 2]

Specifying TAG attributes

You can specify these attributes for a TAG:

Attribute:	Description:
`Description`	Provide information about the tag. Specify: `Description := "text"`
`Radix`	Specify the display style as decimal, hex, octal, binary, exponential, or float. Specify: `Radix := value`
`ProduceCount`	Specify the number of consumers allowed (0-255). Specify: `ProduceCount := value`
`Producer`	If the controller produces this tag, specify the name of the remote controller that consumes this tag. You must also specify a RemoteTag and RPI attribute. Specify: `Producer:= name`
RemoteTag	If the controller produces this tag to a controller that supports tag names, specify the name of the tag on the remote controller. You must also specify a Producer and RPI attribute. Specify: `RemoteTag := name`
RPI	If the controller produces this tag, specify the RPI value in milliseconds. You must also specify a Producer and RemoteTag attribute. Specify: `RPI := milliseconds`
RemoteFile	If the controller produces this tag to a PLC-5 controller, specify the PLC-5 file number (1-999) on the PLC-5 controller. You must also specify a Producer and a RPI attribute. Specify: `RemoteFile := number`
PLCMappingFile	If this tag is mapped to a PLC controller, specify the file number (0-65535). Specify: `PLCMappingFile := number`
PLC2Mapping	If this tag is mapped to a PLC-2 file, set this attribute to 1. Specify: `PLC2Mapping := 1`
Comment	Provide information about a tag component. Specify: `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag

IMPORTANT If consume information is provided on an alias tag, the alias tag is converted to a base tag before it can consume data.

Defining TAG initial values

The `initial_value` format follows the C-language initialization syntax, except that you use square brackets instead of curly brackets. The following table shows some examples of entering initial values.

If the tag is:	Enter:
single, atomic value	[*Value*]
structure with three members	[*Value1*, *Value2*, *Value3*]
structure with a nested structure	[*Value1*, [*Value2*, *Value3*], *Value4*]
structure with a nested array	[*Value1*, [*ArrayValue1*, *ArrayValue2*], *Value3*]

Defining a comment for a TAG component

The comment attribute of a tag declaration lets you provide information about a component of the tag, such as a specific bit, array element, or structure member. For example:

To add a comment to this operand:	Enter:
bit 3 of a tag	COMMENT.3 := "*description*"
element 8 of an array tag	COMMENT[8] := "*description*"
preset value of a tag	COMMENT.PRE := "*description*"

TAG guidelines

Keep these guidelines in mind when defining a tag:

- Tags must be defined after devices (if there no devices, then after the data types) within the controller body.
- Base tags and aliases can be defined out of order within a tag block.
- You cannot define a 2nd dimension without a 1st dimension or a 3rd diemsion without a 2nd dimension.
- The initial values must comply with the tag type and dimensions.
- Whitespace can not occur within the initial values or within the type/dimension specifier.

TAG examples

```
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
```

Defining a Program

A PROGRAM component follows this structure:

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

Where:

Item:	Identifies:
program_name	the program
Attributes	attributes of the program (such as MAIN or FAULT) can also contain a description of the program separate each attribute with a comma (,)
TAG	program-scoped tags follows same format as controller-scoped tags see page 3-9
ROUTINE	additional routines for this program see page 3-15

Specifying PROGRAM attributes

You can specify these attributes for a PROGRAM:

Attribute:	Description:
`Description`	Provide information about the program. Specify: `Description := "text"`
Main	Name of the main routine of the program. Specify: `Main := name`
Fault	Name of the program fault routine, if any. Specify: `Fault := name`

PROGRAM guidelines

Keep in mind these guidelines when defining a program:

- The MAIN and FAULT attributes can be defined in any order.
- The TAG declaration block must occur before the routine block.
- All tag collection declaration blocks that occur in a program definition block are imported as tags of a given program and can only be seen by routines under that program. Controller tags, on the other hand can be seen by routines in any program.

PROGRAM example

```
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
```

Defining a Routine

A ROUTINE component follows this structure:

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

Where:

Item:	Identifies:
routine_name	the routine
Attributes	attributes of the routine can also contain a description of the routine separate each attribute with a comma (,)
rungs	ladder logic for more information on entering logic, see chapter 3

For the syntax of entering rungs and instructions, see the next chapter.

Specifying ROUTINE attributes

You can specify these attributes for a ROUTINE:

Attribute:	Description:
`Description`	Provide information about the routine. Specify: `Description := "text"`

ROUTINE example

```
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
```

Defining a Task

A TASK component follows this structure:

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

Where:

Item:	Identifies:
`task_name`	the task
`Attributes`	attributes of the task can also contain a description of the task separate each attribute with a comma (,)
`program_name`	each program within the task

Specifying TASK attributes

You can specify these attributes for a TASK:

Attribute:	Description:
`Description`	Provide information about the task. Specify: `Description := "text"`
Type	Specify the type of task (PERIODIC or CONTINUOUS). There can be only one continuous task. Specify: `Type := type`
Priority	Specify the priority of a periodic task (1-15) Specify: `Priority:=number`
Rate	If the task is a periodic task, specify how often to run the task (1000-2,000,000,000 us). Specify: `Rate:=number`
Watchdog	Enter the watchdog timeout for the task (1000-2,000,000,000 us). Specify: `Watchdog :=number`

TASK guidelines

Keep these guidelines in mind when defining a task:

- Tasks must be defined after programs and before controller objects.
- There can only be at most 32 tasks.
- There can only be one continuous task.
- A program can be scheduled under only one task.
- Scheduled programs must be defined - i.e. must exist.

TASK example

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

The task attributes (Type, Priority, Rate, and Watchdog) can be defined in any order. The list of programs scheduled for a task are listed in the task declarations block, as shown above. The programs are executed in the order they are specified.

Defining a Controller Object

A CONFIG component defines controller objects and follows this structure:

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

Where:

Item:	Identifies:
`object_name`	the controller object For details on the available controller objects and their attributes, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1. This manual describes each object and its valid range of values.
`Attributes`	attributes of the controller object can also contain a description of the task separate each attribute with a comma (,)

Controller objects are optional. There can be only one of each controller object you choose to define. Controller objects appear at the end of the import/export file.

CONFIG examples

The following two examples show a DF1 controller object and a SerialPort controller object.

```
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
```

Entering Ladder Logic

Introduction

This chapter explains the how to enter ladder logic in a complete import/export file.

For information about:	See page:
Entering rung logic	4-1
Entering comments	4-3
Entering instructions in neutral text	4-3

Entering Rung Logic

You enter rung logic within a ROUTINE component in an import/export file. Each rung follows this structure:

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

Where:

Item:	Identifies:
RungType	the rung
RungNeutralText	the logic

The following rung types are available:

Rung type:	Description:
N	normal
I	insert
D	delete
IR	insert with a replace
rR	pending replace IR
R	replace
rI	pending replace I
rN	pending replace N
e	pending insert rung
er	pending replace rung

Rung guidelines

- Rungs are specified using neutral language. See the rest of this chapter for the neutral text format for the supported instructions.
- Each rung ends with a semicolon (;).

Rung example

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

Entering Branches

You can enter a single branch or simultaneous branches on a rung. A branch follows this structure:

```
[,BranchNeutralText ]
```

Where:

Item:	Identifies:
[ ]	the branch
,	the beginning of each branch within the branch, to account for simultaneous branches
space	the end of each branch within the branch, to account for simultaneous branches
BranchNeutralText	the logic

Example with a single branch

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

Example with two simultaneous branches

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

Entering Rung Comments

The comments for rungs are similar to those for components except that the syntax is a bit different. The rung comment syntax is:

```
RC: "comment" "more" "etc";
```

A rung comment must be followed by a rung.

Entering Neutral Text for Instructions

The following tables lists each instruction and its neutral text format.

IMPORTANT For details on the parameters of and instruction and its supported values, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Instruction:	Neutral text format:
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);`

Instruction:	Neutral text format:
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,master_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*);

Instruction:	Neutral text format:
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(axis,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(axis,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);`

Instruction:	Neutral text format:
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);`

Structuring the Tag (.CSV) Import/Export File Format

Introduction

This chapter explains the overall structure of a tag import/export file. The file extension for a tag import/export file is .CSV.

IMPORTANT To edit the .CSV file, it is recommended that you use a database program tool, such as Microsoft Access, or a raw text editor. Many other desktop tools, such as Microsoft Word or Excel, might change the structure of the .CSV file and cause an import of the file to fail. For more information on the implications of using Excel to edit the exported .CSV file, see appendix A.

Conventions

The tag import/export utility is based on the CSV format used by spreadsheet programs. The examples follow these conventions:

Convention:	Meaning:
`user_value`	items in italics indicate user-supplied information
`LITERAL`	items in all uppercase indicate a required keyword or symbol that must be entered as shown
`" "`	double quotes must enclose some values, as shown in the examples

The CSV (comma separated values) format uses commas to identify separate information. This is a standard format used by spreadsheet programs.

White space characters include spaces, tabs, carriage return, newline, and form feed. These characters can occur anywhere in an import/export file, except in keywords or names. If white space characters occur outside of descriptions, they are ignored.

Internal file comments

You can enter comments to document your import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in names and descriptions. You enter comments by starting the line (record) with REMARK and a comma.

Placing Information in a Tag (.CSV) Import/Export File

The tag import/export file contains two components of information. These components are:

Item:	Identifies:
header information	the content of the tag import/export file each line is a comment line
record	each tag is an individual record

The overall format is:

```
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
.
.
```

Global tags precede program tags.

All records in a tag import/export file follow this structure:

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

Where:

Item:	Identifies:
`Type`	the type of tag valid types are: TAG tag ALIAS alias tag COMMENT tag operand component
`Scope`	what part of the project owns the tag if no scope is specified, the scope is controller if a scope is specified, it identifies the program
`Name`	name of the tag
`Description`	description of the tag (optional) enclose in double quotes
Datatype	datatype of the tag - use any valid datatype name
Specifier	optional • for an alias, specifies base tag • for a tag comment, specifies the tag name and member or bit

Specifying a Tag Record

Each TAG record defines a tag within a controller project. A TAG record follows this format:

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

Specifying dimensions

You specify tag dimensions the same way as you enter the tag in logic.

To specify:	Enter:
1 dimension	number
2 dimensions	number,number
3 dimensions	number,number,number

TAG examples

The following examples show TAG records.

Example:	Description:
TAG,,timer_1,"this is the first timer","TIMER",""	There is no scope specified, so this is a controller scoped tag. The tag is named timer_1. The description is "this is the first timer." The datatype is TIMER There are no additional specifiers for this tag.
TAG,,fault_record,"",fault_structure",""	There is no scope specified, so this is a controller scoped tag. The tag is named fault_record. There is no description. The datatype is a user-defined fault_structure. There are no additional specifiers for this tag.
TAG,recipe_b,int_array,"",INT[10,10],""	This tag is program-scoped to the program named recipe_b. The tag is named int_array. There is no description. The datatype is INT[10,10] - a INT array with two dimensions. There are no additional specifiers for this tag.

Specifying an Alias Record

Each ALIAS record defines an alias within a controller project. An ALIAS record follows this format:

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

The following examples show ALIAS records.

Example:	Description:
ALIAS,,hot,"","","temp"	There is no scope specified, so this is a controller scoped tag. The alias tag is named hot. There is no description. There is no datatype for an alias, it is the same as the base tag. The specifier is the name of the base tag.
ALIAS,recipe_b,start_value,"have this much at first","","int_a"	This tag is program-scoped to the program named recipe_b. The alias tag is names start_value. The description is "have this much at first." There is no datatype for an alias, it is the same as the base tag. The specifier is the name of the base tag.

Specifying a Comment Record

Each COMMENT record defines a comment about a component of a TAG record. A COMMENT record follows this format:

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

The following examples show COMMENT records.

Example:	Description:
COMMENT,,timer_1,"this is the enable bit",,"timer_1.en"	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "timer_1". The description is "this is the enable bit of the first timer." There is no datatype for a comment. The specifier is the tag member associated with the comment.
COMMENT,,ratio,"this is the bit to monitor",,"ratio.3"	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "ratio". The description is "this is the bit to monitor." There is no datatype for a comment. The specifier is the bit associated with the comment.
COMMENT,recipe_b,table,"look at this element",,"table[8]"	This tag is program-scoped to the program named recipe_b. The comment is associated with the tag "table". The description is "look at this element." There is no datatype for a comment. The specifier is the element of the array associated with the comment.
COMMENT,,mask_1,"use this mask value",,""	There is no scope specified, so this is a controller scoped tag. The comment is associated with the tag "mask_1". The description is "use this mask value." There is no datatype for a comment. There is no specifier because this specifies a description of a tag, not a component of a tag. **Important:** If you use the COMMENT statement for a tag, it overwrites the description part of the TAG statement for that tag. The COMMENT tag essentially defines a new description for the tag.

Sample Scenarios of Importing/Exporting Tags

The following examples show how you can use the partial import of tags (the collision mode is overwrite):

Scenario:	Result:
Export tags Edit tag attributes, but not names Import tags back into controller project	Changed attributed overwrite existing tag attributes
Export tags (contains tag named Joe) Open .CSV file in Excel Change Joe to Joseph Close file Import file back into controller project	Both Joe and Joseph are in the tag list Any logic referring to Joe still refers to Joe
Export tags Add new tags Import tags back into controller project	New tags are added
Export tags Delete tags Sam and Mitch Import tags back into controller project	Tags Sam and Mitch still exist in tag list

Import/Export Example

Introduction

This chapter shows an example of a

- complete (.L5K) import/export file (see below)
- partial, tag (.CSV) import/export file (see page 6-7).

These examples are only meant to illustrate the import/export results. The logic may not apply to your application.

Example Complete Import/Export File

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

  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,
                        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0];
                OutputData := [0];
                OutputForceData := [0,0,0,0,0,0,0,0,0,0,0,0];
        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”
                        ““;
```

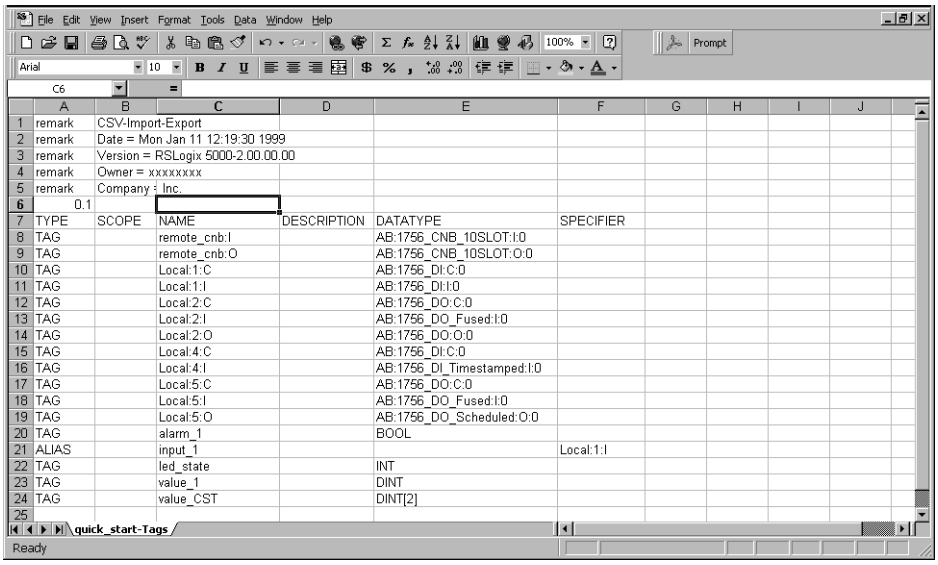
```
                N: XIO(L1_to_L1.EN)MSG(L1_to_L1);
                RC: “PURPOSE - This rung continuously reads data from location S:23 of a PLC5C at node 1.$N”
                    “$N”
                    “SYSTEM CONFIGURATION - The ControlLogix chassis consists of a 1756-L1 in slot 0 and a
1756-CNB (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
```

Example Tag Import/Export File

```
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]",""
```

The above example tag file would look like this if opened using a spreadsheet program:



	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										

Considerations for Using Microsoft Excel to Edit a .CSV File

Introduction

This appendix describes how using Microsoft Excel can affect a .CSV file.

IMPORTANT To edit the .CSV file, it is recommended that you use a database program tool, such as Microsoft Access, or a raw text editor. Many other desktop tools, such as Microsoft Word or Excel, might change the structure of the .CSV file and cause an import of the file to fail.

Recommendations

If you use Microsoft Excel to edit your .CSV tag file:

- Use single quotes instead of double quotes within descriptions and comments.
- Do not create descriptions or comments that consist only of numbers, have leading zeros, or have a leading symbol that Microsoft Excel treats specially. For example, do not create descriptions like:

  002
  +2
  =2
  -2
  .0

- Do not create descriptions or comments that start with a +, -, or = symbol. Even if you add text after the symbol, Excel displays `#NAME?` in the cell.

RSLogix5000 Data Transformations

When RSLogix5000 programming software exports tags, it performs these conversions:

Original content:	Content in .CSV file after export:	Details:
'	$'	no problems with Excel
"	$"	There is a general problem with the double quote (") and the dollar sign ($). Excel alters the content based on these symbols (see the Excel section below).
newline	NL	no problems with Excel
tab	$T	no problems with Excel
$	$$	no problems with Excel

Microsoft Excel Data Transformation

When you open the exported .CSV file in Excel, these conversions occur:

Original content:	Content in .CSV file after export:	Content after opening in Excel:	Content after saving from Excel:	Details:
.0	".0"	0	0	RSLogix5000 addresses this as the specifier for a tag. If you enter this as an entire comment, you lose any preceding period (.). If you enter any text before or after this, Excel maintains the content.
=2	"=2"	2	2	If you enter this as an entire comment, you lose any preceding equal sign (=). If you enter any text before or after this, Excel maintains the content.
+2	"+2"	2	2	If you enter this as an entire comment, you lose any preceding plus sign (+). If you enter any text before or after this, Excel maintains the content.
002	"002"	2	2	If you enter this as an entire comment, you lose any preceding zeros. If you enter any text before or after this, Excel maintains the content.
test string	"test string"	test string	test string	Excel puts quotes around cell contents only if there is an embedded comma. RSLogix5000 always places double quotes around text. But RSLogix5000 can still handle the description without quotes.
"test string"	"$"test string$""	$test string$""	"$test string$"""""	Both Excel and RSLogix5000 alter content when it includes a dollar sign ($).
has "quoted text" within string	"has $"quoted text$" within string"	has $quoted text$" within string"	"has $quoted text $"" within string"""	Both Excel and RSLogix5000 alter content when it includes a dollar sign ($).
this has 'embedded' text	this has $'embedded$' text	this has $'embedded$' text	this has $'embedded$' text	Single quotes work fine in both software packages.

Original content:	Content in .CSV file after export:	Content after opening in Excel:	Content after saving from Excel:	Details:
+text	"+text"	#NAME?	#NAME?	Do not start a description or comment with a plus sign (+).
-text	"-text"	#NAME?	#NAME?	Do not start a description or comment with a minus sign (-).
=text	"=text"	#NAME?	#NAME?	Do not start a description or comment with an equal sign (=).

E

F

G

I

L

M

N

O

P

R

S

T

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-6.8.4 - October 1999
Supersedes Publication 1756-6.8.4 - February 1999
PN 957236-55