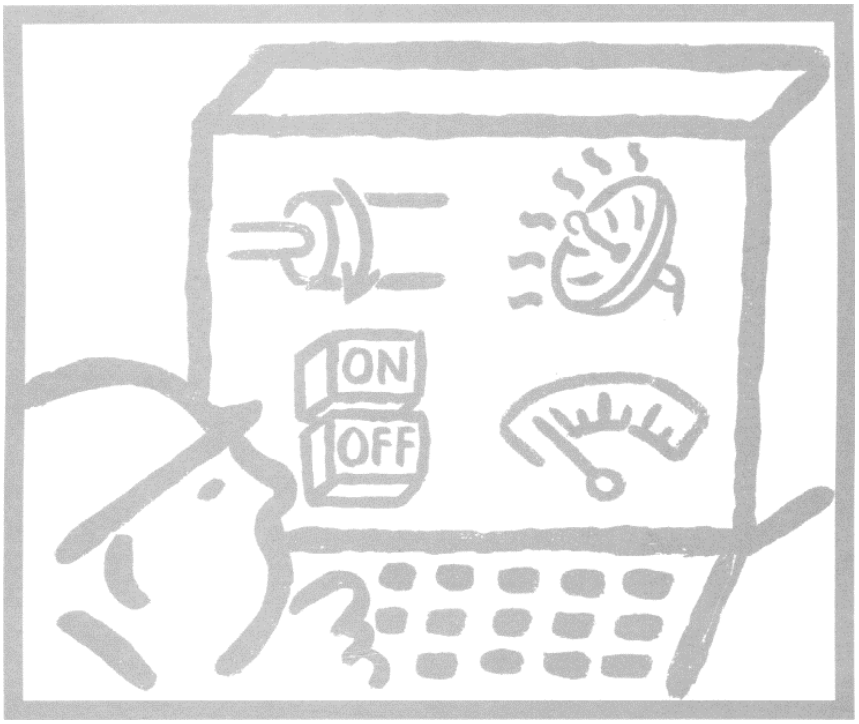
# ASIC-200 Version 5.0

integrated industrial control software




*User Guide*

Pro-face
XYCOM

| Revision | Description | Date |
|---|---|---|
| D | Name change, correct where applicable with document | 4/07 |

*User Guide 139837(B)*

Published by: Pro-face
750 North Maple Road
Saline, MI 48176



ASCI-200 release 5.0 documents include:

| | |
|---|---|
| Getting Started | 137586 |
| User Guide | 139837 |
| Language Reference | 139183 |
| HMI Guide | 139168 |

Note: the current revisions of each of these documents should be used.

Note: Features available on your system depend on product version and installed options (toolkits).

# Contents

# Introduction

This user guide describes how to use the control system to develop and run your application program, including:

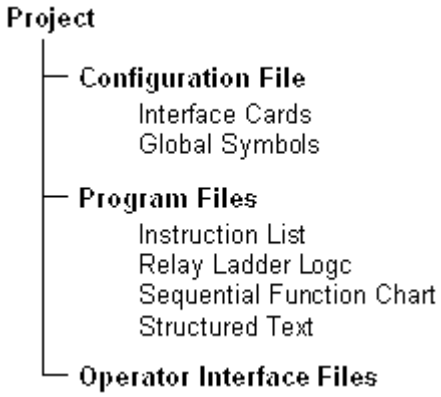
- Control Projects
- IO Configurations
- Symbol Configuration
- RLL Programming
- SFC Programming
- Structured Text Programming
- Instruction List Programming
- Motion Programming

It also discusses system options, how to use an uninteruptable power supply (UPS), DDE (Dynamic Data Exchange) and OPC (OLE for Process Control) communication, and Import/Export features. For information on standard functions and functions blocks, refer to the **Language Reference**. For information on using the operator interface editor, refer to the **HMI Guide**.

# Projects

## Managing Projects

Projects let you logically organize your work. Each project contains a group of program, configuration, and operator interface files that are used for a common purpose. A project groups and organizes the application programs and configuration files for a control application in a unique folder.

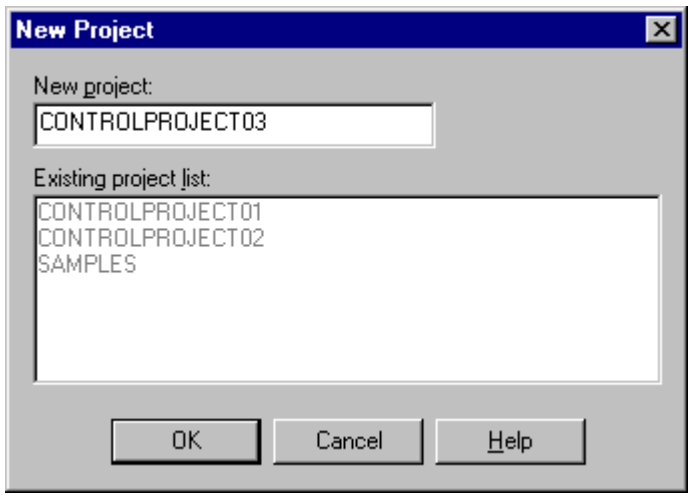



All project management functions are performed from the Program Editor. The following notes pertain to projects:

- The Program Editor can have only one project open at a time.
- The project name should identify the common purpose of the files.
- The active project is displayed in the Program Editor window title bar.

### Creating a New Project

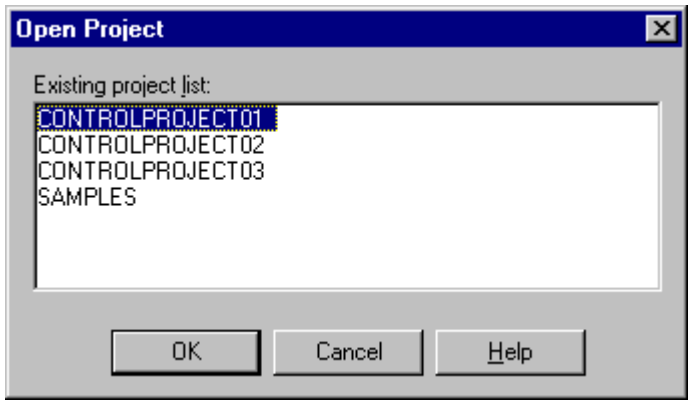
1. Select *New* from the *Project* menu. The *New Project* dialog box appears.
2. Type the name of the new project and click *OK*.




**Note:** If there was an active project open it is closed and the new project becomes the active project.

## Opening a Project

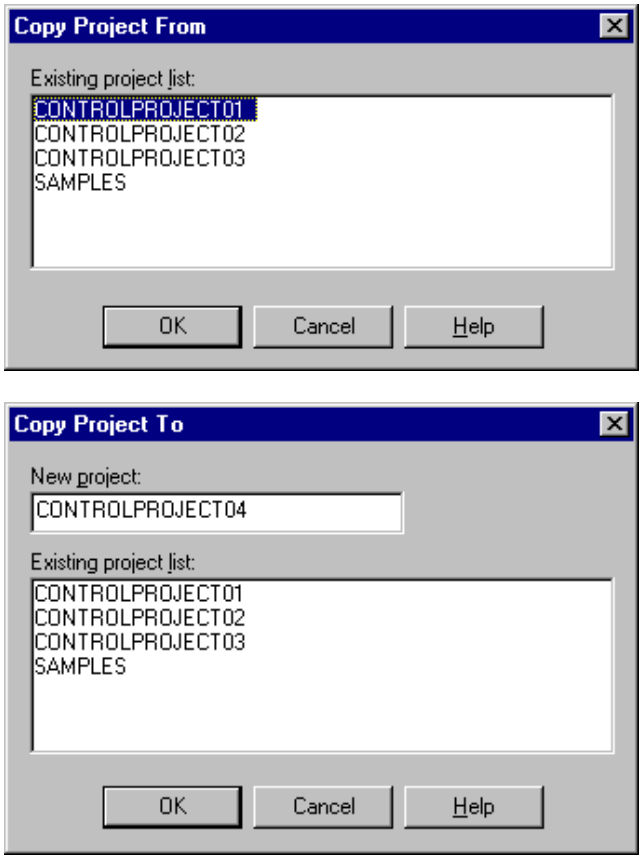
1. Select *Open* from the *Project* menu. The *Open Project* dialog box appears.
2. Select the project you want to open and click *OK*.




When a project is opened it becomes the active project. The previously active project is closed along with any open program files. When a project is opened, all program files that were previously open in the project are opened again, and the configuration file that was last active in that project is activated.

## Copying a Project

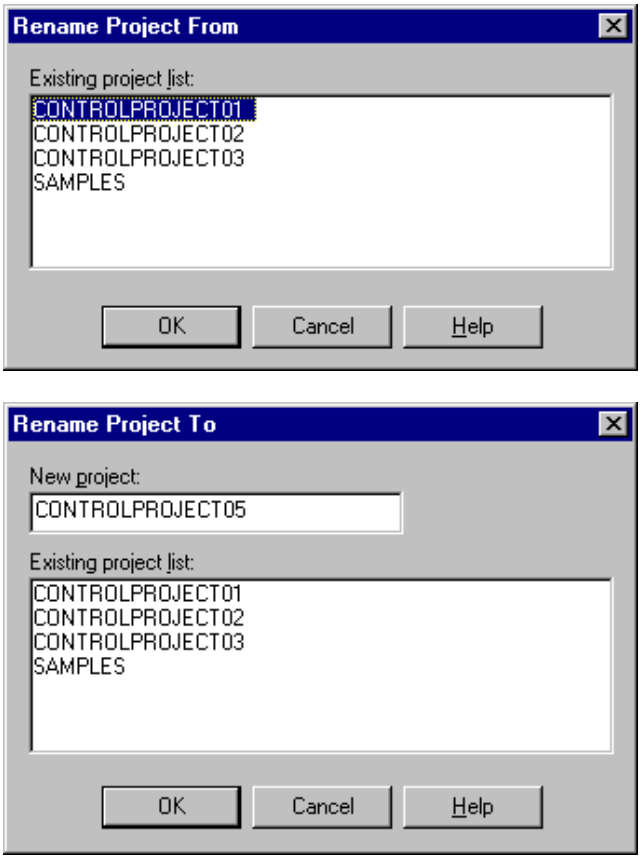
1. Select *Copy* from the *Project* menu. The *Copy Project From* dialog box appears.
2. Select the project you want to copy and click *OK*. The *Copy Project To* dialog box appears.
3. Type the name of the copy and click *OK*.




When a project is copied, all of the program, operator interface and configuration files in that project are copied into the new project, and the new project becomes the active project. Program files that were previously open in the source project are copied to the new project and opened. The configuration file that was active in the source project is copied to the new project and activated.

## Renaming a Project

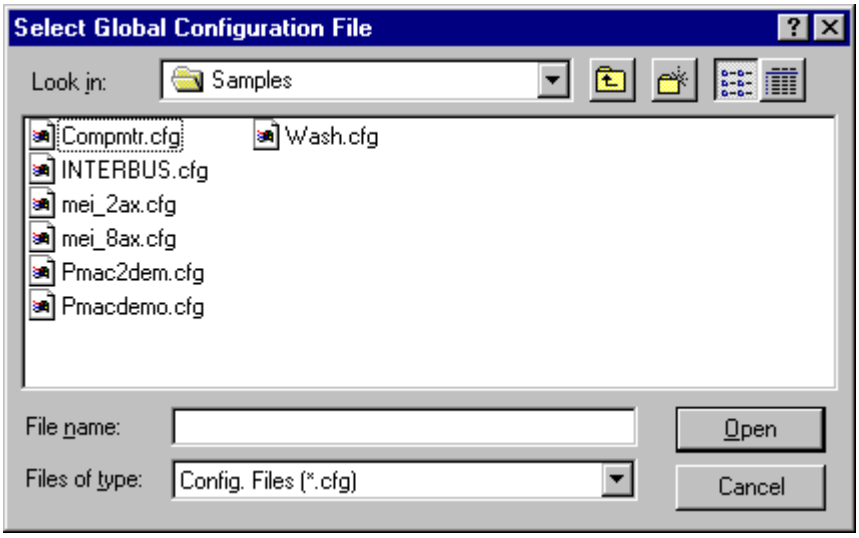
1. Select *Rename* from the *Project* menu. The *Rename Project From* dialog box appears.
2. Select the project you want to rename and click *OK*. The *Rename Project To* dialog box appears.
3. Type the new name of the project and click *OK*.




After a project is renamed, it becomes the active project.

## Activating a Configuration

1. Select *Activate* from the *Project* menu. The *Select Global Configuration File* dialog box appears. It displays a list of configuration files defined for the current project.
2. Select the configuration you want to activate and click *OK*.


Select Global Configuration File
Look in:
Samples
Compmtr.cfg
INTERBUS.cfg
mei_2ax.cfg
mei_8ax.cfg
Pmac2dem.cfg
Pmacdemo.cfg
Wash.cfg
File name:
Open
Files of type:
Config. Files (*.cfg)
Cancel

# I/O Configuration

## Overview

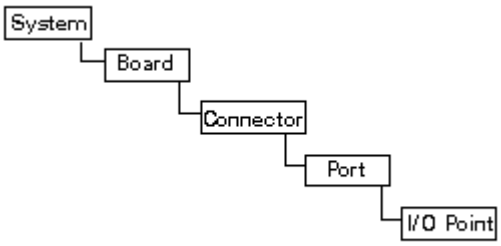
The System Configuration dialog box is used to configure and display the I/O hardware that is used across all the control programs in a given project.

The configuration tells the software what interface cards are in the computer and what I/O are attached to those cards. For motion control applications, it also contains information about what axes are present in the system and to which cards they are attached. You assign symbol names to the I/O points using the configuration editor and optionally enter comments regarding the function of particular elements.

The configuration editor takes you step by step through the configuration process. You start by naming the system (the computer) you are working on and defining the program scan time. The PC slots follow this on the system and the control system cards (not the basic computer cards) in each slot, and then to each I/O module connected to the card, and finally the I/O points and axes.

If a card is a communications card or I/O scanner, then the configuration goes even deeper to describe remote racks, slots, boards etc.

The following figure shows the configuration levels for simple onboard I/O:



# Help with Hardware Conflicts

There are several settings that must be properly configured to communicate with most interface cards. The most common are the:

- data port address
- memory base address
- interrupt

For the interface card to work properly, the jumper or DIP switch settings on the card must match the settings in the *Define Board* dialog box **AND** not conflict with other hardware in the PC.

**IMPORTANT:**

The best way to determine what resources your current hardware devices are taking up is to look in the NT Diagnostics in the *Administrative Tools* menu. This displays most of the hardware resources that are in use on your system. Be aware that some devices that are in use may not report this usage to the NT diagnostics, and devices that are not currently in use will not be reported. **However, these devices still can cause conflicts.** The NT diagnostics are a good place to start.

## Data Port

The data port is often used by the device driver to communicate with the card. Some port addresses are standard across most PCs and should not be used. The following is a list of common port uses:

COM1: 3F8-3FF
COM2: 2F8-2FF
COM3: 2E8-2FF
COM4: 3E8-3FF
LPT1: 378-37A
Floppy: 3F0-3F7
Video: 3B0-3BB and 3C0-3DF

## Memory Address

The memory address is used to set-up a shared memory area for the interface card. Normally memory from C800-DFFF will be available. This area memory is normally used by special-function cards like the an interface card. For example: if the card uses 4000 bytes hex of shared memory and starts at D0000, it will use D0000-D3FFF.

## Interrupt

The interrupt is used to communicate with the card. Valid interrupt values are between 0 and 11.

Some interrupt levels are standard across most PCs. Here is a list of common uses:

COM1 and COM3: IRQ4
COM2 and COM4: IRQ3
Floppy: IRQ6
LPT1: IRQ7

IRQ1 and 2 are always in use by the system hardware.

# Configuration Files

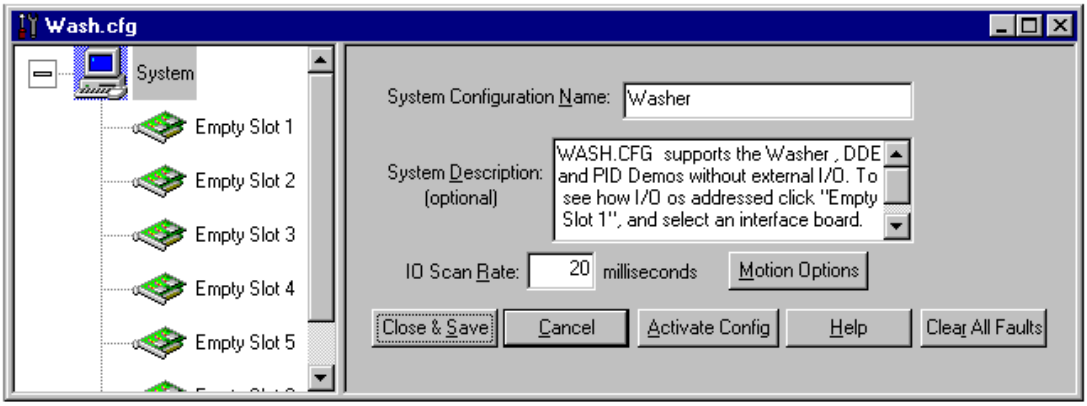
## Elements of a Configuration

I/O stands for Inputs and Outputs to and from the industrial controller. A configuration lets you define the I/O structure and assign tag names to I/O points and I/O ports. For applications with motion control the configuration lets you define motion parameters. All configurations have a *.cfg file name.

All configuration operations are performed from the Program Editor.

## Creating a New Configuration

- Select *New Config* from the *File* menu. The configuration dialog box appears.




## Editing an Existing Configuration File

1. Select *Open Config* from the *File* menu. The configuration dialog box appears.
2. Edit the fields as desired and click *Close* and *Save*.

## Activating a Configuration File

Select *Activate Config* from the *Project* menu to activate a new configuration file for the active project. The name of the active configuration file is displayed on the lower status bar of the Program Editor.

The active file is used by the runtime environment to determine the structure of the I/O systems and global symbol naming. The active configuration file is also used by the Program Editor to make system level Input, Output, and Memory symbols available to the user while creating programs.

When a new configuration file is activated, active programs are cancelled if information from the previously active configuration does not exist in the new configuration. The activation of a configuration file is recorded in the active project.

Whenever that project is opened, the active configuration file for that project will be activated.

## Saving a Configuration File

### To save the active configuration

- Select *Save* from the *File* menu or click the save button  on the editor tool bar.

When you save a configuration for the first time, the Program Editor displays the *Save As* dialog box, so you can name your configuration. Otherwise, if you want to change the name or folder of the active configuration before you save it, use the *Save As* file command.

### To save a configuration with a new name

1. Click *File* and select *Save As*.
2. The Program Editor displays the *Save As* dialog box so you can change the name of the configuration and folder.

### To save a configuration to a previous release format

You can save a configuration to an earlier format to insure compatibility with a target runtime system that is of an earlier release than the development system, for example.

1. Click *File* and select *Save As*.
2. The Program Editor displays the *Save As* dialog box so you can change the name of the configuration and folder. Additionally, you can change the file type to ASIC Revision 4.01.

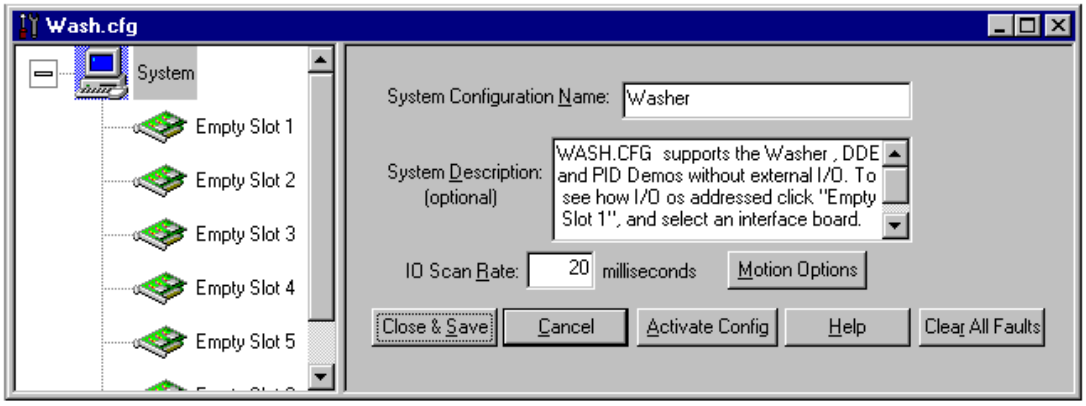
When you save a file to revision 4.01 (or any revision less than the current revision), any features in the current version that are not in the previous version you are saving to are permanently removed. Even if you open the file in a version you originally created it in, these features will not be present.

# Configuration Utility

## System Configuration

To edit the system configuration, select the *System* item in the configuration tree. Using the *System Configuration* dialog box, you can provide a system name to the configuration. The file name must be a valid DOS type eight character name. No spaces are allowed. The system name in the example shown below is *Washer*.

Configuration files are stored by default in the current project directory.




### *I/O Scan Rate*

You configure the I/O scan interval in the *System Configuration* dialog box. The I/O scan specifies the intervals at which control programs update the I/O and execute program logic. Lower priority Windows tasks are interrupted by the I/O scan. At each I/O scan, active control programs begin execution and run until completion.

Before program logic is solved, the I/O scanner reads inputs. After the control logic is executed, outputs are written. The executing time of control logic is variable, since the numbers and type of instructions active in any I/O scan interval is variable. Once outputs are updated, the control task is suspended and other lower priority NT tasks resume execution.

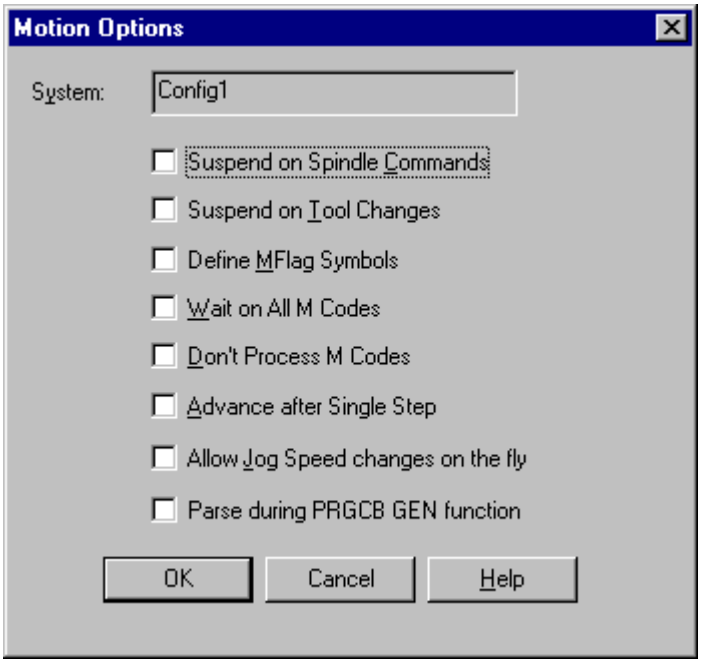
When defining a scan rate, consider:

- Minimum input pulse width - the minimum time an input must maintain a state to be recognized by the software.

- Minimum throughput - the minimum time for the control system to produce a change in state of an output in response to a change in state of an input.
- Maximum throughput - The maximum time for the control system to produce a change in state of an output in response to a change in state of an input.

### *Motion Options*

To edit motion options, click on *Motion Options* in the Configuration Utility. The *Motion Options* dialog box allows you to enter optional motion configuration settings. Some of these options were developed for specific or customized applications and may not apply to your application. For more information, refer to Configuring Motion Options.

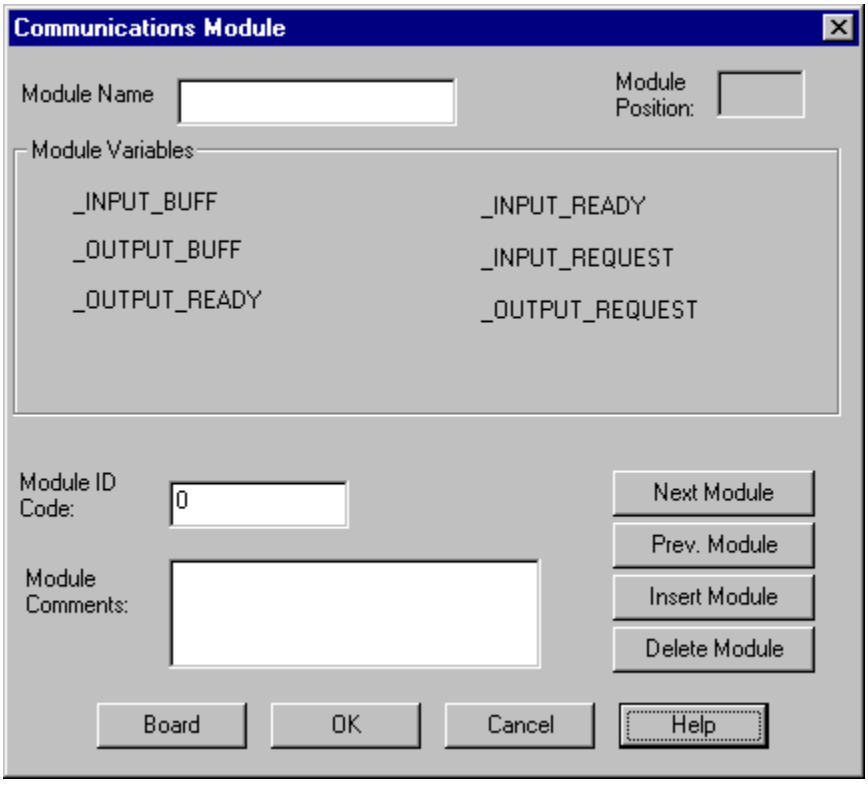



| Option | Description |
|---|---|
| Suspend on Spindle Commands | In an RS-274D program, suspends motion execution whenever spindle (S, M3, M4 or M5) commands are executed. Refer to Using the Suspend on Spindle Commands Feature for more information |

| Option | Description |
|---|---|
| Suspend on Tool Changes | In an RS-274D program, suspends motion execution whenever a tool (T) command is executed. Refer to Using the Suspend on Tool Changes Feature for more information. |
| Define MFlag Symbols | Configures the control system to generate global motion symbols Refer to Using the Define M Flag Symbols Feature for more information. |
| Wait on All M Codes | The control system software has the ability to wait on RS-274D M Codes. Once the M code processing is done, the control system software lets the application program wait for your logic to inform the system that the operation has completed and for execution to resume with the next sequential RS-274D block. Refer to Using the Wait on All M Codes Feature for more information. |
| Don't Process M Codes | The control system software supports the special M code functionality as described in the RS-274D specification. Clicking this check causes the application program to ignore the following M codes:<br>M3 (spindle positive)<br>M4 (spindle negative)<br>M5 (spindle stop)<br>Refer to Using the Do Not Process M Codes Feature for more information. |
| Advance after Single Step | Advance the motion block pointer to the next block. |
| Allow Jog Speed changes on the fly | For MEI controllers only. Allows jog speed changes without having to come to a stop first. |
| Parse during PRGCB GEN function | When generating an SFC program from an NC program, parses the SFC program as well. |

## Using the Define Board Dialog

1. From the configuration dialog box, click *Define Board*.
2. Set the board attributes as appropriate and click *OK*.




## Using the Connector Dialog

Each type of I/O board has a slightly different dialog box for configuring I/O points. For more specific information about configuring I/O points, see the help file for the driver you are using.

A name given to an I/O point appears in the symbol list when using the Program Editor along with the hardware definition information.

# Symbol Configuration

## Symbols

Constants, variables, and function block instances are given symbolic (identifier) names by the programmer. I/O points are given symbolic names in the I/O configuration. System-defined variables (e.g., run-time variables) are given symbolic names by the programmable control system. Collectively, the term **symbol** refers to these items. Each symbol has an associated data type. Furthermore, symbols have a scope (global or local) and associated properties.

Symbols are defined and viewed in the Symbol Manager. The content of the information is defined by the data type and can be real, integer, string of characters, etc. The scope of a symbol can be local to a program or global. Global symbol definitions are stored in the active configuration file. Local symbol definitions are stored within the application program file. I/O points and system and run-time symbols are predefined and can be viewed and used in programs, but not edited. They appear as global symbols.

Refer to the following table for a description of symbol types.

| Symbol Types | Description |
|---|---|
| Local symbol | A local symbol can be referenced only within the program in which it is defined or in macro steps called by an SFC program. The program must be open for you to define local symbols to be used within in it. You cannot access a local symbol within the Operator Interface or for DDE operations. |
| Global symbol | All programs within a project can reference a global symbol. You can use global symbols within the Operator Interface and for DDE operations |

| Symbol Types | Description |
|---|---|
| I/O points | I/O points are external locations. Because you can reference them in a program just like symbols, I/O points appear in the Symbol Manager and are listed as global symbols.<br>I/O points function like global symbols: you can reference them from any program, and you can use them within the Operator Interface and for DDE operations. I/O symbols can only be edited from the Configuration Utility; you cannot edit an I/O point from within the Symbol Manager. |
| Function blocks | Most function blocks need to be instantiated. To be used in Structured Text or Instruction List programs, an instance of the function block type must be explicitly declared in the Symbol Manager. |
| System objects (PID, PRGBC, and TMR) | These need to be instantiated. To be used in Structured Text, an instance of the system object type must be explicitly declared in the Symbol Manager. |

**Note:** Refer to the Language Reference for information on identifiers, data types, literals, type conversion, generic data types, pointers, and system keywords.

## User-Defined Data Type

For more sophisticated data handling, you can create your own structured data types. A structure can contain several members of different base types or user-defined structured types. Consider a user-defined structure named UserStructure01 having and integer type USInt member, a Boolean type USBool, and a string type USString. The individual members can be accessed in the following manner:

```
UserStructure01.USInt:=101;
UserStructure01.USBool:= TRUE;
UserStructure01.USString:="ABC";
```

User-defined types are valid anyplace that accepts an ANY or USER-DEFINED data type. Refer to **Editing User-Defined Data Structure** Types for more information.

## Arrays

To access a particular element of an array, enter the symbol name followed by square brackets with the number of the element you wish to access. For example to access the fifth element of an array symbol called Myarray you would type Myarray[5]. This is assuming you have defined the lower bound of the array as 1. You can also index into an array by placing a symbol name of type INT inside the square brackets.

## Pointer Symbols

Structured Text has two pointer operators: the pointer reference & operator and the pointer dereference * operator. These operators are used in indirect addressing operations.

In programming languages, data values are typically referred to by symbolic name. This is known as direct addressing. The data value is given a symbolic name and that symbolic name is used to directly access that data value.

```
X := Y;
```

The data value known as X is assigned the value of the Y data value.

Indirect addressing is a common paradigm in programming languages. When using indirect addressing, the symbolic name refers to the location where the data value is stored. These indirect symbols are commonly called pointer symbols. To get the actual data value using indirect addressing, the symbolic name of the pointer symbol is used to obtain the location of the data value, then the location of the data value is used to get the actual data value (an indirect operation).

If pVar1 is a pointer symbol, then in the following assignment, pVar1 is assigned the location of the X data value.

```
pVar1 := & X;
```

If pVar1 is a pointer symbol, then in the following assignment, Y is assigned the value contained in Var1, since pVar1 contains the location of Var1.

```
Y := * pVar1;
```

If pVar1 is a pointer symbol, then in the following assignment,Var1 is assigned the value contained in Y.

```
* pVar1 := Y;
```

Refer to the Language Reference for complete information on pointers and to Structured Text Programming for information on using pointers in structured text statements.

## Symbol Manager

The Symbol Manager is used to view, create, edit, copy, and delete variable and constant symbols, and function block instances. I/O points and system symbols can be only be viewed. The Symbol Manager displays a list of local symbols defined in the active file and global symbols defined in the active configuration.

Using drag and drop techniques, the Symbol Manager can be used to add symbols into Structured Text program statements, Instruction List program statements, or RLL program contacts and coils. You can also drag and drop symbol information to other OLE capable software programs, such as

Microsoft WordPad, Word, Excel and others. Symbols can also be selected from lists in various dialog boxes.

## Opening the Symbol Manager

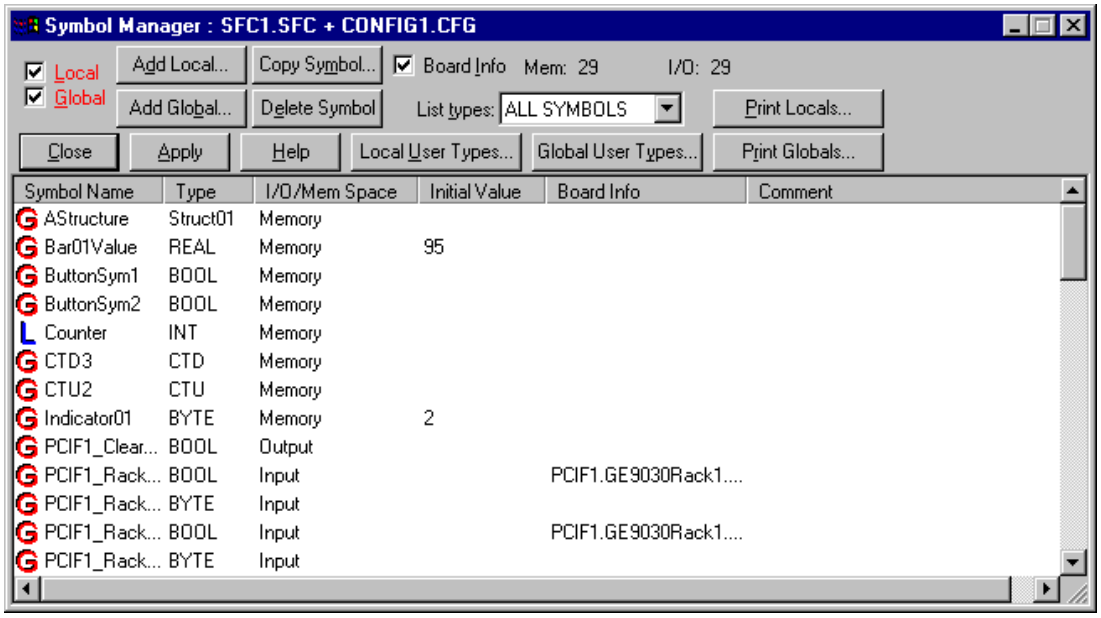
### To Open the Symbol Manager

- Select Symbol Manager from the Program Editor or Operator Interface Editor *Tools* menu.

**Notes:**

1. The Symbol Manager can also be opened from various dialog boxes in the editors.
2. You cannot have the Symbol Manager open in both the Program Editor and Operator Interface Editor.

The following figure shows an example of the Symbol Manager. The description of I/O and memory symbols is displayed in table format. In front of each symbol name is a G for global symbols or an L for local symbols. An asterisk (*) preceding the symbol name indicates it is a pointer symbol. If the symbol name corresponds to an I/O point, the board information is displayed in the appropriate column.



**Symbol Manager Controls**

| Control | Description |
|---|---|
| Local<br>Global | If checked, that symbol type is displayed in the symbol list. |
| Board Info | If checked, the board info is displayed (interface board, rack, etc.). |
| List Types | Filters the symbol list by data type and function block type. |
| Add Local | Create and add a local symbol. |
| Add Global | Create and add a global symbol. |
| Copy Symbol | Copy the currently selected symbol. |
| Delete Symbol | Delete the currently selected symbol. |
| Close | Close the Symbol Manager. |
| Apply | Makes symbol edits visible outside the Symbol Manager. |
| Help | Displays Symbol Manager help. |
| Print Locals | Prints a list of local symbols. |
| Print Globals | Prints a list of global symbols. |
| Local User Types | Creates a local user type. |
| Global User Types | Creates a global user type. |

## Creating a Symbol

### To create a new symbol

- Click either the *Add Local* or *Add Global* button, or use the context menu within the symbol list. The *Symbol Details* dialog box appears. Enter the appropriate information in the *Symbol Details* dialog box and click *OK*. The new symbol will appear in the symbol list.

If a program is not open, you can create only global symbols. If any program is open, you can create both local and global symbols. A local symbol can be used only by the program that is open (active) when you define the symbol.

> **Note:** When done editing symbols, be sure to click *Apply* to make the changes visible outside the Symbol Manager.

## Editing a Symbol

1. Select the symbol you want to edit and press Enter, or double-click on the symbol.
2. The *Symbol Details* dialog box appears.
3. Modify information and click *OK* to accept changes.

> **Note:** When done editing symbols, be sure to click *Apply* to make the changes visible outside the Symbol Manager.

## Copying a Symbol

You can quickly create a new symbol with the same properties as an existing symbol by using the copy command.

1. Select the symbol you want to copy.
2. Click the *Copy Symbol* button or use the context menu. The *Symbol Details* dialog box opens with all the parameters filled in exactly as in the selected symbol.
3. A new symbol name must be entered. Type a new symbol name, make any other needed changes, and click *OK*.

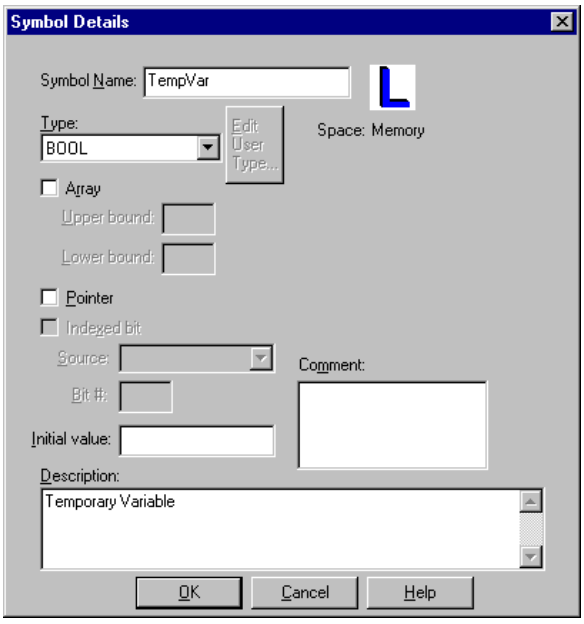
   **Note:** When done editing symbols, be sure to click *Apply* to make the changes visible outside the Symbol Manager.

## Deleting a Symbol

- Select the symbol you want to delete and click the *Delete Symbol* button or use the context menu.

  **Note:** When done editing symbols, be sure to click *Apply* to make the changes visible outside the Symbol Manager.

## Symbol Details



**Symbol Details**

| Field | Description |
|---|---|
| Symbol Name | Any valid identifier can be used for the symbol name. |
| Type | The elementary or user-defined structure data type, or function block type of the symbol. Refer to Error! Reference source not found. for more information. |
| Edit User Type | If the symbol is defined as a user type, this button is enabled and the user type can be edited. Refer to Editing User-Defined Data Structure Types for more information. |
| Array | If checked, defines the symbol as an array. Provide upper and lower bounds for the array. Refer to Arrays for more information. |
| Pointer | If checked, defines the symbol as a pointer. Refer to Pointer Symbols for more information. |
| Indexed Bit | The indexed bit feature lets you reference a specific bit within a byte, word, or double word symbol. Refer to Naming a Bit in a Symbol. |
| Initial Value | Used to provide an initial value for the symbol. A local symbol is reset to its initial value each time the program is run. A global symbol is assigned its initial value when the configuration is activated. |
| Comment | The comment is a short description that appears within the Symbol Manager. |
| Description | The description can be longer than the comment, but it does not appear in the Symbol Manager, only in the Symbol Details dialog box. |

## Naming a Bit in a Symbol

The indexed bit feature lets you reference a specific bit within a byte, word, or double word symbol. You can index a global symbol only with a global bit and a local symbol with a local bit.

1. Open the Symbol Manager and click the *Add Global* or *Add Local* button. The Symbol Details dialog box appears.
2. Enter the name of the bit in the *Symbol Name* field.
3. Select the BOOL data type in the *Type* field.
4. The *Indexed Bit* check box is only enabled if you have at least one symbol of type BYTE, WORD, or DWORD already defined. Select the *Indexed Bit* check box. The *Source* and *Bit* # fields become active.
5. The *Source* list shows all defined symbols of BYTE, WORD, and DWORD data types (for either Global or Local symbols). Select the symbol that you want to index in the *Source* field.
6. Enter the bit number in the *Bit* # field. Valid values are 0-7 for bytes, 0-15 for words, and 0-31 for double words.

7. Enter the optional description into the *Description* field.
8. Click on the *OK* button. The indexed bit appears in the *Symbol List* field.

## Editing User-Defined Data Structure Types

You can define your own stucture data types to organize data or to operate on the data as a group with certain operations (e.g., file operations). A user-defined structure data type consists of a group of data type members (integers, real numbers, strings, etc., or even other user-defined structure data types. Structure members do not have to be of the same data type. For example, you can define a structure data type called Report. It consists of three members called Year, Task, and ItemNumbers. Year and ItemNumbers are integers and Task is a string. You can then use a symbol of the Report data type in any program, and the data it contains consists of three members that represent a year, a task name, and a number.

User-defined structure data types can be global or local. A local data type can be used only by the  program that is open (active) when you define the data type.

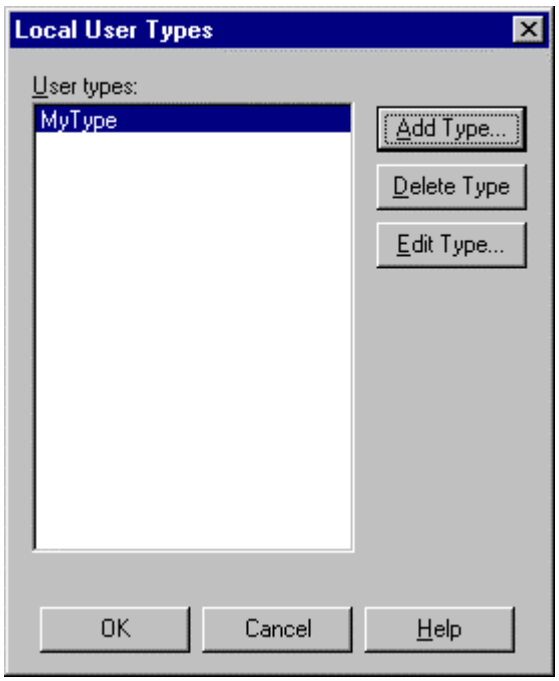
Once you have defined data structure types, they will be added to the Types list box in the Symbol Manager. You can now define one or more variable data structures of that data structure type (e.g. mystruct : struct1). You reference the members within your structure in an application program using dot notation. The format is "struct.member" (e.g. mystruct.x).

You can define a structure that contains other structures as members, but a structure cannot contain itself in any way.  For example if you defined structure type struct2 that contained a member a : struct1 and b : INT and a variable data structure c : struct2, then you can reference variables c.a.x(BOOL) and c.b(INT).

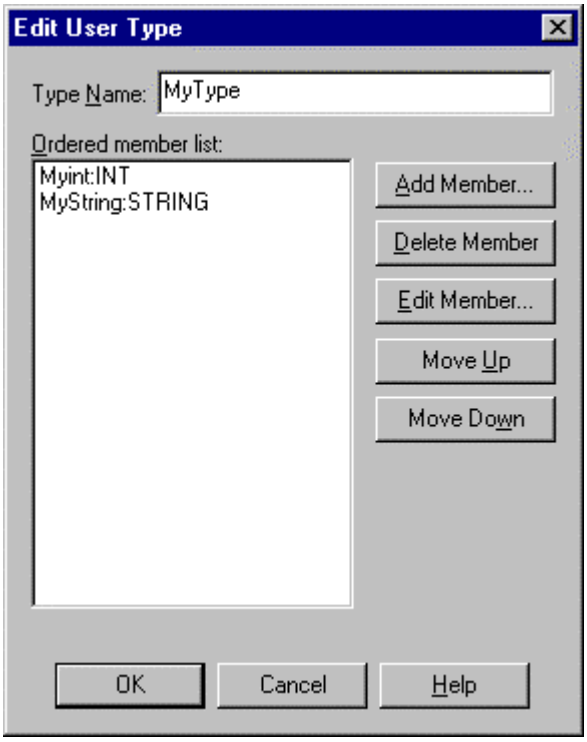
**Note:** The names of the structure members must be unique so that they will not conflict with other global or local variable names or structure members. You should try to create structure name in uppercase letter to follow the convention.

### To edit user-defined data types

1. Open the Symbol Manager
2. Click on *Local User Type* or *Global User Type*. The respective *User Types* dialog box appears. It lists defined user types (either local or global) and provides buttons to add, edit, and delete user types.




3. Do any of the following:
   - To create a new type, click *Add Type*. The *Edit User Type* dialog box appears.
   - To edit a user type, select the type name and click *Edit Type*. The *Edit User Type* dialog box appears.
   - To delete a user type, select the type name and click *Delete Type*.




**Edit User Type**

| Field | Description |
|---|---|
| Type Name | Displays the user type. If adding a user type, enter the type name. |
| Ordered member list | Displays a list of the type members. |
| Add Member | Adds a new member. Displays an *Edit Type Member* dialog box in which to assign a name and data type to the member. |
| Delete Member | Deletes the selected member. |
| Edit Member | Edits the selected member. Displays an *Edit Type Member* dialog box in which to assign a name and data type to the member. |
| Move Up<br>Move Down | Moves the selected member up or down in the ordered list. |

## Using Symbols

You can manually enter symbol names where needed by typing the symbol name. However, they can be more easily added using symbol list boxes that

appear (for example in the RLL editor), or dragging them from the Symbol Manager (into Structured Text or Instruction List documents).

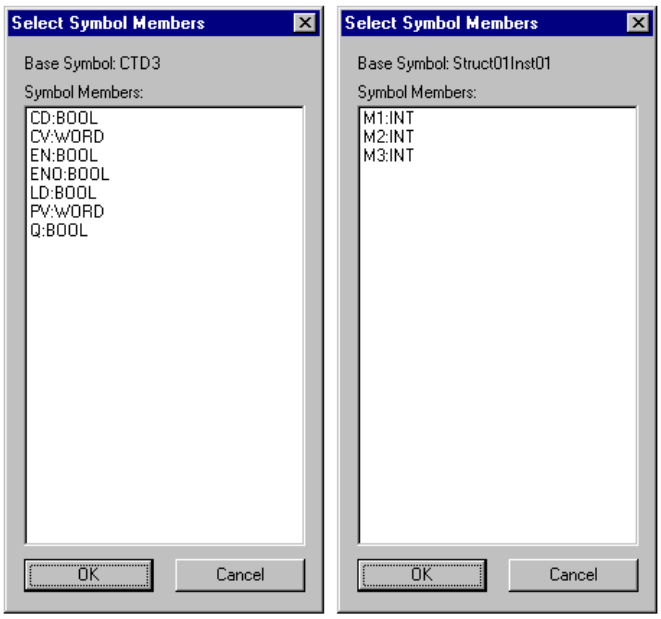
## Drag-and-Drop

Drag-and-drop features of the Symbol Manager allow you to do the following:

- You can drag a Boolean symbol onto an RLL rung. You are prompted to add the symbol as a contact or a coil. When you select contact or coil, the standard *Edit Contact* or *Edit Coil* dialog box appears.
- You can drag a Boolean symbol onto an existing RLL contact or coil. You are prompted to replace the existing contact or coil symbol with the one you have dragged from the Symbol Manager.
- You can drag symbols from the Symbol Manager directly into a location in a Structured Text or Instruction List document.

## Enumerations

For symbols that have member elements (structures, function blocks, and system objects), when a symbol of that type is dragged from the Symbol Manager into a Structured Text document (for example), the Select Symbol Members list appears. You can then drag-and-drop the member element you want to use from this list. This also works when adding symbols (that have member elements) to the Watch Window.



# System Symbols

## Predefined System Symbols

The system software automatically creates the following symbols that can be used with application programs.

| Symbol | Description |
|---|---|
| TODAY | Contains the current system date.<br>The TODAY system symbol is a DATE data type that contains the current system date and can be used to determine when an event takes place. The following operators can be used with the TODAY symbol: EQ, LT, GT, LE, GE, and NE. Use the assignment statement or MOVE command to define a value for TODAY. Use the ADD command to add a time duration to TODAY. |
| NOW | Contains the current system time.<br>The NOW system symbol is a TOD data type that contains the current system time and can be used to determine when an event takes place. The following operators can be used with the NOW symbol: EQ, LT, GT, LE, GE, and NE. Use the assignment statement or MOVE command to define a value for NOW. Use the ADD command to add a time duration to NOW. |
| NULL | Used to set a pointer symbol to a null value or to compare a pointer symbol (equal or not equal) to a null value. |
| TMR Variables | These variables contain status information for the TMR data type. |
| Counter Variables | These variables contain status information for RLL counters (CTD, CTU, and CTUD). These symbols are Local to the application program. |
| Timer Variables | These variables contain status information for RLL timers (TOF, TON, and TP). These symbols are Local to the application program. |
| "stepname".X | Contains the active/inactive status of an SFC step. These symbols are Local to the application program. |
| "stepname".T | Contains the elapsed execution time of an SFC step. These symbols are Local to the application program. |
| Motion Control | These variables contain status information for the axis, axis variables group, program control, and spindle |
| File Control Block | These variables contain status information for file operations. |
| Program Control | These variables contain the status information for the PRGCB data type. |

## Run-Time Symbols

The following symbols are automatically created by the system software. These symbols are accessible from the Symbol Manager or the Watch Window and can be used in user application programs.

| Symbol Name | Description |
|---|---|
| RT_ERROR | (INT) Math errors:<br>0 = no error<br>1 = divide by zero<br>2 = negative square root<br>RT_ERROR must be cleared by the user. |
| RT_FIRST_SCAN | (BOOL) set to TRUE on the first scan of the first program running in the ASIC run-time engine. After all programs are aborted, RT_FIRST_SCAN will be set again for the first scan of the first program to run. |
| RT_SCAN_OVERRUN | (BOOL) set to TRUE when I/O scan + logic scan exceed scan rate. |
| RT_MAX_SCAN | (REAL) duration in milliseconds of maximum run-time engine scan. |
| RT_LAST_SCAN | (REAL) duration in milliseconds of last runtime engine scan. |
| RT_AVG_SCAN | (REAL) duration in milliseconds of average runtime engine scan (runtime scan= logic + I/O + overhead). Rolling average calculated over the last 100 scans. |
| RT_LOGIC_MAX | (REAL) duration in milliseconds of maximum logic scan. |
| RT_LOGIC_LAST | (REAL) duration in milliseconds of last logic scan. |
| RT_LOGIC_AVG | (REAL) duration in milliseconds of average logic scan. Rolling average calculated over the last 100 scans. |
| RT_IO_MAX | (REAL) duration in milliseconds of maximum I/O scan. |
| RT_IO_LAST | (REAL) duration in milliseconds of last I/O scan. |
| RT_IO_AVG | (REAL) duration in milliseconds of average I/O scan. Rolling average calculated over the last 100 scans. |
| RT_MEM_PCT | (REAL) contains remaining percentage of system RAM (heap space) allocated for the programmable control system software. |
| RT_LOW_BATTERY | (BOOL) low battery signal from UPS. |
| RT_POWER_FAIL | (BOOL) power fail signal from UPS. |
| RT_SCAN_RATE | (REAL) configured scan rate of (in milliseconds) as set in the active configuration. |

# Application Programs

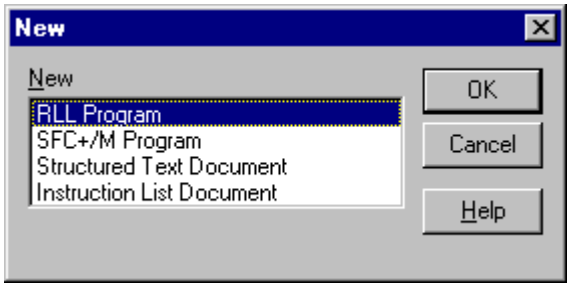
## Managing Application Programs

Program management lets you create and save Relay Ladder Logic (RLL), Sequential Function Chart (SFC), Structured Text (ST), and Instruction List (IL) programs. You can open many programs at the same time. The active program is displayed on the Program Editor title bar.

## Creating a New Program

### To create a new program in a new window

1. Select *New Editor* from the *File* menu or click the new program button  on the editor tool bar. A dialog box appears that lets you specify a new Relay Ladder Logic (RLL) program, Sequential Function Chart (SFC), Structured Text Document, or Instruction List Document.
2. Select the type of program you want to create and click *OK*. A new editor window of the appropriate type opens. The program is given a default name (*RLL1*, for example).



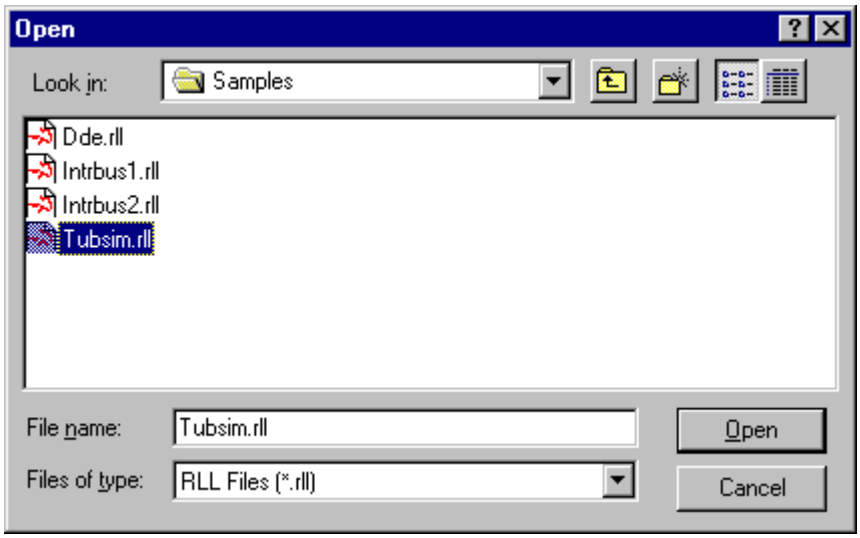

You can open many programs at the same time. Use the *Window* menu to switch between open programs or use the mouse to click on a partially visible program and bring that program's window to the top.

## Opening a Program

Each program is opened in its own editor window.

**To open an existing program**

1. Select *Open Editor* from the *File* menu or click the open program button  on the editor tool bar. An *Open* dialog box appears displaying the programs in the current project.
2. If no project is active when a file open command is executed, the new project or the open project dialog box is displayed.
3. From the *Files of type list*, select the type of program you want to open.
4. Select a file from the list box, then click *OK* to open it.




You can open many programs at the same time. Use the Window menu to switch between open programs, or use the mouse to click on a partially visible program and bring that program's window to the top.

## Viewing Programs

When a program is opened, the contents of the program are displayed in a window. You can open many windows of the same program. Open programs are listed under the *Window* menu.

**To access other open programs in Program Editor**

- Select the program from the list under the *Window* menu.

**Note:** You can also use the mouse to click on a partially visible program and bring that to the front.

Commands that are executed from the menus or by pushing buttons on the tool bars are performed on the active program.

**To view a new window of an open program**

- Click *Window* on the menu bar and select *New Window*.

## Saving a Program

**To save the active program**

- Select *Save* from the *File* menu or click the save program button  on the editor tool bar.

When you save a program for the first time, the Program Editor displays the *Save As* dialog box, so you can name your program. If you want to change the name or folder of the active program before you save it, use the *Save As* file command.

**To save a program with a new name**

3. Click *File* and select *Save As*.
4. The Program Editor displays the *Save As* dialog box so you can change the name of the program and folder.

**To save a program to a previous release format**

You can save a program to an earlier format to insure compatibility with a target runtime system that is of an earlier release than the development system, for example.

1. Click *File* and select *Save As*.
3. The Program Editor displays the *Save As* dialog box so you can change the name of the program and folder. Additionally, you can change the file type to ASIC Revision 4.01.

When you save a file to revision 4.01 (or any revision less than the current revision), any features in the current version that are not in the previous version you are saving to are permanently removed. Even if you open the file in a version you originally created it in, these features will not be present.

**Note:** When saving an SFC program with macros, only the main SFC (and its actions) is saved in the previous release format; if necessary, each macro must be opened and saved as the previous release format.

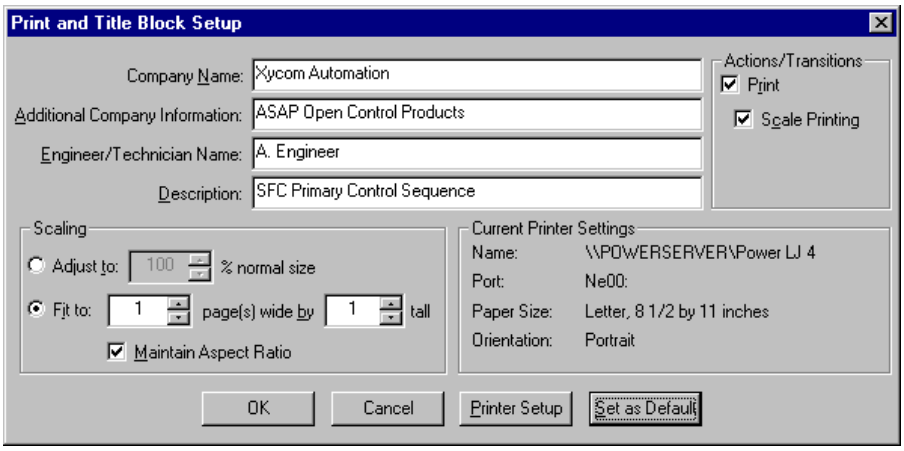
# Printing a Program

Programs can be printed as needed. Printing options allow you to add descriptive information to the printed sheet, view a print preview, and scale the diagram if necessary. Before printing RLL and SFC diagrams you should look at the print preview to see if there would be awkward page breaks and scale or adjust the diagram if possible. As a further aid, Print Boundaries can be displayed (a toggle item on the *View* menu) in normal view mode that show the edges of the printable page.

### To perform printer setup

- Focus on the program of interest and choose *Print Setup* from the *File* menu. The *Print and Title Block Setup* dialog box appears. Refer to the table for descriptions.

**Note:** The print setup of each program is saved with the diagram, meaning each can have its own print properties. However, you can also specify a print setup as the default - any newly created programs will use the default print setup until you edit it.




| Field | Description |
|---|---|
| Company Name<br>Additional Company Information<br>Engineer/ Technician<br>Description | Space is available at the bottom of each print out for this descriptive information. These fields can be edited to include any descriptive information you wish to include with the printed diagram. These fields are also displayed in the Print Preview. |

| Field | Description |
|---|---|
| Actions/ Transitions | Appears only for SFC diagrams. You can enable or disable printing of action and transition code as well as select whether or not they are scaled the same as the parent SFC diagram. They are printed on separate sheets. |
| Scaling | Percent scaling of the program. You can adjust this as needed or, by selecting *Fit to*, automatically scale a diagram to fit a specified number of pages. Scaling is always the percent reduction or magnification of the normal size. Fit to will never magnify a page. |
| Current Printer Settings | These are the settings from the *Printer Setup*. To change any of these, click *Printer Setup*. |
| OK | Accepts the Print and Title Block Setup and returns to the Program Editor. |
| Cancel | Cancels any Print and Title Block Setup changes and returns to the Program Editor. |
| Printer Setup | Displays the standard Windows printer setup dialog box. |
| Set as Default | Saves the current Print and Title Block Setup as the default. Any new programs will inherit these properties. |

**To view a print preview**

- Focus on the program of interest and choose *Print Preview* from the *File* menu or tool bar. The diagram window changes to a print preview of the diagram. You can print, browse through the pages (if more than one), toggle between one page/two page display mode, zoom in and out, and close the print preview.

Print preview shows exactly how the page will be printed, including the coordinates and print and title block setup fields.

**To print a program**

- Focus on the program of interest and choose *Print* from the *File* menu or tool bar (or from the print preview). The standard Windows print dialog box appears. Make any changes to the print options and click *OK* to print the program.

## Printing Program Cross-References

This lists symbol usage for a program. It lists all symbols in a configuration, the number of times they are used in the program, and their location in the program. For example, the rung number and contact or coil type for an RLL program.

**To print program cross-references**

- Select *Print Xref* from the *File* menu.

## Closing Application Programs

### To close a program

- Select *Close* from the *File* menu.

## Panning and Zooming

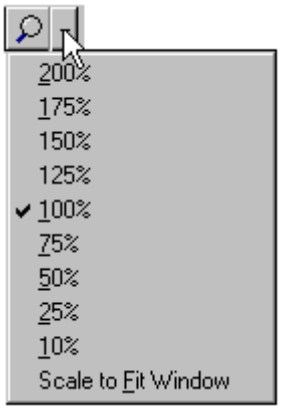
### To pan a diagram

- Use the horizontal and vertical scroll bars to bring the area of the diagram of interest into view.
- Alternatively, the Page Up and Page Down keys can be used to scroll vertically; the Home and End keys can be used to scroll horizontally.
-

### To magnify the a diagram, do any of the following

- Restore the zoom level to 100% by toggling *Zoom* from *View* menu off or use the magnifier icon.
- Choose *Zoom*, then one of the zoom percentages from the *View* menu, or use the zoom icon as shown in the figure.
- Click *Scale to Fit Window* from the *View* menu or zoom icon to scale the diagram to fit the work space area.
- Toggle between 100% zoom level and the last selected zoom level by selecting *Zoom* from the *View* menu or use the magnifier icon.

  **Note:** The zoom value appears in the status bar.




### To locate a program element using zooming

1. Zoom the program to fit the window.

2. Select the desired element.
3. Toggle zooming off. The selected element is positioned as close to the center of the window as possible.

## Turning Comments On and Off

Descriptive comments can be added to application programs to document functions of the program. RLL and SFC program comments can be displayed or hidden. The *View Comments* toggle affects all open programs.

**To toggle program comments**

- Select *Program Comments* from the *View* menu or click *View Comments* button on the tool bar.

  **Note:** A check mark next to *Program Elements* and a depressed *View Comments* button indicates that comments are visible.

## Selecting Program Elements in SFC and RLL Editors

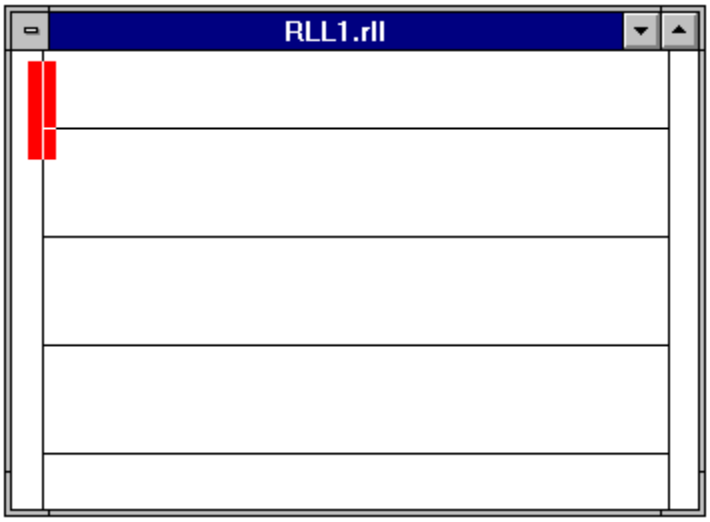
- You can use area selection to select program elements by positioning the cursor at one corner of a rectangular area that will be drawn to select the elements, then clicking-and-dragging to the other corner and releasing the mouse button. All program elements completely within the rectangular area are selected.
- While using area selection, auto-scrolling is enabled. That is the view boundaries are repositioned as if you were using the scroll bars.
- After selecting one or more elements, you can reposition them by dragging to another part of the diagram. If you drag past the view boundaries in any direction, auto-scrolling is enabled.
- If you position the cursor outside the view boundary while using area selection or element dragging, after 5 line scrolls, all further scrolls will be page scrolls. Scrolling in the opposite direction or returning the cursor inside the view boundaries resets to normal scrolling.

# RLL Programming

## About Structuring RLL Application Programs

Relay Ladder Logic (RLL) diagrams are commonly used on programmable controllers to construct continuous logic programs. RLL diagrams are designed to resemble the electrical diagram for an equivalent electrical relay logic circuit.

The RLL diagram contains two vertical power rails. The left power rail is assumed to be an electrical current source and is energized whenever the RLL program is running. The power rail on the right is assumed to be an electrical current sink. The two power rails are connected by horizontal lines called rungs (like the rungs of a ladder) on which the logical instructions are placed.




The basic RLL program instructions, contact and coils, represent actual hardware components (limit switches, solenoid coils, lights, etc.) and single-bit internal memory locations.

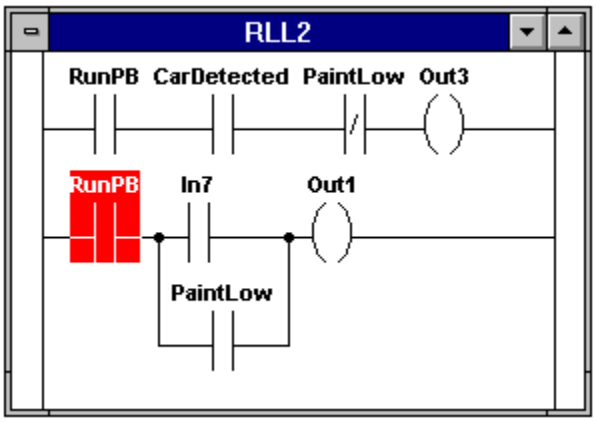
RLL diagrams use input contacts to represent Boolean input symbols (variables). These contacts act as:

- Normally open (active high), which means that current passes to or energizes the RLL element to its right when the symbol associated with it is high (1 or on).
- Normally closed (active low) relay contacts, which means that current passes to or energizes the RLL element to its right when the symbol associated with it is low (0 or off).

These inputs usually have the name of the Boolean symbol they represent located above the graphical element. They are variations of the input contact, including transition sensing contacts that pass current to the next element for only one scan of the RLL program.

RLL diagrams use output coils to represent Boolean output variables. If the logic to the left of an output coil is energized (set true), the Boolean symbol represented by the output coil receives a Boolean 1 (high); otherwise, it receives a Boolean 0 (low). Variations of the coil element exist, including inverting and pulsed (single scan outputs. Some output coils are latching and only set (or reset) the output variable to true (or false) if the logic to the left is true.

RLL logic elements that reside on the same horizontal rung are assumed to be logically ANDed together. Function blocks are provided to facilitate complex operations that use more than one Boolean symbol.



# Relay Ladder Logic Instructions

Relay Ladder Logic consists of:

- "Relay Contacts" on page 41
- "Relay Coils" on page 42
- Function blocks, refer to the Language Reference.

## Relay Contacts

The contact typically represents a discrete input point, such as a limit switch. A contact can also represent an internal memory location, and as such, it is termed a Boolean. The contact can have one of two states: TRUE or FALSE. You refer to the contact in your program and the object that it represents by its symbolic name, which you assign in the Symbol Manager. You can assign the same symbolic name to a contact and a coil, and the output from one rung can serve as an input to another rung.

### *Normally Open Contacts*

The normally open contact operates as follows:

- The contact passes power flow if the point that it represents is TRUE.
- The contact does not pass power flow if the point that it represents is FALSE.

### *Normally Closed Contacts*

The normally closed contact operates as follows:

- The contact passes power flow if the point that it represents is FALSE.
- The contact does not pass power flow if the point that it represents is TRUE.

### *Positive Transition Sensing Contact*


The positive transition sensing contact operates as follows:

- The contact passes power flow if the point that it represents transitions from FALSE to TRUE immediately prior to the evaluation of the contact. Otherwise, the contact does not pass power flow.
- The contact cannot pass power flow again until the point that it represents transitions from FALSE to TRUE again.

### *Negative Transition Sensing Contact*

The negative transition sensing contact operates as follows:

- The contact passes power flow if the point that it represents transitions from TRUE to FALSE immediately prior to the evaluation of the contact. Otherwise, the contact does not pass power flow.
- The contact cannot pass power flow again until the point that it represents transitions from TRUE to FALSE again.

## Relay Coils

The coil typically represents a discrete output point, such as a solenoid. A coil can also represent an internal memory location, and as such, it is termed a Boolean. You refer to the coil in your program and the object that it represents by its symbolic name, which you assign in the Symbol Manager. You can assign the same symbolic name to a contact and a coil, and the output from one rung can serve as an input to another rung. You can place multiple coils on a single rung, and the status of one coil is not affected by the status of the others.

### *Output Coils*

The output coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil has power flow.
- The coil sets the point that it represents to FALSE if the coil does not have power flow.

### *Negated Output Coils*

The negated output coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil does not have power flow.
- The coil sets the point that it represents to FALSE if the coil has power flow.

### *Set (Latch) Coil*


The set (latch) coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil has power flow.
- The point continues to be TRUE (the point is set, or latched) even when the coil no longer has power flow.
- The point can be set to FALSE only by a reset coil.

### *Reset (Unlatch) Coil*


The reset (unlatch) coil operates as follows:

- The coil sets the point that it represents to FALSE when the coil has power flow.
- The point remains FALSE (the object is reset, or unlatched) even when the coil no longer has power flow.
- The point can be set to FALSE only by a reset coil.

### *Positive Transition Sensing Coil*


The positive transition sensing coil operates as follows:

- When power flow to the coil transitions from FALSE to TRUE, the coil sets the point that it represents to TRUE.
- The point remains TRUE, unless it is set to FALSE elsewhere, for the duration of one scan cycle.

### *Negative Transition Sensing Coil*


The negative transition sensing coil operates as follows:

- When power flow to the coil transitions from TRUE to FALSE, the coil sets the point that it represents to TRUE.
- The object remains TRUE, unless it is set to FALSE else where, for the duration of one scan cycle.

### *Jump Coil/Label*


Use the jump and label elements to disable sections of program code temporarily. You must use the jump coil and label together. A label without a jump coil causes a runtime error. The jump coil and label operate as follows:

- When a jump coil receives power flow, program execution ignores all logic between the jump coil and its corresponding label.
- When a jump coil is actively skipping logic, outputs between the jump coil and the label are not activated.
- All logic between a jump coil and label is executed normally when the jump coil does not receive power flow.
- Program execution cannot jump backwards to a previous rung.

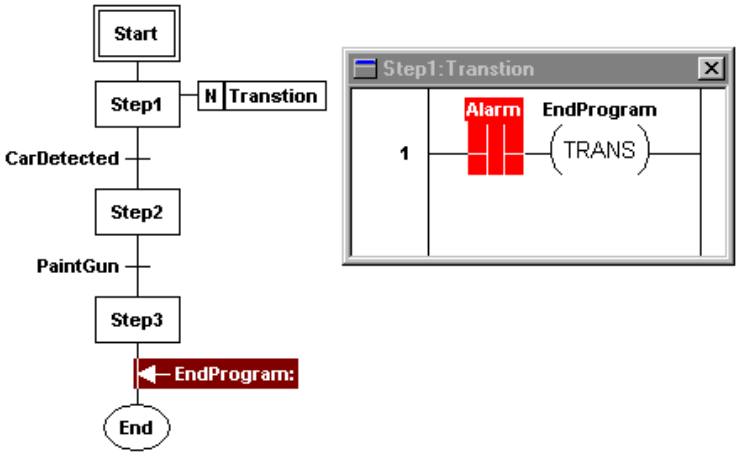
  **Note:** If you do not close a jump coil with a label, power flow jumps to the end of the program and does not execute any code following the jump coil.

### *SFC Transition Coil*


The SFC transition coil is an RLL program element that you can use only under specific conditions (within an SFC Action) in an SFC program. The SFC transition coil, which executes similarly to the jump element described above, operates as follows.

You use the SFC transition coil in an RLL Action associated with a Step or a Macro Step.

- Associated with a Step — When the SFC transition coil receives power flow, the Structured Text within the Step is cancelled, and program execution jumps to the SFC label specified in the SFC transition coil.
- Associated with a Macro Step — When the SFC transition coil receives power flow, the child SFC called by the macro step is cancelled and program execution in the parent SFC (the SFC with the macro step) jumps to the SFC label specified in the SFC transition coil.

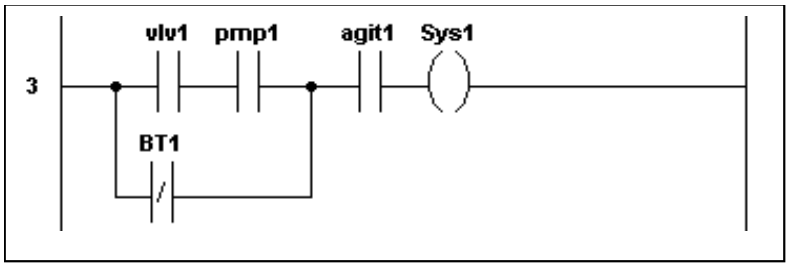



# How RLL Application Programs are Solved

This topic explains how RLL programs using simple relays and function block are solved.

## How Relay Logic is Solved

After the system writes to the physical outputs, it reads physical inputs and then solves the RLL logic. Power flow and solving of the program logic is always from top to bottom and from left to right. In the following example, power flow begins at the left power rail. If contact vlv1 is on, power flow continues to contact pmp1. The three contacts, vlv1, pmp1, and agit1 are in a series and represent the logical ANDing of the three contacts. Contact bt1 is in parallel with contacts vlv1 and pmp1, representing the logical OR of bt1 with vlv1 and pmp1. If contact bt1 is on, power flow continues to agit1, even if vlv1 is not on. If power flow continues to coil sys1, then it turns on the circuit, completing power flow to the right rail.

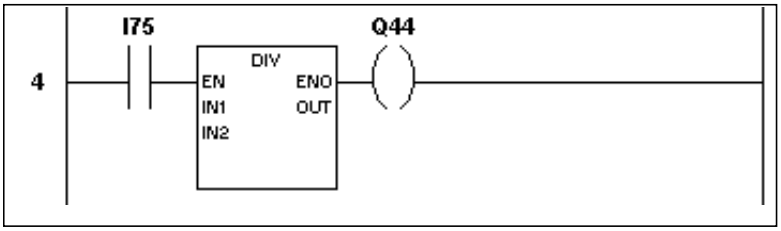


### How Function Blocks Are Solved

Function blocks provide a mechanism for solving more complex operations not easily handled by contacts and coils, such as:

- math operations
- logic functions
- counters and timers, etc.

Function block inputs receive power flow from the rung and transfer power flow to the next element on the rung through their outputs. They can also read and write data from their internal inputs and outputs. In the following example, the division function block (DIV) is enabled by its rung input EN, which receives power flow when contact I75 is enabled. It receives divisor and dividend data through two internal inputs (IN1, IN2). The function block writes the quotient to an internal output (OUT) and transfers power flow to the next program element, Q44, through its rung output (ENO).




## Extensions to IEC 1131-3 Ladder Diagrams

Extensions to the IEC-1131-3 standard Ladder Diagram language include:

- Output Coils can be placed anywhere on a rung, and:
  - store the partial Boolean result evaluated up to that point
  - pass power on to the next element to the right
- You can use "OR" branches that do not contain any logical elements to:
  - shunt around a block of logic without deleting it
  - generate compiler warnings that signal possible omission of elements

## Creating Relay Ladder Logic Programs

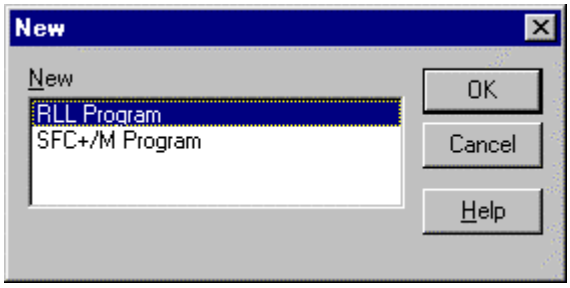
This section provides information about:


- "Creating an RLL Program" on page 47
- "Adding Program Elements" on page 47
- "Moving and Editing Program Elements" on page 54
- "Documenting RLL Application Programs" on page 56

## Creating an RLL Program

To open the RLL editor and begin creating a program, you need to enter the Program Development environment.

1. Start the Program Editor.
2. Click on *File* and select *New Editor.* The *New* dialog box appears.



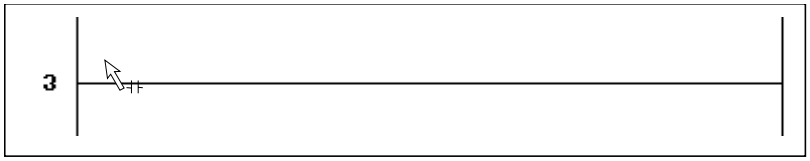

3. Select *RLL Program* and click on *OK*.
4. The RLL editor displays a new RLL file with the two power rails and one rung.

## Adding Program Elements

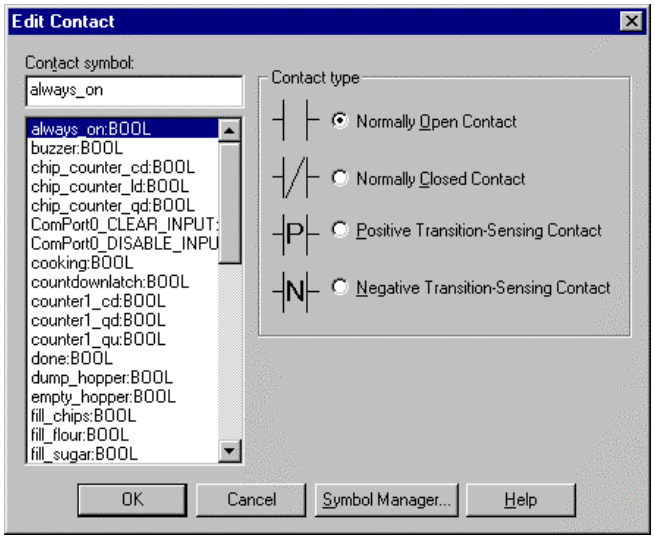
Use this section to help you build an RLL program.

### *Adding a Contact*

1. Click on the *Contact Tool* on the *RLL toolbar*. The cursor changes into the contact cursor.
2. Move the cursor to the location on the rung where you want to place the new contact.



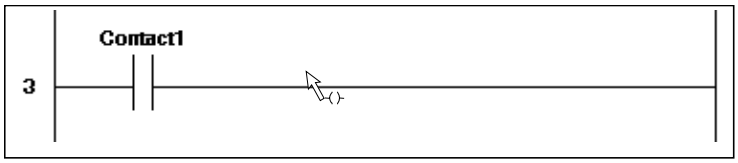

3. Click the left mouse button. The *Edit Contact* dialog box appears.



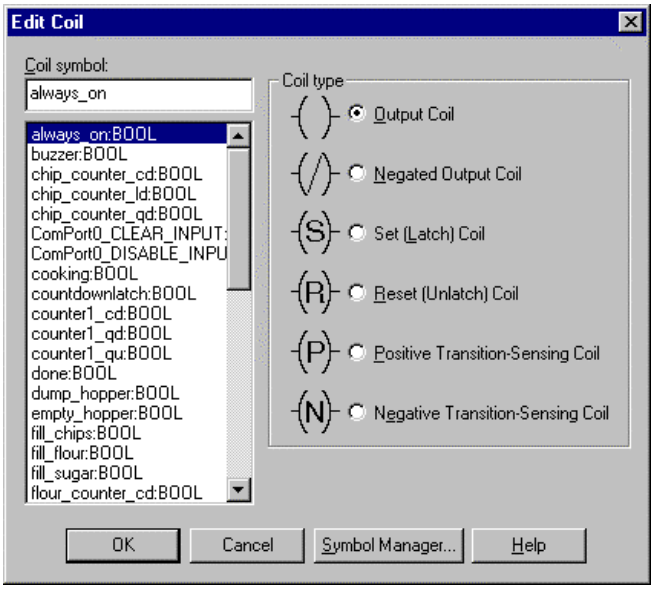

4. If you have already defined the symbol names for your system, click on the symbol to represent the contact (heaters_bldg_2 in the figure).
5. If you have not defined the symbol names, or if you want to define a new symbol, you need to access the Symbol Manager and fill in the symbol data for the contact. Refer to the "Defining Symbols" chapter for more information about defining symbols.
6. Select the contact type (Normally Open, Normally Closed, etc.) and then click on *OK*. The new contact appears on the rung at the location you specified.

### *Adding a Coil*

1. Click on the *Coil Tool* on the *RLL toolbar*. The cursor changes into the coil cursor.
2. Move the cursor to the location on the rung where you want to place the new coil.




3. Click the left mouse button. The *Edit Coil* dialog box appears.

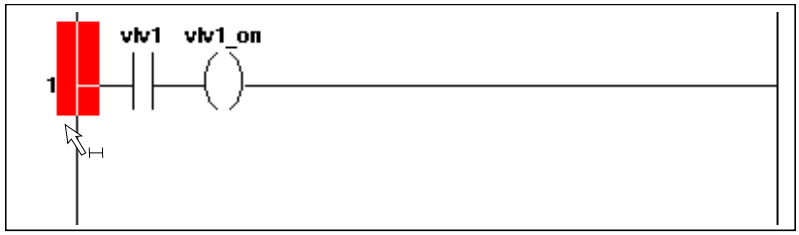



4. If you have already defined the symbol names for your system, click on the symbol to represent the coil.
5. If you have not defined the symbol names, or if you want to define a new symbol, you need to access the Symbol Manager and fill in the symbol data for the coil.
6. Select the coil type (Output, Negated Output, etc.) and then click on *OK*. The new coil appears on the rung at the location you specified.

You can place an Output Coil anywhere on a rung, including to the left of an input coil or within an OR branch. An Output coil stores the logical result of the logic evaluated up to its location on the rung.

### *Inserting a New Rung*

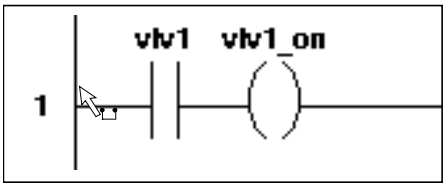
1. Click the *Rung Tool*. A rung symbol attaches to the mouse pointer.
2. Move the cursor to the location on the left power rail where you want to insert the new rung.



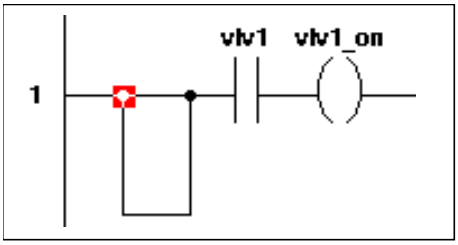
3. Click the left mouse button. The editor inserts the rung at the specified location.

### *Adding a Branch*

1. Click on the *Branch Tool* on the *RLL toolbar*. The cursor changes into the Branch Tool cursor.
2. Move the cursor to the location on the rung where you want to insert the branch.



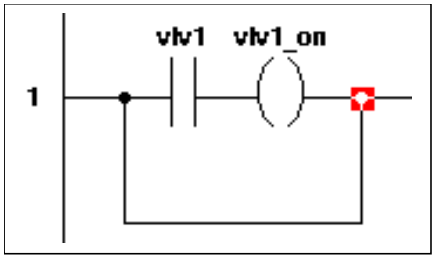

3. Click the left mouse button. The editor inserts the branch at the specified location.




4. After inserting the branch, you can adjust the contact points as necessary.

#### To move the contact points of a branch

1. Click on the *Select Tool*  .
2. Move the cursor over the contact point that you want to move and press the left mouse button.
3. With the mouse button depressed, move the cursor and the contact to the new location on the rung and release the button. The editor connects the branch at the new location on the rung.

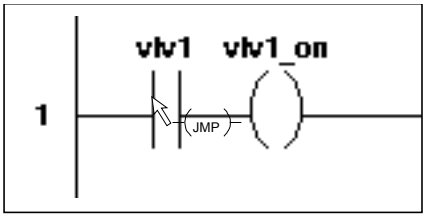


You can insert an branch that contains no logic (a shunt). You can use a shunt to temporarily disable a section of logic without deleting it from the program. Used with a contact that turns off power flow to the logic in question, the shunt maintains power flow across the rest of the rung. This feature is useful for debugging your program.

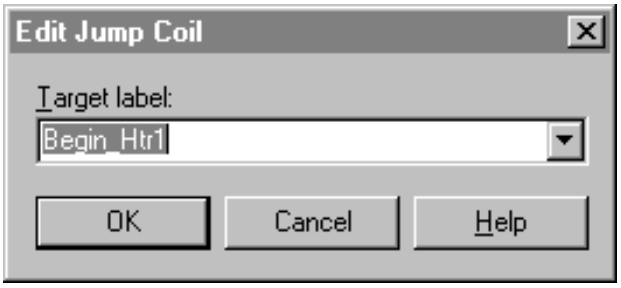
You can also move a contact point from rung to rung without deleting the logic contained within the branch.

### *Adding a Jump Coil*

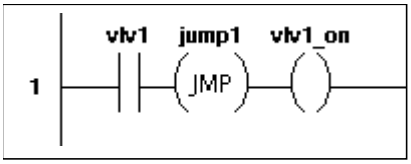
1. Click on the *Jump Tool* on the *RLL toolbar*. The cursor changes into the Jump cursor.
2. Move the cursor to the location on the rung where you want to place the new jump.




3. Click the left mouse button. The *Edit Jump Coil* dialog box appears.




4. Enter a target label and click on *OK*. A target label name cannot contain any spaces. The editor inserts the jump at the specified location.



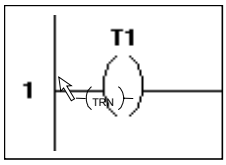

### *Adding an SFC Transition Coil*

The SFC transition coil is an RLL program element that you can use only under specific conditions (within an SFC action) in an SFC program. To add

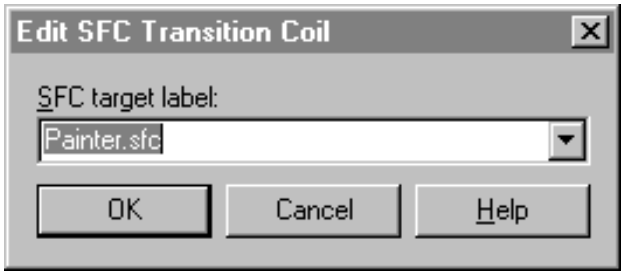
an SFC transition coil to the program, you must be editing an action in an SFC program.

### To add an SFC Transition Coil

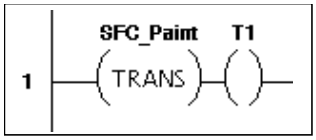
1. Click on the *SFC Transition Coil* tool on the RLL toolbar. The cursor changes into the SFC Transition Coil cursor.
2. Move the cursor to the location on the rung where you want to place the SFC Transition Coil.




3. Click the left mouse button. The *Edit SFC Transition Coil* dialog box appears.



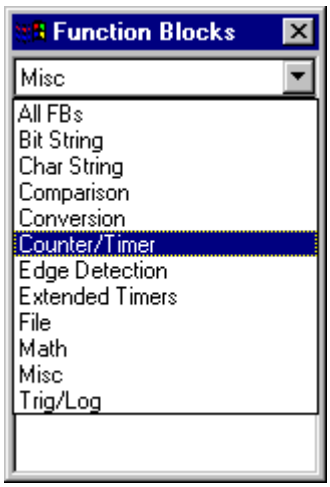

4. Type the name of the SFC target label and click *OK*. The editor inserts the SFC Transition Coil at the specified location.



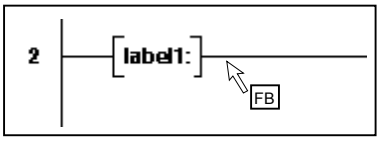

## *Adding Function Blocks*

### To Add a Function Block

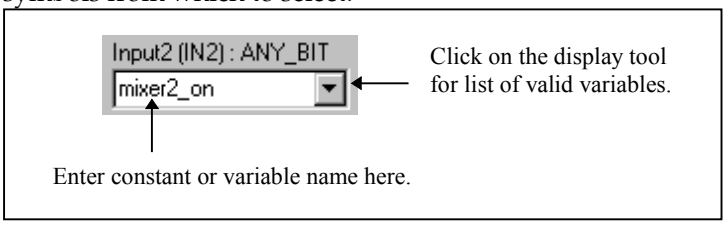
If the Function Block palette is not being displayed, select *View/Function Block Palette* from the menu bar. The editor displays the Function Block Palette.



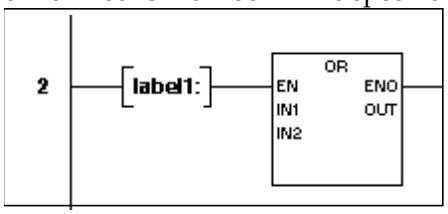

1. Click on the *Display tool*  to display a list of the function block types.
2. Click on the type of function block that you need. The Function Block Palette changes to display the selected function block types.
3. Click on the specific function block that you want to add. The cursor changes into the function block cursor.
4. Move the cursor to the location on the rung where you want to place the function block.




5. Click the left mouse button. The dialog box for the block appears.
6. Fill in the appropriate information for the function block.
7. To enter a constant (an integer, real number, characters of a string, etc.) type the value directly into the field. To enter a symbol, either type the symbol name into the field, or click on the display tool for a list of valid symbols from which to select.



8. When you have finished filling out the dialog box, click on *OK*. The editor inserts the block at the specified location.




## Moving and Editing Program Elements

This section explains how to select, edit, and move RLL elements.

### *Selecting Program Elements*

#### To select elements with the keyboard

Use the arrow keys to select the next element in the desired direction. A beep will sound if the selection cannot be made.

#### To select elements with the mouse

1. Select the Selector tool from the RLL Tool Bar or the SFC Tool Bar.

2. Move the cursor to the desired element and press the left mouse button. The element will highlight in the select highlight colors.

**Notes:**

1. To select a select diverge or simultaneous diverge in a SFC program, select the top or bottom bar of the diverge.
2. To select a rung in a SFC program, select the left or right power rail of the rung.

#### To select multiple elements at one time

1. Select the Selector tool from the RLL Tool Bar or the SFC Tool Bar.

2. Move the cursor to the top and left of the top, leftmost element in the desired group (be sure the cursor is not over any element).
3. Press and hold the left mouse button and drag the cursor. As the cursor is dragged, a selection box will appear.
4. When the desired elements are in the selection box, release the left mouse button. All elements entirely inside of the selection box will be selected.

**Notes:**

1. To select a branch connector in a RLL program, both connectors and all elements on the branch must be inside of the selection box.
2. To select a rung in a RLL program, the left and right power rails and all elements on the rung must be inside of the selection box.
3. To select a loop in a SFC program, the topmost loop arrow, all loop transitions and all elements contained in the loop must be inside of the selection box.
4. To select a jump in a SFC program, the jump diamond, all transitions and all target labels must be inside of the selection box.
5. To select a select diverge or a simultaneous diverge in a SFC program, the top and bottom bars and all elements in the diverge must be inside of the selection box.

### *Moving a Branch*

1. Select the Selector tool from the RLL Tool Bar or the SFC Tool Bar.

2. Move the cursor over the desired branch connector. Press and hold the left mouse button.
3. Drag the branch connector to the desired location (the cursor will change to a branch connector cursor).
4. Release the left mouse button to drop the branch connector at the desired location.

- To cancel the drag operation, press the Esc key.
- If the target location for the branch connector is on the same rung and not inside of any embedded branches or outside of any nested branches, the dragged branch connector is located at the drop position and the other end of the branch remains in its current position. However, if the target location for the branch connector is on a different rung, inside of an embedded branch or outside of a nested branch, both ends of the branch are moved to the target position (the entire branch will be moved).

### *Moving Program Elements*

1. Select the desired element(s) and move the cursor over one of the selected elements.
2. Press and hold the left mouse button and drag the selected elements to the desired location. When dragging begins the elements being dragged will be blanked out and the cursor will change to represent the element being dragged. If multiple elements are being dragged the cursor will be

changed to the group drag cursor. Release the left mouse button to drop the element(s) at the desired location.

- To cancel the drag operation, press the Esc key.

### *Editing Program Elements*

1. Click on the *Select Tool*.
2. Double click on the element (Step, Transition, label, etc.). The dialog box appropriate for the element (*Edit Step*, *Select RLL Transition Logic*, *Bypass Jump Transition Logic*, etc.) appears.
3. Make changes in the dialog box as needed.

### *Deleting a Branch*

The operation of the Cut tool is based on you selecting an object and then clicking on the Cut tool to delete the object. To delete the branch, however, follow one of these procedures.

**To delete a Branch with out any elements**

1. Click on the *Select Tool*.
2. Place the cursor in the middle of the branch and click.
3. Click on the Cut tool to delete the branch.

**To delete a Branch that contains one or more elements**

1. Click on the *Select Tool*.
2. Drag an area that includes the entire branch and its connection points.
3. Click on the Cut tool to delete the branch.

### *Undoing/Redoing Edits*

Click the *Undo* button on the *Tool bar* to undo an operation.

## Documenting RLL Application Programs

A program comment can consist of any meaningful description that you want to display on a Rung. You can choose whether the system displays the comments or hides them.

### *Adding and Editing Rung Comments*

1. Click on the *Comment Tool* on the RLL menu bar.
   
   When no comment is associated with a rung, the comment displayed is *(* Rung Comment *)*.
2. Move the cursor to the Rung in the program where you want to edit the comment and double click. The *Program Comments* dialog box appears.

3. Enter the comment and click on OK. The comment appears on the Rung specified.

### *Adding and Editing Symbol Descriptions*

1. Open an RLL or SFC program.
2. Click on Symbol Manager Tool.

   The Symbol Manager dialog box for local/global symbols appears.

Add or change information as desired.

# SFC Programming

## About Structuring Sequential Function Charts

A Sequential Function Chart represents an application program as a series of sequential **steps**. You associate control logic with these steps called **actions**. The logic in the action executes when the step becomes active.

Steps are connected by links and control is passed between steps via transitions. Transitions can be a Boolean expression or a single RLL rung.

You can manage multiple control paths using **divergences**. A select divergence lets you choose one path to be active from two or more control paths; whereas, a simultaneous divergence lets you activate multiple control paths simultaneously in parallel.

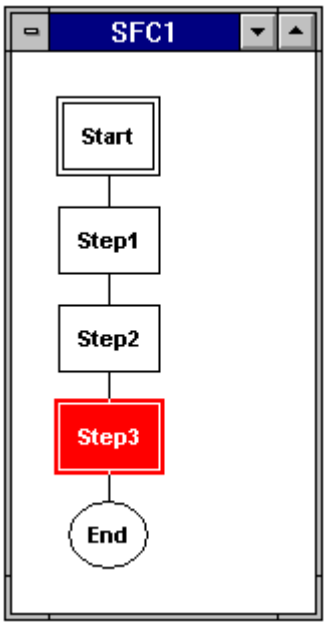
Other program flow capabilities exist. You can include control loops that let you repeat a series of steps or transfer control flow to another location using a jump-and-label structure.

This section describes these SFC components:



### About Steps

A step represents a condition in which the behavior of the system follows a set of rules defined by the actions and functions associated with the step. While the SFC is being executed, a step is either active or inactive. At any given moment, the state of the factory process is defined by the set of active steps and the values of its internal and output variables. The figure below shows a series of steps.




A step is graphically represented within an SFC as a box containing the step name, which identifies the step. Program flow into and out of the step is through a vertical line entering the top of the box and another line exiting from the bottom of the box.

When you create a new SFC, the system automatically generates the:

- first step, labeled start,
- last step, labeled end.

You cannot edit these steps; they simply represent the initiation and termination of the SFC.

Typically, you separate steps in an SFC with transitions, which are program elements. As an enhancement to the IEC-1131-3 specification, the control software lets you place a step immediately before or after another step, at runtime the system inserts the required "dummy" transition for you. For details about transitions, see "About Transitions" on page 67.

SFC steps can have one of more actions attached to them. An action can contain one or more rungs of ladder logic and uses special action qualifiers to control the execution of the action. For details about actions, see "About Actions" on page 62.

You can create SFCs with multiple paths, and it is possible for more than one SFC to be active at a time. Divergences help you manage multiple control paths. See "About Divergences" on page 68.

From within a single step you can call another entire SFC (the child) for execution—a macro step. When the child SFC has completed, program

control returns to the macro step that made the call. For information about macro steps, see "About Macro Steps" on page 70.

### *Step Parameters*

To create a step, you must configure step parameters in the edit step dialog box. This table summarizes the parameters. For more information, see Working with Steps on page 81.

| Field/Button | Description |
|---|---|
| Motion/Process Commands | Contains the program code for the Step. |
| RS-274D | Selects the RS-274D programming language. |
| Structured Text | Selects the Structured Text programming language. |
| Link File | Links the Step to a file containing the program code |
| Edit Linked File | Opens the linked file of program code for the Step and displays it in an editor. |
| Symbol Manager | Accesses the Symbol Manager. |
| **Display** (Click on one.) | |
| Step Name | Displays the Step name within the Step. |
| Motion/Process Commands | Displays the code (motion control or structured text) within the Step. |
| Step Description | Displays the Step description within the Step. |
| Icon | Displays an icon within the Step. |
| Width | Sets the width in pixels for the Step description. |
| Icon... | Accesses the Icon palette. |
| Remove | Deletes an icon from the Step, if one is assigned. |

### *Using the Step System Symbols*

Each SFC step has two system symbols (.X and .T). You can use these system symbols in any expression, contact or coil instead of a symbol of the same type. Reference the system symbols by typing the step name followed by a period and the symbol suffix.

| Symbol | Definition | Example |
|---|---|---|
| X | Boolean step-is-active Step active symbol X is TRUE when the step is active and FALSE when the step is inactive. | STEP1.X<br>Refers to the step active symbol for step STEP1. |
| T | Step time. Step time symbol T contains the current elapsed time of the step in milliseconds. When a step is inactive, T contains the total elapsed time of the step. T is set to zero when the Step becomes active. | STEP1.T<br>Refers to the step time symbol for step STEP1. |

## About Actions

Actions contain Relay Ladder Logic. Actions are graphically represented as a rectangular box containing the action's name. This box is connected to the step with a horizontal line. You can associate more than one action to a step.

Actions associated with a step are invoked when the step becomes active. You can specify when an action is executed relative to when the step becomes active by using an action qualifier. The qualifiers are:

- Action qualifier—determines when the RLL runs relative to the activation of the step, which can be either Structured Text or Motion Control language. You can use an action qualifier with or without a motion qualifier.
- Motion qualifier—determines when the RLL runs relative to the execution of the Motion Control code within the step.

Using a program label can also affect when an action is executed. The action does not run until the label in the step code is encountered.

The action qualifier is shown as a box containing an abbreviation attached to the right side of the action.

### *Action Function*

In the example below, the action called PaintColorAction consists of several rungs of RLL that are executed when Step2 becomes active. In this particular example, the RLL execution does not begin until code execution in the step encounters the label called label_a. The P code is the action qualifier and means that the RLL is executed one time, i.e., it is pulsed.


You can associate zero or more actions with a step, and you can associate one action with more than one step by referencing the action's name.

The SFC transition coil used in an action associated with a step or a macro step provides the following program flow controls:

- Associated with a step—When the SFC transition coil receives power flow, the Structured Text within the step is cancelled, and program execution jumps to the SFC label specified in the SFC transition coil.
- Associated with a macro step—When the SFC transition coil receives power flow, the child SFC called by the macro step is cancelled, and program execution in the parent SFC (the SFC with the macro step) jumps to the SFC label specified in the SFC transition coil.

### *Action Parameters*

To create an action, you must configure action parameters in the *Edit Action* dialog box. This table summarizes the parameters. For more information, see "Working with Actions" on page 87.

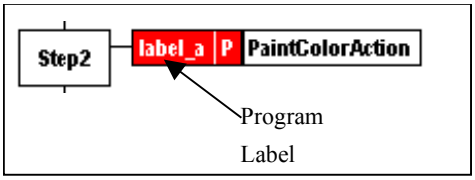
| Field/Button | Description |
|---|---|
| Program Label | (Optional) The RLL code does not begin running until the code in the step encounters the label. For more information, see "Program Label" on page 63. |
| Action Qualifier | Specifies an action qualifier. For more information, see "Action Qualifier" on page 64. |
| Motion Qualifier | Specifies a motion qualifier. Select None for no qualifier. For more information, see "Motion Qualifier" on page 64 |
| Time Duration | (Action qualifier only) Specifies the time duration for Limited and Delay qualifiers. If the action qualifier does not actually use the time duration, any value entered for this parameter is ignored. For more information, see "Time Duration" on page 66. |
| Specify Duration | Accesses the Define Time Duration dialog box. |
| Action Name | Specifies the name of the Action. For more information, see "Action Name" on page 66. |

### *Program Label*

If you specify the optional program label, the RLL code does not begin running until the code in the Step encounters the label. The action and the program label must be in the same SFC. No cross-reference between programs is allowed. If no label is in the step with which the action is associated, then the program label parameter is ignored.

The labels in a macro step SFC cannot be referenced from the parent SFC, and the labels in the parent SFC cannot be referenced by the macro SFC.

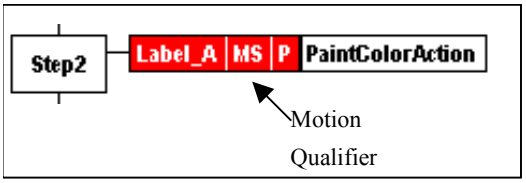
For a step with Structured Text, the label consists of a label name followed by two colons, e.g., LabelA::. For a step with Motion Control code, the label is an N followed by a block number, e.g., N45. If you enter a label, it appears within the action as shown below.



### *Motion Qualifier*

> **Note:** Not all motion qualifiers are supported by all motion cards. Refer to the motion card documentation for supported motion qualifiers.

Motion Qualifiers specify motion constraints that must be satisfied before the RLL begins running. If you use a program Label, the motion constraint applies to the motion block following the program Label. Qualifiers appear within the Action as shown below.

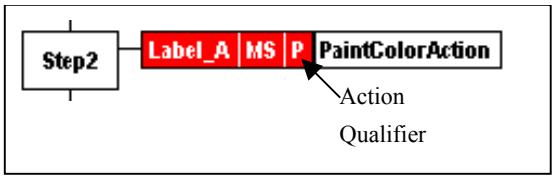



Choose from the following qualifiers. Leave a blank for no qualifier.

| If you do not want the RLL to begin running until the: | Use this qualifier: |
|---|---|
| motion starts | Motion Started (MS) |
| acceleration profile is complete | Acceleration Complete (AC) |
| motion reaches the target speed | At Speed (AS) |
| motion begins the deceleration profile | Deceleration Started (DS) |
| motion is finished | Motion Complete (MC) |
| motion is finished and all axes associated with the motion are within the In Position tolerance of the programmed endpoint | In Position (IP) |
| move command is finished | End of Block (EB) |

### *Action Qualifier*

Action qualifiers specify constraints on the execution of the RLL code. Qualifiers appear within the action as shown below.



| If after the Step becomes active, you want the: | Use this qualifier: |
|---|---|
| RLL to begin running and stops when the Step becomes inactive. | Non Stored (N) |
| RLL to begin running and continue to run until reset by the Reset qualifier. | Stored (S) |
| RLL to execute once. | Pulsed (P) |
| RLL to begin running after a delay*<br>The RLL stops when the Step becomes inactive. | Time Delayed (D) |
| RLL to begin running and stops when the time limit* expires or the Step becomes inactive. | Time Limited (L) |
| RLL to begin running after a delay* and continue to run until reset by the Reset qualifier.<br>If another Action qualifier resets the RLL, during the delay, the reset has no effect because the RLL has not yet been stored.<br>If the Step becomes inactive before the delay completes the RLL is never stored and does not run at all. | Delayed and Stored (DS) |
| RLL to begin running after a delay* and continue to run until reset by the Reset qualifier.<br>If an Action is reset during a delay, then the RLL does not execute. | Stored and Time Delayed (SD) |
| RLL to begin running and after the specified time* stop.<br>A Reset qualifier is required to reset the RLL. Otherwise, without the reset, the RLL does not run again.<br>If the Step becomes inactive, the RLL continues to run until the duration times out. A Reset qualifier is not required, but one can be used to stop the RLL execution. | Stored and Time Limited (SL) |
| RLL begins running and stops after the specified time* expires.<br>If the Step becomes inactive, the RLL continues to run until the duration times out. A Reset qualifier is not required, but one can be used to stop the RLL execution. | Pulse Width (PW) |

The RLL that was started by the Stored qualifier is terminated by the Reset qualifier (R). You can use the Reset qualifier in another action that is associated with the same step or in an action associated with another step. The RLL continues to run between steps. If the action is associated with another step, the action must have the same name as the action that is to be reset.

### *Time Duration*

You can enter the duration directly or click on *Specify Duration* and enter time intervals in the dialog box.

If you enter the duration directly, follow the IEC 1131-3 specification:

T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds.

Examples:

| Time | Format | Time | Format |
|---|---|---|---|
| 14.7 days | T#14.7d | 4 seconds | Time#4s |
| 2 minutes 5 seconds | T#2m5s | 1 day 29 minutes | t#1d29m |
| 74 minutes* | Time#74m | 1 hour 5 seconds 44 milliseconds | T#1h5s44ms |

*The IEC 1131-3 specification allows overflow of the most significant unit in a time duration.

If you prefer to use the dialog box, enter the time into each field as appropriate.

**Notes:**

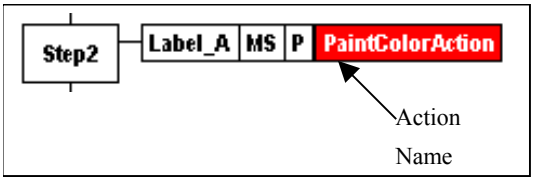
1. If you specify a duration for an action and choose an action qualifier that is not time dependent, the duration is ignored.
2. For an action that has both a program label and a duration specified, duration does not begin timing down until after the step code encounters the program label.

### *Action Name*

Specify the name of the action. Use this name if you refer to the action from another action, such as resetting an action that was stored in another action.

An action can have the same name as an RLL transition. However, the RLL logic for actions and for RLL transitions is scoped differently. Therefore, using the same name for an action does not mean that the same RLL logic is executed for the RLL transition, and vice versa.

The action name appears within the action as shown in the following figure.



### *Action Manager*

Use the Action Manager to you manage the actions that are attached to SFC steps. The Action Manager displays a list of all actions that in the active SFC file and lets you rename and delete those actions.

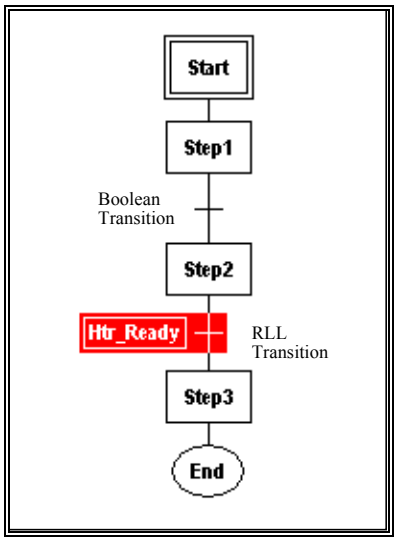
## About Transitions

A transition represents the condition that lets program flow to pass from one or more steps preceding the transition to one or more steps following the transition along the corresponding directed link. Each transition has an associated transition condition that is the result of the evaluation of a single Boolean expression. When the system evaluates the code of a transition, the result must be either TRUE or FALSE.

A transition condition can be a:

- **Boolean Transition** - Boolean expression in the Structured Text language. It is represented as an unlabeled horizontal line
- **RLL Transition** - named RLL object containing a single rung with an output coil that has the same name as the transition object. It is represented as a labeled line (RLL Transitions) containing the name of the RLL output coil.

The step that follows a transition cannot execute until the transition before it evaluates as true.




In this figure, program flow has passed the Boolean Transition and Step2, which follows it, and is currently at the RLL Transition. Until the RLL Transition evaluates to TRUE, Step3 cannot execute.

Program flow into and out of the transition is through a vertical line extending through the horizontal line.

The IEC 1131-3 standard specifies that a SFC diagram must have a transition between every step, and a step between every transition. As an extension to the standard the SFC editor allows you to place one step immediately after

another or one transition after another. However, at runtime the necessary "dummy" steps or "dummy" transitions are inserted automatically.

For information about how transitions are solved, see "How Transitions are Evaluated" on page 76.

### *Transition Parameters*

| This Transition: | Has these parameter(s): |
|---|---|
| RLL | Transition name<br>RLL Logic |
| Boolean | Boolean expression containing operators and symbols |

For more information, see "Working with Transitions" on page 85.

### *About RLL Transition Manager*

Use the RLL Transition Manager to you manage the RLL transitions that are embedded in a SFC program. The RLL Transition Manager displays a list of all RLL transitions that are embedded in the active file and lets you rename and delete those RLL transitions.

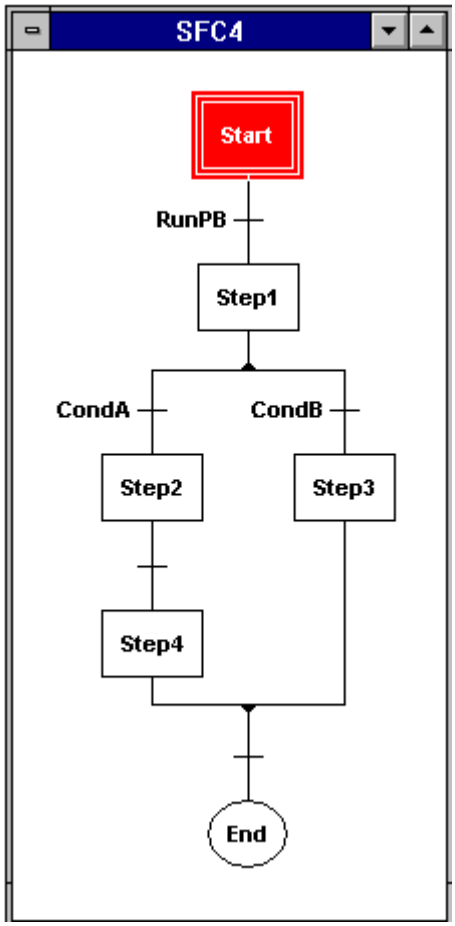
## About Divergences

You can control multiple paths in an SFC using divergences. Divergences can be a:

- select divergence .
- simultaneous divergence.

### *About Selected Divergences*

A select divergence lets you choose from two or more control paths. Each path begins with a transition condition that determines which control path is activated. At some point in the SFC diagram, all paths within the select diverge must converge into a single path. This figure shows a selected divergence.



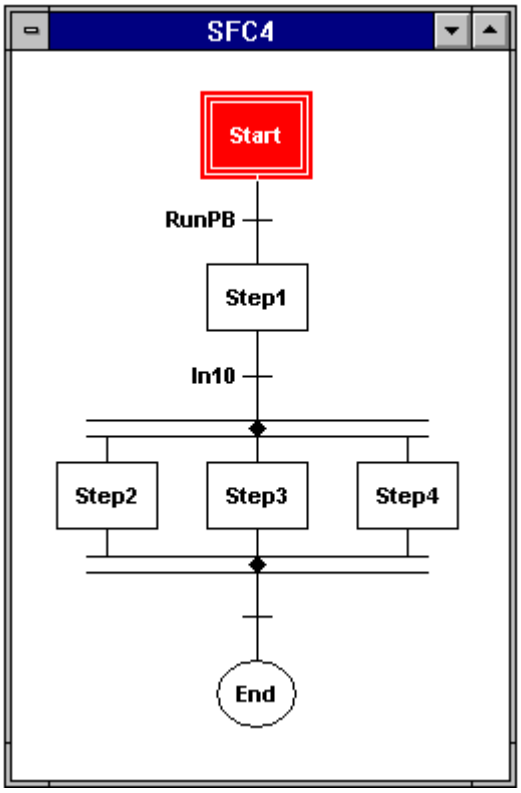

*Selected Divergence*

### *About Simultaneous Divergences*

A simultaneous divergence lets you execute multiple control paths simultaneously in parallel. All the control paths in the simultaneous divergence are activated as soon as power flow enters the divergence. At some point in the SFC diagram, all paths within the diverge must converge into a single path, but the convergence must wait until all the paths have:

- been executed.
- arrived at the point of simultaneous convergence.

The simultaneous diverge is represented as single control path entering at the top and a double horizontal line with two or more control paths exiting from below. Simultaneous convergence is represented as two or more control paths at the top, a double horizontal line, and a single control path exiting at the bottom.




*Simultaneous Divergence*

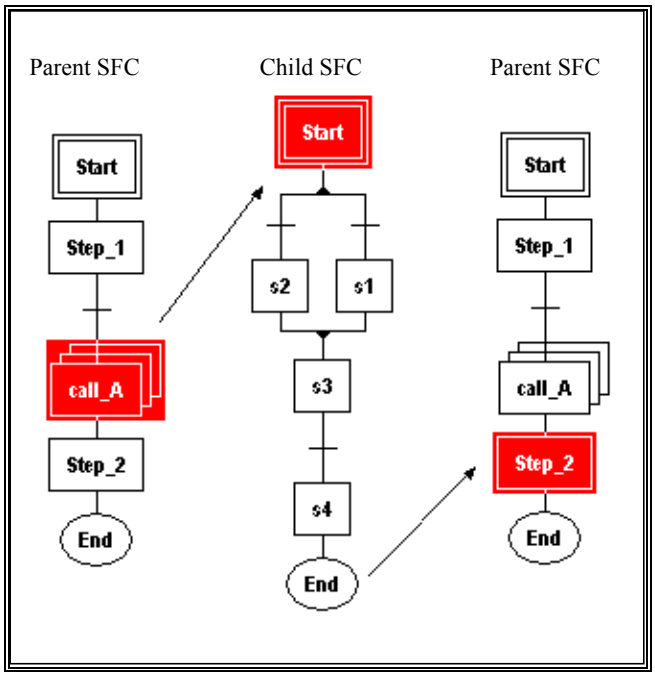
To help ensure proper convergence, do not use labels to jump

- outside a simultaneous divergence.
- into a simultaneous divergence.
- to another path within a simultaneous divergence.

## About Macro Steps

The macro step lets you call one SFC for execution from a step in another SFC. Program flow transfers to the SFC that was called (child SFC). When the child SFC has completed execution, program flow returns to the parent SFC and resumes after the macro step.

In this figure, call_A is the macro step in the parent SFC that calls the child SFC for execution. When the child SFC completes execution, program flow resumes at Step_2 in the parent SFC.




Representation of a macro step is similar to a step, multiple boxes containing an identifier. Program flow into and out of the macro step is through a vertical line entering the top and another line exiting from the bottom.

You can use another program element, called an action,. help coordinate the execution of code within a macro step with other program code.

If a local symbol is defined in a parent SFC, the child SFC can see the local symbol. However, you cannot see this local symbol when viewing the Symbol Manager for the child SFC.

## About Program Flow Control Features

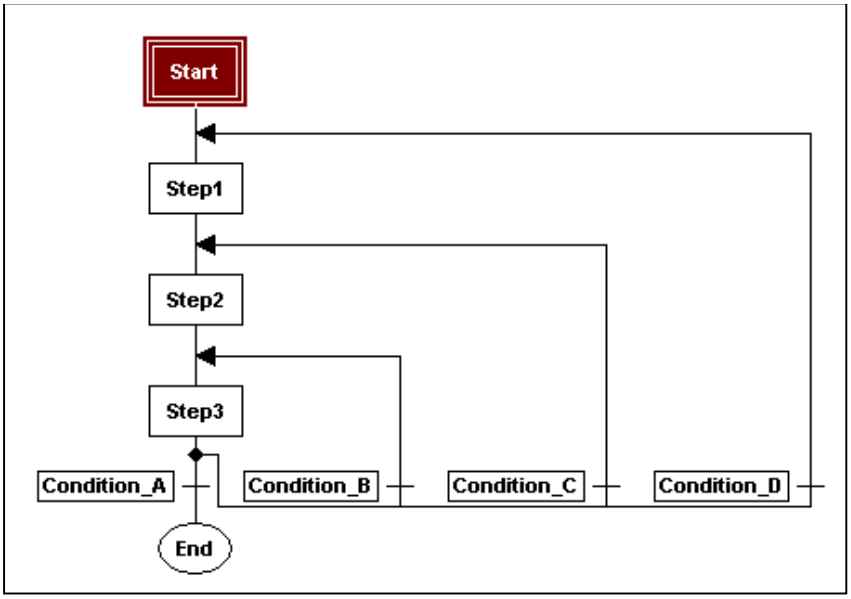
You can use control loops and jump and label constructs to control flow within an SFC diagram.

### *Control Loops*

In an SFC, program flow usually proceeds from top to bottom. Control loops let you go back to a previous location to repeat a Series of steps. A control loop consists of two transitions: one to continue in the downward direction and one on a directed link that loops back in the upward direction. An arrow at the top of the control loop indicates the point at which power flow re-enters the control path.

In the figure below, program flow continues to the End when Condition_A is TRUE. When Condition_B is TRUE and Condition_A is FALSE, program flow returns to the point above Step3. When Condition_C is TRUE and

Condition_A and Condition_B are FALSE, program flow returns to the point above Step2.




With multiple branches, logic evaluation takes place from left to right. If all transitions are FALSE, program flow halts until one transition becomes TRUE.

You can define the transitions for the loop by either of the following methods.

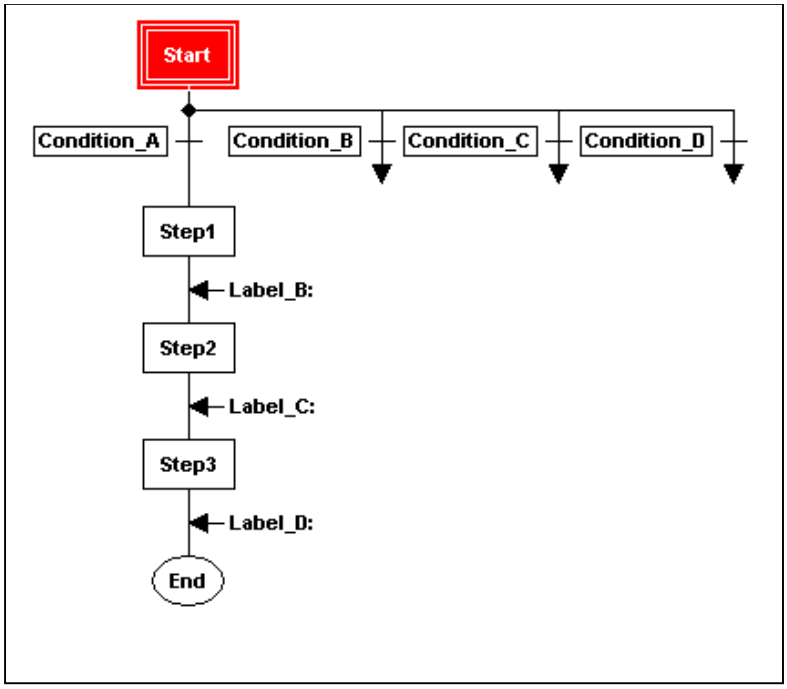
- A Boolean Transition consisting of a Boolean expression that is composed in Structured Text.
- An RLL Transition consisting of a single RLL rung with an output coil having the same name as the transition itself.

For more information, see "Adding SFC Program Flow Controls" on page 90.

### *Jump and Labels*

You can transfer flow to any part of an SFC using a jump and label construct. The jump element is represented by two transitions: one to continue in the downward and one to transfer to a label identifier. The label element is represented by a directed arc to the left that identifies the point where power flow re-enters the control path and a label identifier followed by a colon.

In the following figure, program flow continues to Step1 when Condition_A is TRUE. When Condition_B is TRUE and Condition_A is FALSE, program flow jumps to Label_B, bypassing Step1. When Condition_C is TRUE and Condition_A and Condition_B are FALSE, program flow jumps to Label_C, bypassing Step1 and Step2.




With multiple branches, logic evaluation takes place from left to right. If all Transitions are FALSE, program flow halts until one transition becomes TRUE.

You can define the transitions for the jump by either of the following methods.

- A Boolean Transition consisting of a Boolean expression that is composed in Structured Text.
- An RLL Transition consisting of a single RLL rung with an output coil having the same name as the transition itself.

For more information, see "Adding SFC Program Flow Controls" on page 90.

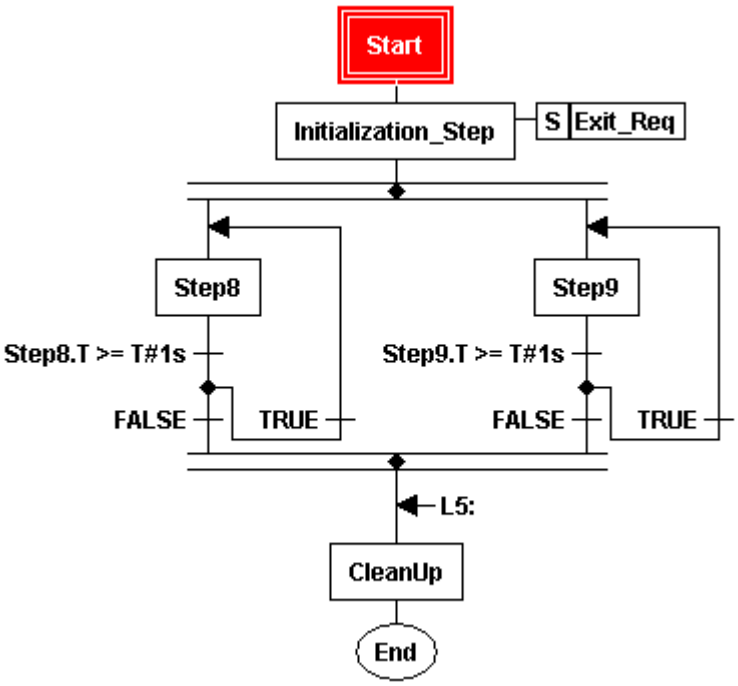
### *Jump/Label Parameters*

| Field | Description |
|---|---|
| Jump Target Label | Specifies Label to which program flow is transferred. |
| Label Name | Specifies point in SFC where program flow resumes. |

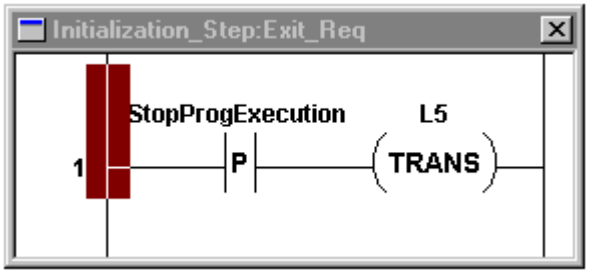
Labels must start with an alphabetical character and be followed by any alphanumeric characters and/or an underscore.

### *Exiting Loop Structures*

There are times when you may wish to exit a looping structure, for example upon an E-Stop condition. Refer to the following figure. In normal operation the divergence loops are executed and continue to execute.




One way to exit from this looping is to use a transition coil in an RLL action. Refer to the following figure. Since Exit_Req is a Stored Action, it continues to run after program flow has exited the associated step. When StopProgExecution is enabled, the transition coil forces execution to the associated label, L5, where program execution continues.



## How SFCs are Solved

Program flow in an SFC moves from top to bottom. The code within each Step is executed. When the code has completed, program flow moves to the next program element.

| If the next element is a: | Then: |
|---|---|
| Step | the code within that step is executed |
| Transition | program flow continues when the transition becomes true |

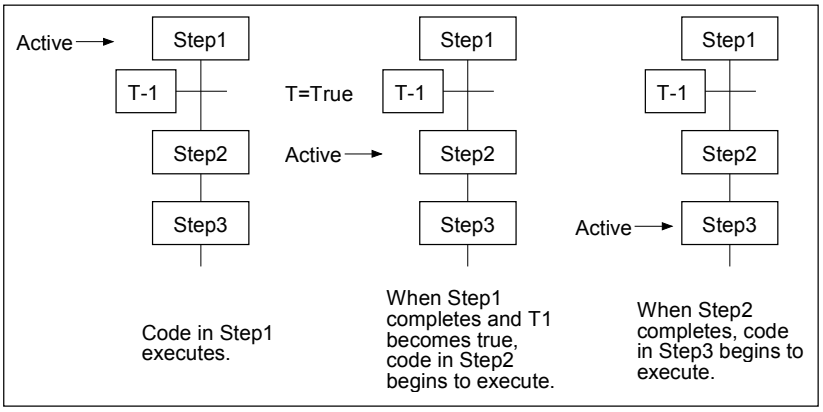
Any step in an SFC takes a minimum of two I/O scans to complete. In the first scan, any actions attached to the step are evaluated. If there are no looping structures (i.e. FOR, WHILE, REPEAT) any Structured Text logic is solved to completion. The transition is tested after all the logic has been solved.

If looping occurs in a Structured Text program, the program is solved to the end and loops back at the start of the next I/O scan. A transition is not tested until the Structured Text has been solved to completion.

You can use the Structured Text constructs END_FOR_NOWAIT and END_WHILE_NOWAIT to finish looping in one I/O scan.

Even though the SFC editor lets you place one step immediately after another, each step still takes **a minimum of two I/O scans** to complete, because parsing the program creates "dummy" transitions between the steps. These transitions are always evaluated true. These transitions are required to be there per the IEC 1131-3 standard. The SFC editor helps reduce your programming effort by automatically including these transitions in the parsed program.

In the following example, one step follows another. Program flow still moves from top to bottom, and execution of a program element does not begin until the preceding element has completed. After the first transition becomes true, Step2 becomes active. After the Step2 logic is complete Step3 becomes active.




The software lets you execute multiple programs of multiple types. For example, you can run two RLL program at the same time as three SFC programs. You can coordinate program execution through global symbols, which are recognized by all program types.

## How Transitions are Evaluated

All Structured Text or RS-274D logic within a steps must be completed before the system evaluates a transition. This is a further enhancement to the IEC-1131-3 specification, which only requires all preceding steps to be active before a transition can be evaluated.

Transitions can follow transitions without steps in between. Once a transition condition is satisfied, the transition is disabled, and next SFC element, step or transition, is activated. At runtime a "dummy" step is inserted (per the IEC standard) between two consecutive transitions. The dummy step performs no logic;however,it takes two I/O scans for the dummy step to activate and deactivate, just like a normal step.

Evaluation of a transition occurs as follows:

| For this transition: | The transition becomes true when: |
|---|---|
| RLL transition | Power flow on the RLL rung reaches the output coil and turns the coil on. Program flow moves to the next Step. |
| Boolean transitions | The Boolean expression resolves to true. Program flow moves to the next Step. |

Transitions must be evaluated as true before program flow can continue to the next step.

| If the transition is: | Then: |
|---|---|
| True | At the next I/O scan, the RLL associated with any active actions is solved with power off to turn off outputs.<br>Then at the subsequent I/O scan, power flow activates the next step. |
| False | The step remains active and any RLL logic is solved.<br>As long as the step is active, any RLL logic is solved; however, Structured Text logic is solved only once at the first I/O scan. |

If a transition is false and remains false, the system does not re-execute the Structured Text or motion code in the steps that precede the transition. However, RLL logic is re-executed. Program flow remains at the transition until the transition becomes true.

## Extensions to IEC 1131-3 SFCs (SFC+/M)

The ASIC-100 product includes several optional extensions to the IEC-1131-3 Sequential Function Charts.

- Step boxes can include RS-274D or Structured Text commands. These commands within a step box are executed sequentially. The execution of a step is not completed until all the commands within a step are completed. The transition conditions following a step are not evaluated until the contents of the step have been executed.
- Steps can be represented by a box containing the step name identifier, a box containing the command list with the step, or an Icon, that consists of a bit mapped picture and up to two lines of text.
- An Icon Palette contains a collection of predefined steps that consists of an icon and a list of commands within the step. These predefined steps can be used for library or canned routines.
- Motion Qualifiers have been added to the Action Qualifiers to synchronize execution of the actions with the motion commands within the Step.
- Files may be marked for inclusion into the command list within a step box. Multiple files can be included. Program commands may be intermixed with the file include statements.
- Steps can follow steps without a transition in between. As soon as the contents of a step have been executed and no transition follows, the step is deactivated, and the next step is activated. At runtime a "dummy" transition is inserted (per the IEC standard) between two consecutive steps. The dummy transitions is always evaluated true.

- Transitions can follow transitions without steps in between. Once a transition condition is satisfied, the transition is disabled, and next SFC element, step or transition, is activated. At runtime a "dummy" step is inserted (per the IEC standard) between two consecutive transitions. The dummy step performs no logic however it takes two I/O scans for the dummy step to activate and deactivate, just like a normal step.
- A Transition Coil (TRANS) has been added to the action logic elements that terminate execution of any commands within a step, stop any motion in progress, deactivate the step, and cause an immediate transition to a label with the same identifier as the transition coil.
- Macro step boxes can be used to include an entire SFC diagram within a SFC macro step. When the macro step is activated, the included SFC diagram begins execution at the start step. When the included SFC diagram executes the end step, the macro step is complete.
- Actions can be attached to macro steps and are be executed as long as the macro step is active.

## About Creating Sequential Function Charts

This section provides information about building an SFC program. Topics include:


- "Creating an SFC Program" on page 79
- "Using the SFC Tool and Menu Bar" on page 80
- "Using the Keyboard


The following key combinations are useful when editing actions without a mouse:

| | |
|---|---|
| Ctrl+Tab | Moves the focus from the parent SFC through all open actions and back to the parent SFC. |
| Esc | Closes the action having the focus. |
| Alt+<dash> (the '-' key) | Displays the system menu in an open action. This allows you to move, size, and close the actiion. |


- Working with Steps" on page 81
- "Working with Transitions" on page 85
- "Working with Macro Steps" on page 86
- "Working with Actions" on page 87
- "Adding SFC Program Flow Controls" on page 90
- "Documenting an SFC Program" on page 98

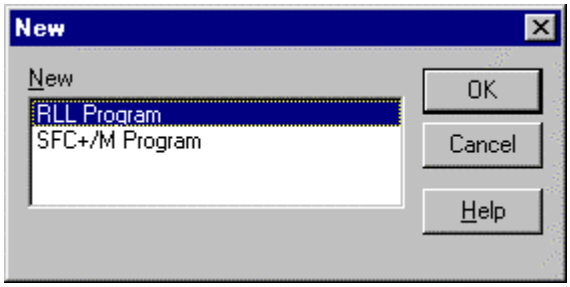
## Creating an SFC Program

After starting the Program Editor, you can create a new SFC program or edit an existing one.

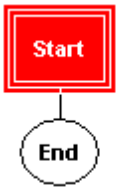
### To create a new SFC program

1. Click on the *New File* on the *Program Editor* tool bar.

2. The new program menu appears.



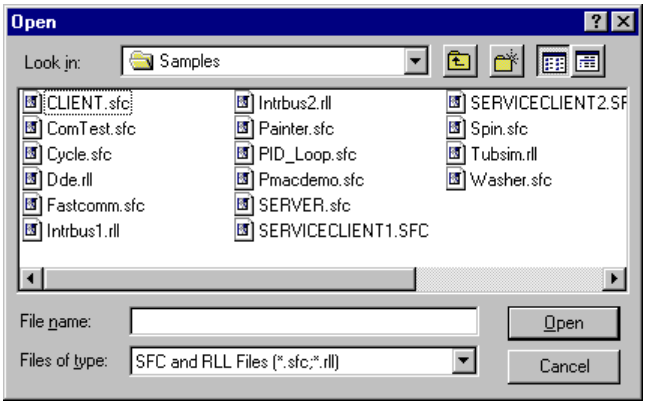

3. Select *SFC+/M Program* and click on *OK*.
4. A new SFC file appears, showing a starting step and an end step.



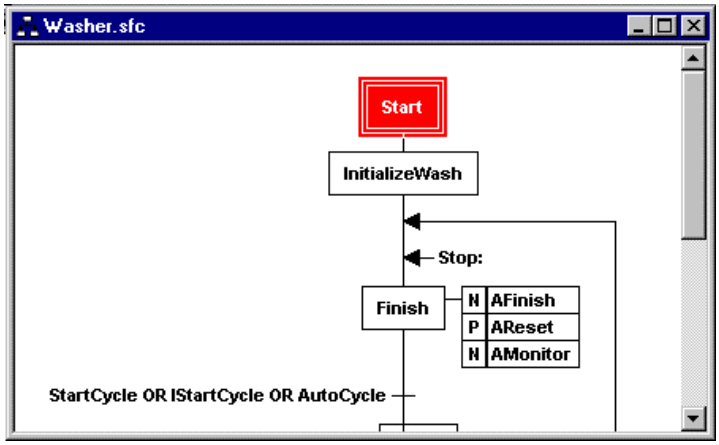

### To edit an existing SFC program

1. Click on *File Open* tool on Program Editor tool bar.

2. The list of existing programs appears.




3. Click on the program to open. You may need to select the SFC program type in the *Files of Type* field first. The SFC program that you selected appears.




4. Begin editing the program elements.

## Using the SFC Tool and Menu Bar

The SFC toolbar contains all the tools needed to create an SFC program.

| Icon | Tool bar Option | Function |
|---|---|---|
|  | — | Lets you select program elements. |
|  | *Insert/Step* | Adds a step to the program. |
|  | *Insert/Macro Step* | Adds a macro step to the program. |

| | | |
|---|---|---|
|  | *Insert/Action* | Adds an action to the program. |
|  | *Insert/Transition* | Adds a transition to the program. |
|  | *Insert/Label* | Adds a label to the program. |
|  | *Insert/Jump* | Adds a jump to the program. |
|  | *Insert/Loop* | Adds a loop to the program. |
|  | *Insert/Select Diverge* | Adds a selected divergence to the program. |
|  | *Insert/Simultaneous Diverge* | Adds a simultaneous divergence to the program |
|  | *Insert/ New App Icon...* | Adds a predefined program step from the library to the program |
|  | *Insert/Comment* | Lets you add comments to the program. |

## Using the Keyboard

The following key combinations are useful when editing actions without a mouse:

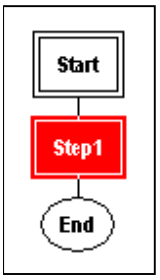
| | |
|---|---|
| Ctrl+Tab | Moves the focus from the parent SFC through all open actions and back to the parent SFC. |
| Esc | Closes the action having the focus. |
| Alt+<dash> (the '-' key) | Displays the system menu in an open action. This allows you to move, size, and close the actiion. |

## Working with Steps

A step represents a condition in which the behavior of the system follows a set of rules defined by the actions and functions associated with the step. After adding a step to a program, you must configure it. For more information about steps, see "About Steps" on page 59.

### *Adding a Step*

1. Click on the *Step Tool* on the SFC toolbar. The cursor changes into the step cursor.
2. Move the cursor to the location in the program where you want to place the new Step and click. The new step appears in the program at the location you specified.



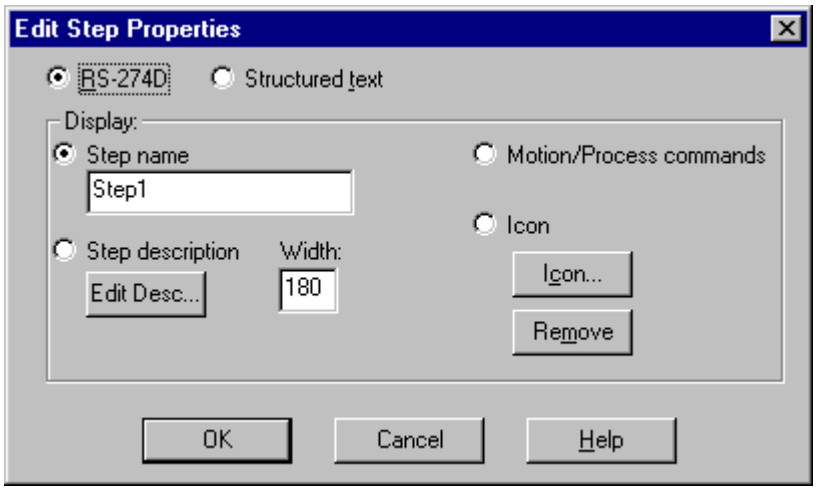

3. Configure the step by editing its properties and commands.

### *Edit Step Properties*

To edit step properties, you must:

- Specify the programming language you want to use for the step.
- Select what information you want to display in the step (i.e. Step Name, Step Description, Motion/Process Commands, or Icon).

1. Select *Step Properties* from the *Edit* or context menu. The *Edit Step Properties* dialog box appears.
2. Refer to the following table when editing step properties. When done, save your changes by clicking *OK*




| To: | Do This: |
|---|---|
| Select the RS-274D programming language. | Click RS-274D |

| To: | Do This: |
|---|---|
| Select the Structured Text programming language. | Click *Structured Text*. |
| Display the step name within the step. | 1. Click *Step Name*.<br>2. Type a step name in the edit box. |
| Display the code (motion control or structured text) within the step. | Click *Motion/Process Commands*.<br>All the programming code for the step appears within the step. For a long series of commands, this can enlarge the size of the step significantly. |
| Display the step description within the step. | 1. Click *Step Description*.<br>2. Click *Edit Description*.<br>3. Enter a *Step Description* in the *Edit Description* box. |
| Set the width in pixels for the step description. | In the *Width* field, enter the number of pixels for the step description. |
| Display an icon within the step. | Click Icon. Refer to **Displaying a Step as an Icon**. |
| Access the Application Icon palette. | Click *Icon*. . . . |
| Delete an icon from the step, if one is assigned. | Click *Remove*. |

### *Edit Step Commands*

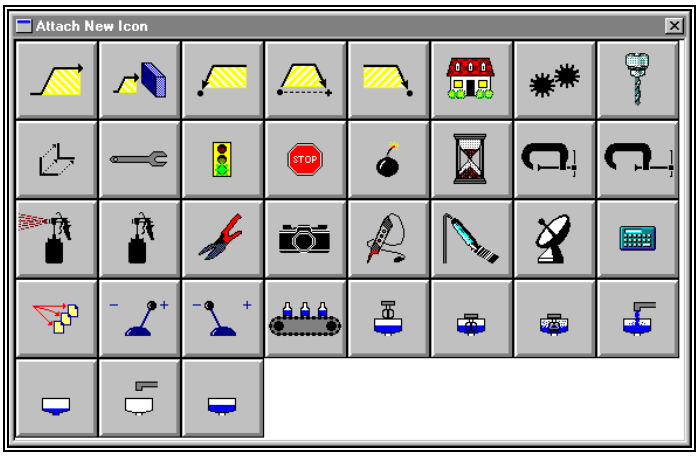
**To edit step commands, do one of the following**

- Double-click on the step.
- Select *Edit Element* from the *Edit* menu or *Edit ST* or *Macro Step* from the context menu.

An editor appears in which you can enter commands. Refer to **Structured Text Programming** for Structured Text commands and to **Integrated Motion** for motion commands.

### *Displaying a Step as an Icon*

1. Edit a step.
2. From the *Edit Step* dialog box, click *Icon*. The *Icon* palette appears.




3. From the *Icon Palette*, click on the icon and enter a title when the system prompts you for it.
4. Click *OK* to return to the *Edit Step* dialog box.

After you finish editing the step, the icon and title appear within the step box.

### *Adding an Application Icon Step*

An Application Icon Step is a step containing these components:

- a code template, which is predefined for a specific function
- an icon, which is appropriate for the function

The control software provides a library of several steps that can be used within an SFC (or you can create your own).

#### To add an Application Icon Step

1. Access the application icons by clicking on the *Icon Tool* on the SFC toolbar.
2. Drop the application icon into the SFC.
3. After placing an application icon step into an SFC, edit it in the same way as you edit a normal step.

Since the code is a template, you need to modify the code to suit your application. You can also make other changes to the step options. For more information, see "Edit Step" on page 82.

Once you customize an application icon step, you can add this customized step to the library. Then whenever you need to use a copy of the customized step within an SFC, access it from the library.

## Working with Transitions

A transition condition is a graphical element in the SFC programming language that represents a single Boolean condition that must be satisfied before program execution continues. For information about transitions, see "About Transitions" on page 67 and "How Transitions are Evaluated" on page 76.

### *Adding an RLL Transition*

1. Click *Edit* on the menu bar and if a check is next to *Boolean Transition* deselect it by clicking.
2. Click on the *Transition Tool* on the SFC toolbar. The cursor changes into the transition cursor.
3. Move the cursor to the location in the program where you want to place the new transition and click. The new transition appears in the program at the location you specified.

### *Editing an RLL Transition*

1. Click the *Select Tool*  and double click on the transition. The *Select RLL Transition Logic* dialog box appears.
2. Enter a name for the transition and click *OK*.

   An RLL transition can have the same name as an action. However, the RLL logic for transitions and for actions is scoped differently. Therefore, using the same name for a transition does not mean that the same RLL logic is executed for the action, and vice versa.

   A window appears containing an RLL rung with a coil of the same name as the transition.
3. Add the RLL elements to the rung through the RLL editor, using the same rules for contacts, coils, jumps, etc.

   You can program only one rung for an RLL transition. For more information, see "**Creating Relay Ladder Logic Program**" on page 46.
4. Save your work by clicking on the *Control Menu* box and selecting *Close*.

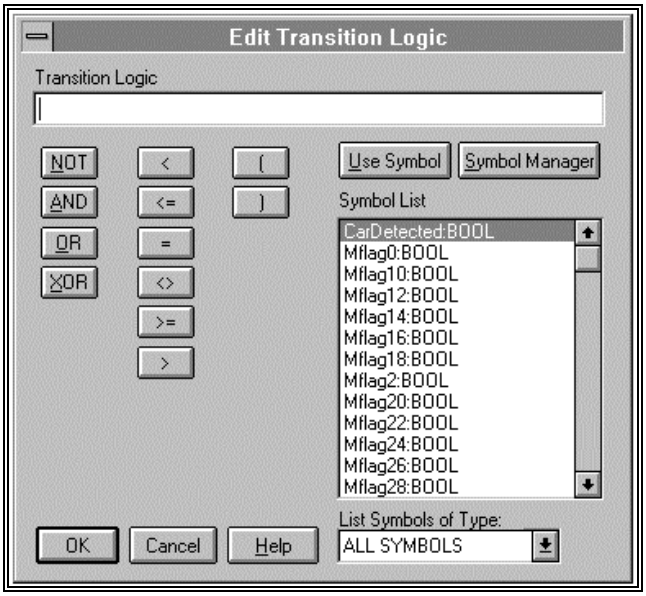
### *Adding a Boolean Transition*

1. Click *Edit* on the menu bar and place a check next to *Boolean Transition* by clicking.
2. Click on the *Transition Tool* on the SFC toolbar. The cursor changes into the transition cursor.

3. Move the cursor to the location in the program where you want to place the new transition and click. The new transition appears in the program at the location you specified.

### *Editing a Boolean Transition*

1. To edit the transition, click on the *Select Tool*  and then double click on the transition. The *Edit Transition Logic* dialog box appears.



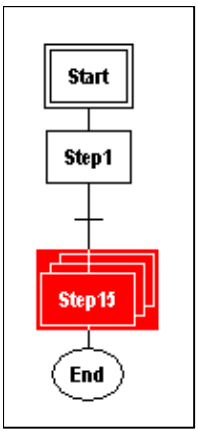

2. Enter the Boolean code for the transition. You can type it in directly or click on the operator buttons and select from symbols that have been defined.
3. Click on *OK* to save your work and close the dialog box.
4. You can access the Symbol Manager to configure local variables and to see a list of all configured variables that you can use in the Boolean expression.

## Working with Macro Steps

A macro step provides a means of calling another SFC from the currently executing SFC. For more information, see "About Macro Steps" on page 70.

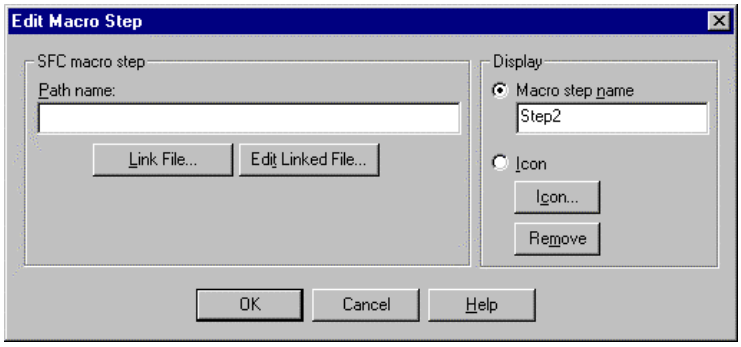
### *Adding a Macro Step*

1. Click on the *Macro Step Tool* on the SFC menu bar. The cursor changes into the *Macro Step* Tool cursor.
2. Move the cursor to the location in the program where you want to place the new macro step and click. The new macro step appears in the program at the location you specified.




### *Configuring a Macro Step*

1. Click on the *Select Tool*  and then double click on the macro step. The *Edit Macro Step* dialog box appears.




2. Enter the appropriate information for configuring the macro step.
3. Click *OK* to save your changes and close the dialog box.

## Working with Actions

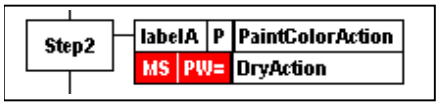
An action consists of one or more sections of RLL code that are associated with a step or a macro step. The system executes an action when its associated step becomes active. For more information, see "About Actions" on page 62.

**Note:** Once you create an action, the only way to delete it is using the Action Manager (from the *Tools* menu). Even if you delete the action from the SFC step, the action remains present in the SFC program (as an unused action) and is compiled along with it. If you get parse errors and cannot seem to locate the problem, it may be in one of these unused actions.

### *Adding an Action*

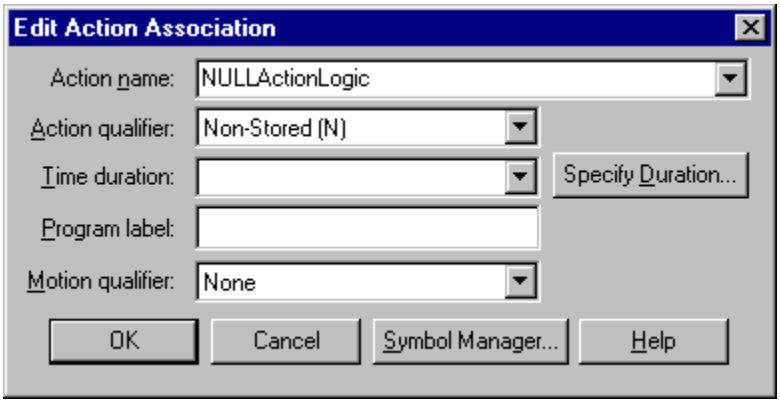
1. Click on the *Action Tool* on the SFC menu bar. The cursor changes into the *Action Tool* cursor.
2. Move the cursor to the location in the program (either on top of a step or on top of a macro step) where you want to place the new action and click. The new action appears in the program at the location you specified.


3. You can add more than one action to a step or macro step by placing the cursor on top of an existing action.




### *Configuring an Action*

Use the *Edit Action Association* dialog box to configure an action.




*Edit Action Association Dialog Box*

| Field/Button | Description |
|---|---|
| Program Label | (Optional) Action does not execute until label in the step code is encountered. The action and the program label must be in the same SFC. No cross-reference between programs is allowed. The labels in a macro step SFC cannot be referenced from the parent SFC, and the labels in the parent SFC cannot be referenced by the macro SFC. If no label is in the step with which the action is associated, then the program label parameter is ignored. |
| Action Qualifier | Specifies an action qualifier. |
| Motion Qualifier | Specifies a motion qualifier. Select None for no qualifier. |
| Time Duration | (Action qualifier only) Specifies the time duration for Limited and Delay qualifiers. If the action qualifier does not actually use the time duration, any value entered for this parameter is ignored. |
| Specify Duration | Accesses the *Define Time Duration* dialog box. |
| Action Name | Specifies the name of the action. |

### *Editing an Action*

1. Click on the *Select Tool*  and then double click on the action. The *Edit Action Association* dialog box appears. Enter the information to configure the Action
2. Click on *OK* to save your changes. The system closes the dialog box and then displays an empty rung of RLL ready for editing.
3. Enter the RLL code.

### *Editing the RLL of an Action*

1. Click on the *Select Tool*  .
2. Double click on the right half of the action as shown.


   The system displays the RLL code for the action.
3. Enter the RLL code.

### *Editing a Configuration Parameter of an Action*

1. Click on the *Select Tool*  .
2. Double click on the left half of the action as shown.


   The system displays the *Edit Action Association* dialog box for the Action.
3. Enter the information for configuring the action.
4. Click *OK* to save your changes.

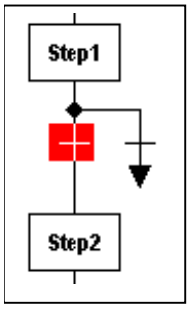
## Adding SFC Program Flow Controls

You can control program flow with jumps, loops, and divergences.

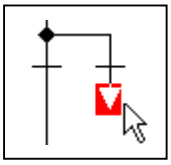
### *Adding a Jump*

A jump-to-label combination is available that lets SFC execution to transfer to any location indicated by a label element. For more information, see "Jump and Labels" on page 72.

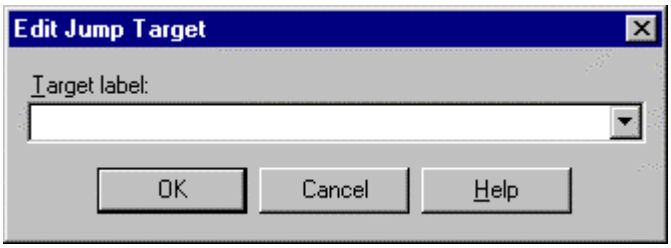
1. Click *Edit* on the menu bar and select *Boolean Transition*.
2. Click the *Jump Tool* on the SFC menu bar. The cursor changes into the *Jump Tool* cursor.
3. Move the cursor to the location in the program where you want to place the new jump and click. The new jump and two transitions appear in the program at the location you specified.




4. To edit the jump, click on the *Select Tool*  and then double click on the arrow head of the jump.



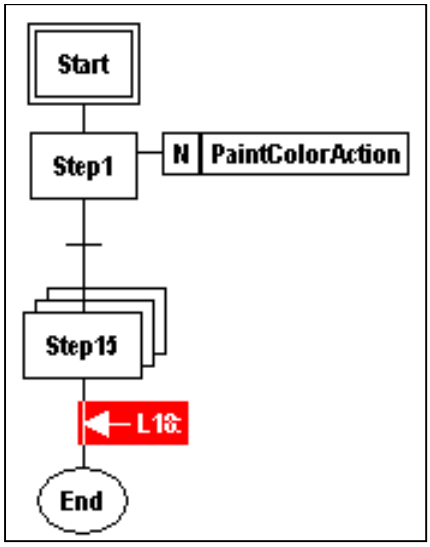
The *Edit Jump Target* dialog box appears.



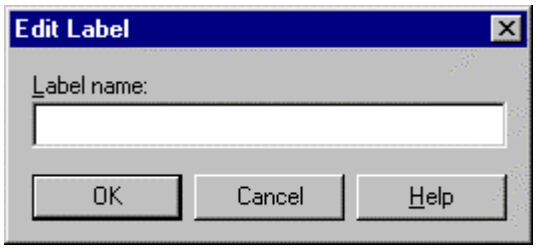

5. Enter the label to which the jump transfers program flow. Then click *OK* to save your changes and close the dialog box.
6. Edit the two transitions.

### *Adding a Label*

1. Click on the *Label Tool* on the SFC menu bar. The cursor changes into the *Label Tool* cursor.
2. Move the cursor to the location in the program where you want to place the new label and click. The new label appears in the program at the location you specified.




3. Click on the *Select Tool*  to edit and then double click on the label. The *Edit Label* dialog box appears.



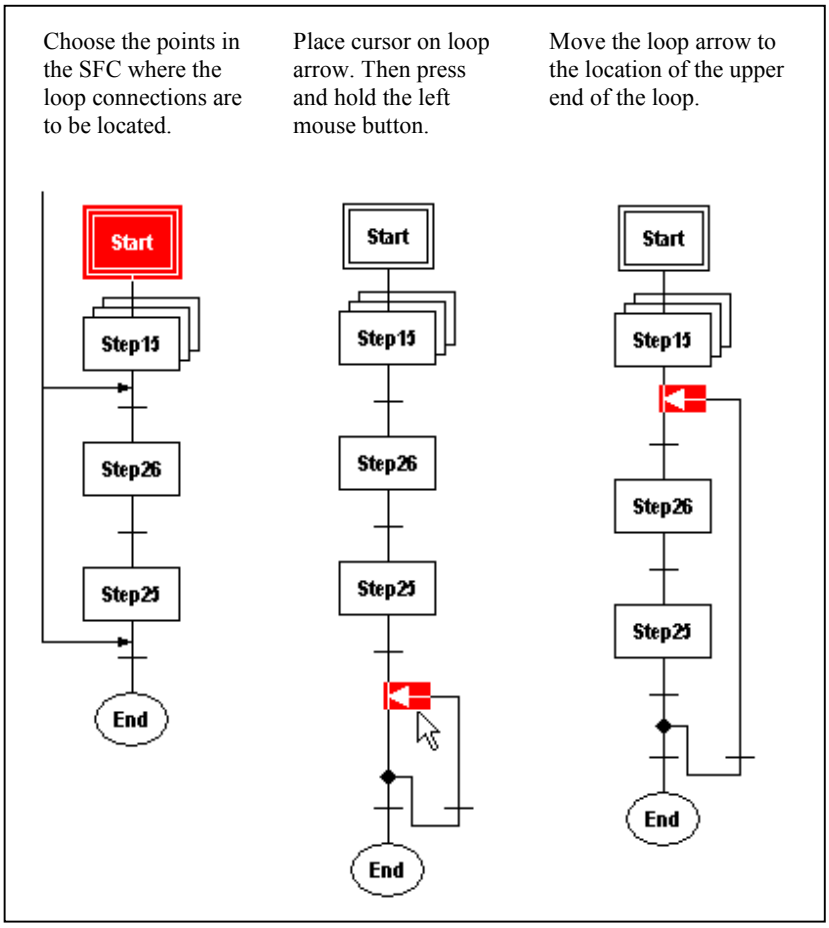

4. Enter a meaningful label and click *OK* to save your changes and close the dialog box.

### *Adding a Loop*

A loop lets the SFC program execution to go back to a preceding location in the program in order to repeat a series of steps. For more information, see "Control Loops" on page 71.

1. Click on the *Loop Tool* on the SFC menu bar. The cursor changes into the *Loop Tool* cursor.
2. Move the cursor to the location in the program where you want to place the lower end of the loop and click. The loop appears in the program at the location you specified.

3. Click on the *Select Tool*  and then place the cursor over the loop arrow.
4. Press and hold the left mouse button and move the loop arrow to the point where the upper end of the loop is to be located.




5. Edit the loop transitions.

## *Moving a Loop*

### Dragging the top of a loop

1. Select the selector tool from the *RLL Tool Bar* or the *SFC Tool Bar*.

2. Move the cursor over the loop top arrow. Press and hold the left mouse button. Drag the loop arrow to the desired location (the cursor will change to a loop arrow cursor).

3. Release the left mouse button to drop the loop arrow at the desired location.

To cancel the drag operation press the Esc key.

| If the target location for the loop arrow is: | Then: |
|---|---|
| 1. on the same diverge branch and is not below the loop transition to which this loop arrow is tied<br>2. not inside of any embedded loop<br>3. not outside of any nested loops | The dragged loop arrow is located at the drop position, and the loop transition remains in its current position. |
| • on a different diverge<br>• below the loop transition to which this loop arrow is tied<br>• or outside of a nested loop | The entire loop and all of its contents both are moved to the target position. |
| inside of an embedded loop | The drag fails |

**Dragging the bottom of the loop**

1. Move the cursor over the loop transition. Press and hold the left mouse button. Drag the loop transition to the desired location (the cursor will change to a loop transition cursor).
2. Release the left mouse button to drop the loop transition at the desired location.

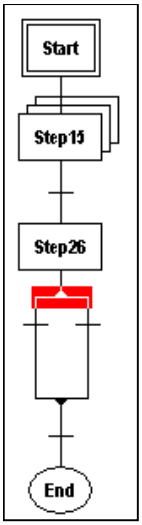
To cancel the drag operation press the Esc key.

| If the target location for the loop transition is: | Then: |
|---|---|
| on the same diverge branch and is not above any of the loop arrows tied to the loop transition<br>not inside of any embedded loops<br>not outside of any nested loops | The dragged loop transition is located at the drop position, and the loop transition remains in its current position. |
| on a different diverge<br>above any of the loop arrows tied to the loop transition<br>or outside of a nested branch | The entire loop and all of its contents is moved to the target position. |
| inside of an embedded loop | The drag fails, and the loop returns to its previous position. |

### *Adding a Select Divergence*

A select divergence lets the SFC program execution to follow one of two or more control paths. For more information, see "About Divergences" on page 68.

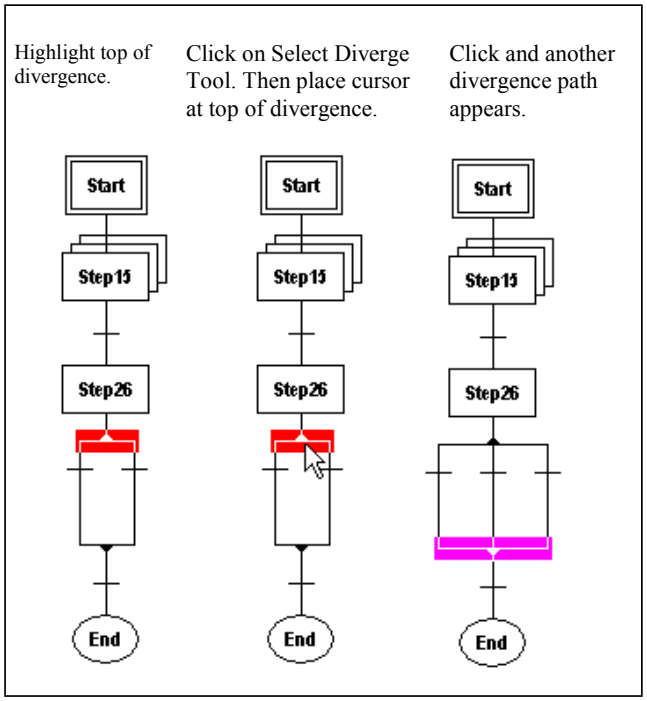
1. Click *Edit* on the menu bar and select *Boolean Transition* for the type of transitions to use with the divergence: RLL or Boolean.
2. Click on the *Select Diverge Tool* on the SFC menu bar. The cursor changes into the *Select Divergence Tool* cursor.
3. Move the cursor to the location in the program where you want to place the select divergence and click. The select divergence appears in the program at the location you specified.




4. Edit the two transitions.

#### To add another path

1. Highlight the top of the select divergence before adding another select divergence.
2. Locate the cursor at the top of the divergence and click. Another divergence path appears.



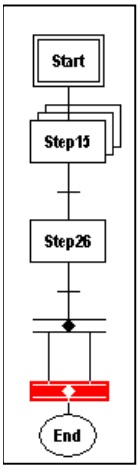

### To delete a path

1. Highlight the path you want to delete.
2. Click the Cut tool  .

## *Adding a Simultaneous Divergence*

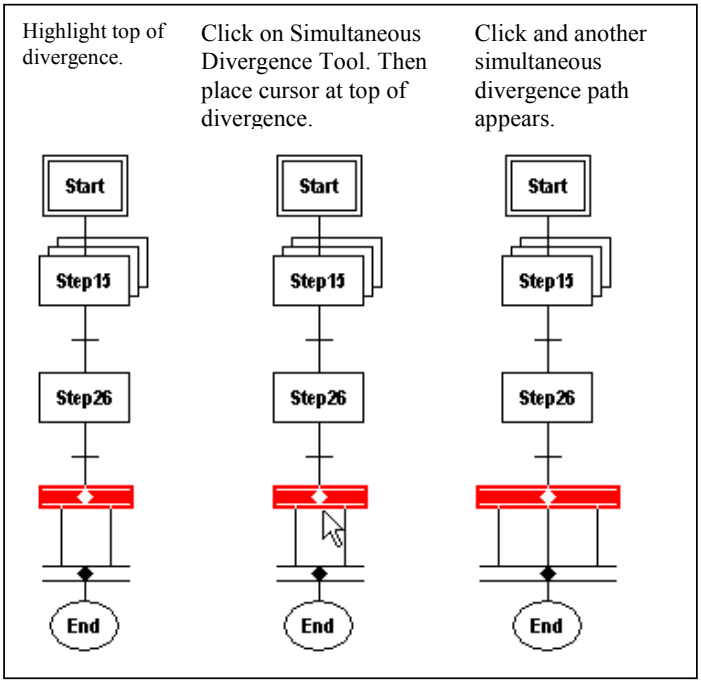
A simultaneous divergence lets the SFC program execution to follow two or more control paths. Execution along each path must be completed for program execution to proceed beyond the simultaneous divergence. For more information, "About Divergences" on page 68.

1. Click on the *Simultaneous Divergence Tool* on the SFC menu bar. The cursor changes into the *Simultaneous Divergence Tool* cursor.
2. Move the cursor to the location in the program where you want to place the *Simultaneous Divergence* and click. The *Simultaneous Divergence* appears in the program at the location you specified.




### To add another path

1. Highlight the top of the simultaneous divergence before adding another.
2. Locate the cursor at the top of the simultaneous divergence and click. Another divergence path appears.



**To delete a path**

1. Highlight the path you want to delete.
2. Click the *Cut tool*  .

**To delete an entire simultaneous divergence**

1. Highlight either the top or the bottom of the divergence
2. Click on the *Cut Tool*.

### *Guidelines for Using Simultaneous Divergence*

Observe the following guidelines when you create a simultaneous divergence.

To ensure that proper convergence, do not use labels in the following ways:

- To jump outside a simultaneous divergence.
- To jump into a simultaneous divergence.
- To jump to another path within a simultaneous divergence.

## Documenting an SFC Program

You can document a SFC program by adding program comments. This section explains how to add, edit, and view program comments.

### *Adding Program Comments*

A program comment can consist of any meaningful description that you want to display adjacent to a program element. You can choose whether the system displays the comments or hides them.

1. Click on the *Comment Tool* on the SFC menu bar. The cursor changes into the Program Comment Tool cursor.
2. Move the cursor to the location in the program where you want to place the comment and click. The *Program Comments* dialog box appears.
3. Enter the comment and click on *OK*. The comment appears in the program at the location you specified.

### *Editing Program Comments*

1. Double click on the comment you want to edit. The *Program Comments* dialog box appears.
2. Edit the comment as desired.

### *Viewing a Comment*

Click *View* and select *Program Comments* or click the view comments button on the editor tool bar.

### To deactivate the view comments mode

Click View and select *Program Comments* or press the view comments button again.

> **Note:** The view comments mode is activated and deactivated for all files.

When the view comments mode has been activated the view menu will display a check mark next to the Program Comments command and the edit tool bar will display the view comments button as depressed.

# Structured Text Programming

## Introduction

The Structured Text programming language is an IEC 1131-3 textual programming language. It is convenient for those who have experience with structured BASIC, Pascal, C or other high-level programming languages.

You use the Structured Text editor to create stand-alone structured text programs. The Structured Text editor features typical text-editing functions such as cut, copy, and paste, find, and replace. It also has tools and commands to automatically insert statement constructs such as IF and CASE selections statements and FOR, REPEAT, and WHILE loops,

You can also incorporate structured text commands into a Sequential Function Chart (SFC) step. A patented extension to the SFC language allows the integration of Structured Text into an SFC step. When you create the application code for an SFC step, you can choose to use Structured Text code. When the SFC is executed, the Structured Text code that you incorporate within each step is processed as the step becomes active. Except for certain functions, the stand-alone and SFC step Structured Text editors operate the same.

This section provides information on using the Structured Text Editor. It is assumed that you are familiar with general Program Editor operation and have some familiarity with the Structured Text language.

**Note:** Stand-alone Structured Text programs run once and exit.

## Structured Text Editor Overview

**Note:** Refer to Customize Text Editor for information on setting tab, color, and font options for the editor.

## Opening a Structured Text Document

### To open an existing document

- Select *Open Editor* from the Program Editor *File* menu and locate the document using the *Open* dialog box that appears.

### To open a new document

- Select *New Editor* from the Program Editor *File* menu and choose *Structured Text Document* from the *New* dialog box that appears.

## Editing Structured Text in an SFC Step

### To edit Structured Text in an existing SFC step

- Double-click on the step or select the step and choose *Edit Element* from the *Edit* menu or *Edit ST or Macro Step* from the context menu.

  Make sure that the *Step Properties* is set to *Structured Text*.

## Entering Statements

### Manual Entry

Statements can be entered by typing in the statement or function call and associated parameters. Be sure to review the syntax of statements in **Language Overview** and refer to the function call syntax in **Language Reference**.

### Accessory Bar

The accessory bar shows commands in graphic form. It is an alternative method of entering statements. The accessory bar appears only when enabled by toggling *Accessory Bar* on the *View* menu. The following figure shows the accessory bar functions. The floating accessory bar can be undocked and positioned at the user's convenience.


### Insert ST Statements Menu

When the Structured Text editor is active, the Program Editor *Edit* menu has an *Insert ST Statements* item that lists Structured Text statements. Select the statement you need and it will automatically be entered at the cursor position in the proper syntax. Replace the any parameters and expressions as needed. Optional parts of the statement should be removed if they will not be used.

### Insert ST Functions Menu

When the Structured Text editor is active, the Program Editor *Edit* menu has an *Insert ST Function Calls* item that lists standard functions that can be used with in the Structured Text language. Select the function you need and it will automatically be entered at the cursor position with the correct syntax. Replace any parameters with ones you have defined in the Symbol Manager.

Refer to **Language Overview** for more information on using functions and function blocks in the Structured Text language.

## Editing Structured Text

The Structured Text editor supports the common editor functions such as cut, copy, and paste, and find and replace. These commands are found on the *Edit* and context menu.

## Bookmarks

Bookmarks allow you to quickly position the cursor at specified lines within the editor. You can set and reset bookmarks and locate previously set bookmarks.

### To set or reset a bookmark

1. Position the cursor at any line within the editor at which you wish to set a bookmark.

- Select *Toggle Bookmark* from the context menu or use the Ctrl-F2 key combination. When a bookmark is toggled on, a blue dot marks its position in the left-hand column of the editor.

### To position the cursor at a bookmark

- With one or more bookmarks previously set, press F2. The cursor will position (and the screen will scroll if necessary) at the bookmark. Successively pressing F2 will cycle through all bookmark positions.

## Printing

### To perform printer setup

- Choose *Print Setup* from the *File* menu. The standard Windows print setup dialog box appears. Make any changes and click *OK* to save the print setup information and continue.

### To view a print preview

- Choose *Print Preview* from the *File* menu or tool bar. The print preview is displayed.

### To print the structured text file

- Choose *Print* from the *File* menu or tool bar. The standard Windows print dialog box appears. Make any changes to the print options and click *OK* to print the Structured Text file.

## Saving

### To save editing changes

- Choose *Save* from the *File* menu or tool bar. The Structured Text file is saved.

## Exiting the Editor

### To exit the editor

- Choose *Exit* from the *File* menu.

If there are any editing changes that need saved, a prompt appears requesting to save the changes first.

# Language Overview

## Expressions

An expression is defined as a combination of operators (mathematical, logical, relational) and operands (constants, symbols, literal values, other expressions) that can be evaluated, yielding a result in one of the supported data types, e.g., integer, real number, etc.

## Operators

The table below lists the operators that you can use within an expression. The order of precedence determines the sequence in which they are executed within the expression. The operator with the highest precedence is applied first, followed by the operator with the next highest precedence. Operators of equal precedence are evaluated left to right.

| Operator | Symbol | Precedence |
|---|---|---|
| Structure Index | . | 1 (Highest) |
| Array Index | [ ] | 1 |
| Pointer Reference | & | 2 |
| Pointer Dereference | * | 2 |
| Parenthesis | ( ) | 3 |
| Function Evaluation | Identifier (argument list)<br>e.g., LN (A), ABS (X) | 3 |
| Exponentiation | **, POW | 4 |
| Negate | - | 5 |
| Complement | NOT | 5 |
| Multiply | * | 6 |
| Divide | / | 6 |
| Modulo | MOD | 6 |
| Add | + | 7 |
| Subtract | - | 7 |
| Comparison | <, >, <=, >= | 8 |
| Equality | = | 9 |
| Inequality | <> | 9 |
| Boolean/Bitwise AND | AND | 10 |
| Boolean/Bitwise Exclusive OR | XOR | 11 |
| Boolean/Bitwise OR | OR | 12 (Lowest) |

These symbols have the following functions.

| | |
|---|---|
| := | assigns an expression to a symbol |
| ; | required to designate the end of a statement |
| [ ] | used for array indexing where the array index is an integer. For example, this sets the first element of an array to the value j+10: intarray[i] = j + 10; |
| :: | used to designate a label. For example, this specifies a label: spray_on:: Labels must be followed by a statement on the same line. |
| (* *) | designates a comment. For example, (*This is a comment.*) |

## Pointer Operators

Structured text has two pointer operators: the pointer reference & operator and the pointer dereference * operator. These operators are used in indirect addressing operations.

### *Indirect Addressing Description*

In programming languages, data values are typically referred to by symbolic name. This is known as direct addressing. The data value is given a symbolic name and that symbolic name is used to directly access that data value.

```
X := Y;
```

The data value known as X is assigned the value of the Y data value.

Indirect addressing is a common paradigm in programming languages. When using indirect addressing, the symbolic name refers to the location where the data value is stored. These indirect symbols are commonly called pointer symbols. To get the actual data value using indirect addressing, the symbolic name of the pointer symbol is used to obtain the location of the data value, then the location of the data value is used to get the actual data value (an indirect operation).

If pVar1 is a pointer symbol, then in the following assignment, pVar1 is assigned the location of the X data value.

```
pVar1 := & X;
```

If pVar1 is a pointer symbol, then in the following assignment, Y is assigned the value contained in Var1, since pVar1 contains the location of Var1.

```
Y := * pVar1;
```

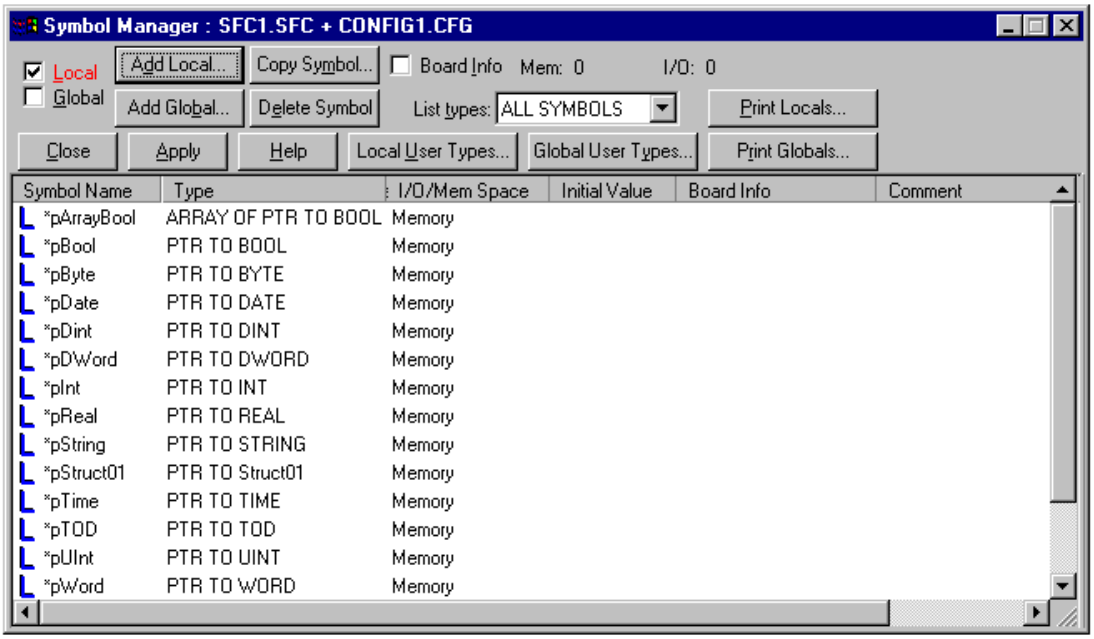
If pVar1 is a pointer symbol, then in the following assignment,Var1 is assigned the value contained in Y.

```
* pVar1 := Y;
```

When a pointer symbol is defined, it is defined as a pointer to a symbol of a specific data type (REAL, INT, STRING, etc.) The pointer symbol pVar1 could be assigned the location of any symbol of its data type.

### *Pointer Definition*

As with other symbols, pointer symbols are defined in the Symbol Manager. The following figure shows the definition of several pointer types.




Pointers to standard data types, user structures, and arrays can be defined. Pointers to function blocks and system objects cannot be defined.

> **Note:** Pointers cannot be passed into FILE functions, bit array functions (SHL, AND_BITS, etc.), and STRING_TO_ARRAY functions.

### *Structured Text Pointer Usage*

Assume VarInt1 and VarInt2 are integer symbols and pInt is defined as a pointer to integer.

**To assign a location to a pointer symbol**

```
pInt := & VarInt;
```

pInt is assigned the location of VarInt. & means "get the location of".

**To get the data value referenced by a pointer symbol**

```
VarInt2 := * pInt;
```

VarInt2 is assigned the value of VarInt1 . * means "get the value located at".

**To use an array pointer**

Assume IntArray is an array of 10 integers (IntArray: ARRAY [1..10] OF INT) and pArray is defined as a pointer to integer.

For array symbols, the name of the array (in this case IntArray) is similar to a pointer to the array data values. However, the array name can never be assigned to a different location, it will always point to the array.

```
pArray := &IntArray;
```

pArray is assigned the location of IntArray.

```
pArray[index] := VarInt1;
```

IntArray[index] is assigned the value of VarInt1.

```
pArray := & IntArray;
```

pArray points to the first element of the array.

```
pArray := & IntArray[index];
```

pArray points to the element of the array located at index.

**To use a pointer to a user structure**

Assume a user structure UserStruct1 is defined and pStruct1 is defined as a pointer to this user structure.

For user structure symbols, the name of the user structure (in this case UserStruct1 ) is a pointer to the user structure data values. However, the user structure name can never be assigned to a different location, it will always point to the user structure.

```
pStruct := UserStruct1;
```

pStruct is assigned the location of UserStruct1.

```
pStruct.intMember1 := VarInt1;
```

The intMember1 member of UserStruct1 is assigned the value of VarInt1.

**To use a pointer to an array of user structures**

Assume an array of user structures structArray (structArray : ARRAY [1..10] OF USER_STRUCT1) is defined, and a pointer to the user structure pStructArray is defined (pStructArray : PTR TO USER_STRUCT1).

```
pStructArray := &structArray;
```

pStructArray is assigned the location of structArray.

```
pStructArray[5].intMember1 := VarInt1;
```

The intMember1 member of structArray[5] is assigned the value of VarInt1.

**To use an array of pointers which point to an INT**

Assume an array of pointers ArrayPtr (ARRAY [1..10] OF PTR TO INT) is defined.

```
ArrayPtr[index] := VarInt1;
```

ArrayPtr[index] is assigned the address of VarInt1.

```
*ArrayPtr[index] := VarInt2;
```

VarInt1 is assigned the value of VarInt2.

**To use an array of pointers which point to an array of INT**

Assume an array of integers intArray (ARRAY [1..10] OF INT) and an array of pointers ArrayPtr (ARRAY [1..10] OF PTR TO INT) are defined.

```
ArrayPtr[index] := &IntArray;
```

ArrayPtr[index] is assigned the address of the first element in intArray.

```
*ArrayPtr[index] := VarInt1;
```

IntArray[1] is assigned the value of VarInt1.

**To assign a null value to a pointer symbol**

```
pInt := NULL;
```

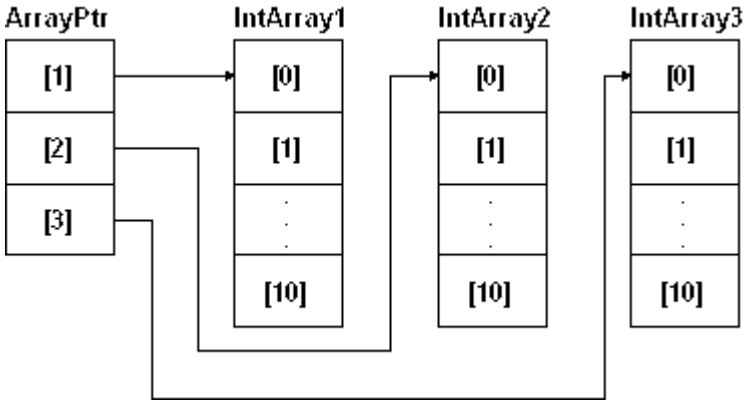
pInt is assigned the NULL pointer value. Pointer symbols can be assigned to and compared (equal and not equal to) NULL. NULL is a system keyword.

**To use an array of pointers to arrays of INT**

Assume arrays of integers intArray1, intArray2, IntArray3 (ARRAY [0..10] OF INT) and an array of pointers ArrayPtr (ARRAY [1..3] OF PTR TO INT) are defined.

```
ArrayPtr[1] := &IntArray1[0];
ArrayPtr[2] := &IntArray2[0];
ArrayPtr[3] := &IntArray3[0];
```

Refer to the following figure.



Then the following syntax can be used:

```
ArrayPtr[1] [0] := VarInt1;        (*IntArray1[0] = VarInt1*)
ArrayPtr[1] [1] := VarInt2;        (*IntArray1[1] = VarInt2*)
ArrayPtr[1] [2] := VarInt3;        (*IntArray1[2] = VarInt3*)

ArrayPtr[2] [0] := VarInt1;        (*IntArray2[0] = VarInt1*)
ArrayPtr[2] [1] := VarInt2;        (*IntArray2[1] = VarInt2*)
ArrayPtr[2] [2] := VarInt3;        (*IntArray2[2] = VarInt3*)

ArrayPtr[3] [0] := VarInt1;        (*IntArray3[0] = VarInt1*)
ArrayPtr[3] [1] := VarInt2;        (*IntArray3[1] = VarInt2*)
ArrayPtr[3] [2] := VarInt3;        (*IntArray3[2] = VarInt3*)
```

## Structured Text Syntax

If you use the built-in editor tools (ST accessory bar and insert menus) to insert Structured Text statements and functions, the correct syntax is automatically entered for you. Keywords appear in all upper case, user-replaceable expressions and parameters appear in mixed case, and options appear in brackets []. Refer to the **Language Reference** for more information on using the Structured Text language, functions, and function blocks.

**Notes:**

1. Structured Text statements must end in a semi-colon (;).
2. Structured Text symbols must be declared in the Symbol Manager.

## Assignment Statement

The assignment statement replaces the value of a variable with the result of evaluating an expression (of the same data type).

**Format**

```
Variable := Expression;
```

Where:

| | |
|---|---|
| Variable | is a symbol, array, array element, etc. |
| Expression | is a single value, expression, or complex expression. |

**Example 1**

Boolean assignment statements:

```
VarBool1 := TRUE;
VarBool2 := (val <= 75);
```

**Example 2**

Array element assignment:

```
Array_1[13] := (RealA /RealB)* PI;
```

**Example 3**

String assignment. String literals must be enclosed in single quotation marks.

```
String_Val := 'This is a string constant';
```

**Example 4**

Function value assignment:

```
Result := SQRT(2);
```

**Example 5**

Function block value assignment: Assume the counter function block instance CTU1 is located in an RLL diagram, then the following assignment in the structured text program gets the current value of the counter.

```
CurrentValue:=CTU1.CV;
```

**Example 6**

Pointers:

```
pVar1 := & X;
```

If pVar1 is a pointer symbol, then pVar1 is assigned the location of the X data value.

**Example 7**

Pointers:

```
Y := * pVar1;
```

If pVar1 is a pointer symbol, Y is assigned the value contained in Var1 , since pVar1 contains the location of Var1.

**Example 8**

Pointers:

```
* pVar1 := Y;
```

If pVar1 is a pointer symbol, Var1 is assigned the value contained in Y.

## BREAK Statement

The BREAK statement stops program execution if debugging is enabled (the program was started with the *Run with Debug* command).

**Format**

```
BREAK;
```

**Example**

If debugging is enabled, program execution stops at the BREAK statement. The statements in StatementList2 and succeeding program statements can be executed by single-stepping through the program.

```
StatementList1;
BREAK;
StatementList2;
```

## CASE Statement

The CASE construct offers multiple-choice conditional execution of statement lists. It conditionally executes one of multiple statement lists in which the condition is determined by the value of an integer variable.

**Format**

```
CASE   IntExpression OF
       Int:                          (*Singular*)
              StatementList;
       Int,Int,Int:                  (*Enumerated*)
              StatementList;
       Int..Int:                     (*Range*)
              StatementList;
[ELSE                                (*Optional*)
              StatementList;]
END_CASE;
```

Where:

| | |
|---|---|
| IntExpression | A variable or expression having a data type of ANY_INT. |
| Int | An integer. Zero or more of each of the singular, enumerated, or range forms can be used. |
| StatementList | Zero or more Structured Text statements. |

**Operation**

The Int values are compared to IntExpression. The StatementList following the first Int value that matches IntExpression is executed. If no Int value matches IntExpression, the StatementList following ELSE is executed; otherwise, no StatementLists are executed. The ELSE part of the CASE construct is optional.

**Example**

This code fragment assigns a value to a string variable.

```
CASE ColorSelection OF
       0:
              ColorString := 'Red';
       1:
              ColorString:='Yellow';
```

```
        2,3,4:
                ColorString:='Green';
        5..9:
                ColorString:='Blue';
ELSE
                ColorString:='Violet';
END_CASE;
```

## Comments

Comments let you incorporate useful annotations into your program code to document program operation. The compiler ignores anything between a (*…*) pair. A comment can be placed after a line of code or on a separate line.

**Format**

```
(*free-form text*)
```

**Example**

```
Result := SQRT(x);      (*Uses the square root function*)
```

## Exit Statement

The EXIT statement is used to terminate and exit from a loop (FOR, WHILE, REPEAT) before it would otherwise terminate. Program execution resumes with the statement following the loop terminator (END_FOR, END_WHILE, END_REPEAT).

**Format**

```
ConditionForExiting EXIT;
```

Where:

| | |
|---|---|
| ConditionForExiting | An expression that determines whether to terminate early. |

**Example**

The following code fragment shows the operation of the EXIT statement. When the variable number exceeds 500, the FOR loop is exited and execution continues with the statement immediately following END_FOR.

```
number:=1
FOR counter := 1 TO 100 DO
        number := number * counter;
        IF number > 500 THEN EXIT;
END_FOR;
```

# IF Statement

The IF construct offers conditional execution of a statement list. The condition is determined by result of a Boolean expression. The IF construct has two optional parts: One option provides conditional execution of an alternate statement list as determined by a second Boolean expression. Another option provides unconditional execution of a third statement list if neither condition is satisfied.

If neither Boolean expression is TRUE and you have included an ELSE statement, the statements following the ELSE are executed. If an ELSE statement is not present, no statements are executed.

**Format**

```
IF  BooleanExpression1  THEN
        StatementList1;
[ELSEIF  BooleanExpression2  THEN    (*Optional part*)
        StatementList2;]
[ELSE                                (*Optional part*)
        StatementList3;]
END_IF;
```

Where:

| | |
|---|---|
| BooleanExpression | Any expression that resolves to a Boolean value. |
| StatementList | Any set of Structured Text statements. |

**Operation**

The following sequence of evaluation occurs if both optional parts are present:

If BooleanExpression1 is TRUE, then StatementList1 is executed. Program execution continues with statement following the END_IF keyword.

If BooleanExpression1 is FALSE and BooleanExpression2 is TRUE, then StatmentList2 is executed. Program execution continues with statement following the END_IF keyword.

If both Boolean expressions are FALSE, then StatmentList3 is executed. Program execution continues with statement following the END_IF keyword.

If an optional part is not present, then the sequence of evaluation skips to the next

**Example**

The following code fragment puts a text message into the string variable Message, depending on the value of I/O point input value.

```
IF Input01 < 10.0 THEN
        Message := 'Low Limit Warning';
ELSEIF Input02 > 90.0 THEN
```

```
        Message := 'Upper Limit Warning';
ELSE
        Message := 'Limits OK';
END_IF;
```

## INCLUDE

The INCLUDE statement incorporates statements from an external file when the structured text file is parsed. The external file can contain either Structured Text or Instruction List statements.

**Format**

INCLUDE FullFilePath;

Where:

FullFilePath — A string that specifies the path and file name of the external file.

**Example**

```
INCLUDE 'C:\ST_Files\ST_File1.TXT';
```

## FOR Statement

The FOR loop repeatedly executes (iterates) a statement list contained within the FOR…END_FOR construct. It is useful when the number of iterations can be predicted in advance, for example to initialize an array. The number of iterations is based on the value of a control variable which is incremented (or decremented) from an initial value to a final value by the FOR statement. You can use an expression for the control variable, but the value must be 0 or greater. By default, each iteration of the FOR statement increments the value of the control variable by 1. An optional BY portion of the construct can be used to specify an increment or decrement of the control variable by specifying a (non-zero) positive or negative integer or an expression that evaluates to a positive or negative integer.

The FOR statement checks the control variable before each iteration, and the statements within the FOR…END_FOR construct are only executed if the current value of the control variable has not exceeded the specified final value.

The END_FOR keyword causes the system to do an I/O scan at the end of each iteration of the FOR loop. Alternatively, you can use the END_FOR_NOWAIT keyword to loop without an I/O scan.

**Format**

```
FOR Int_Variable := Expression TO Expression [BY Expression] DO
                        Statement list;
END_FOR;
```

Where:

| | |
|---|---|
| Int_Variable | An integer variable. |
| Expression | A single value, expression, or complex expression of the same data type as Int_Variable. |
| Statement list | Any list of structured text statements. |

**Example 1**

The following code fragment initializes an array (of 100 elements) by putting 10 in all array elements. Since this operation is not dependent on I/O, the END_FOR_NOWAIT keyword is used.

```
FOR index := 1 TO 100
                    Array01[index] := 10;
END_FOR_NOWAIT;
```

**Example 2**

The following code fragment assigns the values of an I/O point to array elements over ten I/O scans. The latest entry is put in the array element with the smallest index. Since it is desired to perform an I/O scan after each loop, the END_FOR keyword is used.

```
FOR index := 10 TO 1 BY -1 DO
        ArrayInput[index] := Input01;
END_FOR;
```

## Function Call

The Structured Text function call executes a predefined algorithm that performs a mathematical, string, bit string or other operation. The function call consists of the name of the function followed by any required input or output parameters enclosed in parentheses.

**Function Format**

```
FunctionName(Parameter1, Parameter2, . . .); (*Un-named parameters*)
        or
FunctionName(P1:=Parameter1, Parameter1, . . .); (*Named parameters*)
```

**Example 1**

This code fragment shows the TAN function call.

```
Result := TAN( AnyReal );
```

**Example 2**

This code fragment shows a function with named parameters.

```
StringB := LEFT(IN:= StringA, L:=  VarI);
```

# LABEL

The label statement is used within SFC steps only. If an action label has an associated label statement in the SFC step, then the action does not run until the label is encountered.

**Format**

ActionLabel:: Statement;

Where:

| | |
|---|---|
| ActionLabel | The name of an action label. |
| Statement | Any Structured Text statement. |

**Example 1**

The following code fragment causes an action with a label of ActionLabel to start running.

```
ActionLabel:: VarString1:= "Action should be running";
```

# REPEAT Statement

The REPEAT loop repeatedly executes (iterates) a statement list contained within the REPEAT…END_REPEAT construct until an exit condition is satisfied. It executes the statement list first, then checks for the exit condition. This looping construct is useful when the statement list needs to be executed at least once.

**Format**

```
REPEAT
        StatementList;
UNTIL  BooleanExpression  END_REPEAT;
```

Where:

| | |
|---|---|
| BooleanExpression | Any expression that resolves to a Boolean value. |
| StatementList | Any set of Structured Text statements. |

**Operation**

The StatementList is executed. If the BooleanExpression is FALSE, then the loop is repeated; otherwise, if the BooleanExpression is TRUE, the loop is exited. The statement list executes at least once, since the BooleanExpression is evaluated at the end of the loop.

The END_REPEAT keyword causes the system to do an I/O scan at the end of every iteration of the REPEAT loop. Alternatively, you can use the END_REPEAT_NOWAIT keyword to loop without an I/O scan.

> **Note:** It is possible to create an infinite loop that (since the control system runs at the highest priority) does not return control to the operating system, especially when using the

END_REPEAT_NOWAIT keyword. Avoid infinite loops by insuring that the BooleanExpression provides a determinate exit condition.

**Example**

The following code fragment reads values from an array until a value greater than 5.0 is found (or the upper bound of the array is reached). Since at least one array value must be read, the REPEAT loop is used.

```
REPEAT
        Value:=Array01[Index];
        Index:=Index+1;
UNTIL  Value > 5.0  OR Index >= UpperBound END_REPEAT_NOWAIT;
```

## SCAN

The SCAN statement suspends program execution while an I/O scan takes place.

**Format**

```
SCAN;
```

**Example**

```
Statement1;
SCAN;          (*Statements 2 & 3 do not execute until after an I/O scan.*)
Statement2;
Statement3;
```

## WHILE Statement

The WHILE loop repeatedly executes (iterates) a statement list contained within the WHILE…END_WHILE construct as long as a specified condition is TRUE. It checks the condition first, then conditionally executes the statement list. This looping construct is useful when the statement list does not necessarily need to be executed.

**Format**

```
WHILE  BooleanExpression  DO
        StatementList;
END_WHILE;
```

Where:

| | |
|---|---|
| BooleanExpression | Any expression that resolves to a Boolean value. |
| StatementList | Any set of Structured Text statements. |

**Operation**

If BooleanExpression is FALSE, then the loop is immediately exited; otherwise, if the BooleanExpression is TRUE, the StatementList is executed

and the loop repeated. The statement list may never execute, since the Boolean expression is evaluated at the beginning of the loop.

The END_WHILE statement causes the system to do an I/O scan at the end of every cycle of the WHILE loop. Alternatively, you can use the END_WHILE_NOWAIT statement to loop back without an I/O scan.

> **Note:** It is possible to create an infinite loop that (since the control system runs at the highest priority) does not return control to the operating system, especially when using the END_WHILE_NOWAIT keyword. Avoid infinite loops by insuring that the BooleanExpression provides a determinate exit condition.

### Example

The following code fragment asserts an output I/O point as long as the input I/O point EndLimit remains FALSE. A WHILE loop is used since there is no reason to process the statement list if EndLimit is TRUE. The END_WHILE loop keyword is used since the I/O points should be scanned.

```
WHILE NOT EndLimit DO
        MoveForward := TRUE;
END_WHILE;
MoveForward := FALSE;
```

# Instruction List Programming

## Introduction

The Instruction List (IL) programming language is an IEC 1131-3 textual programming language. Its format is similar to an assembly language. You use the Instruction List editor to create stand-alone instruction list programs. The Instruction List editor is accessed from the Program Editor and features typical text-editing functions such as cut, copy, and paste, find, and replace. It also has tools and commands to automatically insert Instruction List statements and functions.

This section provides information on using the Instruction List editor writing Instruction List programs. It is assumed that you are familiar with general Program Editor operation and have some familiarity with the Instruction List language.

**Note:** Instruction List programs are continuously running programs (once each scan).

## Instruction List Editor Overview

**Note:** Refer to Customize Text Editor for information on setting tab, color, and font options for the editor.

### Opening an Instruction List Document

#### To open an existing document

- Select *Open Editor* from the Program Editor *File* menu and locate the document using the *Open* dialog box that appears.

#### To open a new document

- Select *New Editor* from the Program Editor *File* menu and choose *Instruction List Document* from the *New* dialog box that appears.

## Entering Instructions

### Manual Entry

Instructions can be entered by typing in the operator or function call and associated operands or parameters. Be sure to review the syntax, operators, and operator modifiers in **Language Overview** and the refer to the function call syntax in the **Language Reference**.

### Accessory Bar

The accessory bar shows commands in graphic form. It is an alternative method of entering IL statements. The accessory bar appears only when enabled by toggling *Accessory Bar* on the *View* menu. The following figure shows the accessory bar functions. The floating accessory bar can be undocked and positioned at the user's convenience.


### Insert IL Statements Menu

When the Instruction List editor is active, the Program Editor *Edit* menu has an *Insert IL Statements* item that lists instruction list statements (operators). Select the statement you need and it will automatically be entered at the cursor position.

### Insert IL Functions Menu

When the Instruction List editor is active, the Program Editor *Edit* menu has an *Insert IL Functions* item that lists standard functions that can be used with in the Instruction List language. Select the function you need and it will automatically be entered at the cursor position with the correct syntax. Replace any parameters with ones you have defined in the Symbol Manager.

Refer to **Language Overview** for more information on using functions and function blocks in the Instruction List language.

## Editing Instructions

The Instruction List editor supports the common editor functions such as cut, copy, and paste, and find and replace. These commands are found on the *Edit* and context menu.

## Bookmarks

Bookmarks allow you to quickly position the cursor at specified lines within the editor. You can set and reset bookmarks and locate previously set bookmarks.

### To set or reset a bookmark

1. Position the cursor at any line within the editor at which you wish to set a bookmark.

- Select *Toggle Bookmark* from the context menu or use the Ctrl-F2 key combination. When a bookmark is toggled on, a blue dot marks its position in the left-hand column of the editor.

### To position the cursor at a bookmark

- With one or more bookmarks previously set, press F2. The cursor will position (and the screen will scroll if necessary) at the bookmark. Successively pressing F2 will cycle through all bookmark positions.

## Printing

### To perform printer setup

- Choose *Print Setup* from the *File* menu. The standard Windows print setup dialog box appears. Make any changes and click *OK* to save the print setup information and continue.

### To view a print preview

- Choose *Print Preview* from the *File* menu or tool bar. The print preview is displayed.

### To print the instruction list file

- Choose *Print* from the *File* menu or tool bar. The standard Windows print dialog box appears. Make any changes to the print options and click *OK* to print the instruction list file.

## Saving

### To save editing changes

- Choose *Save* from the *File* menu or tool bar. The instruction list file is saved.

## Exiting the Editor

### To exit the editor

- Choose *Exit* from the *File* menu.

If there are any editing changes that need saved, a prompt appears requesting to save the changes first.

# Language Overview

## Instruction List Syntax

An Instruction List program consists of a list of instructions. Each instruction starts on a new line and can contain a label, operator, operator modifiers, operands, and comment fields as shown below:

| Label | Operator | Operand | Comment |
|---|---|---|---|
| Example: | LD<br>ADD | Sym01<br>Sym02 | (*Load Sym01 value into accumulator*)<br>(*Add Sym02 to it*)<br>(*The sum is now the current result*) |

Comments can only be at the end of the line. Blank lines are allowed between the instruction lines. The current result is maintained in an accumulator. Instructions work in the following manner:

current_result:= current_result OPERATOR operand

where the current_result is always to the left of the operator.

## Operators

The table below lists the Instruction List language operators.

| Operator | Modifier | Operand | Description |
|---|---|---|---|
| LD | N | | Set current result equal to operand. |
| ST | N | | Store current result to operand location. |
| S | Note 1 | BOOL | Set Boolean operand to 1. |
| R | Note 1 | BOOL | Reset Boolean operand to 0. |
| AND | N, ( | BOOL | Boolean AND. |
| OR | N, ( | BOOL | Boolean OR. |
| XOR | N, ( | BOOL | Boolean Exclusive OR. |
| ADD | ( | | Addition. |
| SUB | ( | | Subtraction. |
| MUL | ( | | Multiplication. |
| DIV | ( | | Division. |
| GT | ( | | Greater Than comparison (>). |
| GE | ( | | Greater Than or Equal To comparison (>=). |
| EQ | ( | | Equal To comparison (=). |
| NE | ( | | Not Equal To comparison (<>). |
| LE | ( | | Less Than comparison (<). |
| LT | ( | | Less Than or Equal To comparison (<=). |

| Operator | Modifier | Operand | Description |
|---|---|---|---|
| JMP | C, N | LABEL | Jump to label. |
| CAL | C, N | NAME | Function and function block call. |
| ) | | | Evaluate deferred operation. |

**Note**
1. Performed only if the current result value is Boolean 1.

Table Source: IEC 1131-3 (Part 3 of IEC Standard 1131 for programmable controllers), International Electrotechnical Commission.

### Modifiers

Operators can take the following modifiers:

**N** Boolean negation of the operand. For example,

```
ORN Bool1
```

is equivalent to result:= result OR NOT Bool1.

**C** The instruction is performed only if the current_result value is Boolean 1. (Or Boolean 0 if the N modifier is also used. For example,

```
LD    Value
      JMPC  Sort
```

is equivalent to "IF Value is TRUE, jump to Sort, else continue with execution."

**(** Defers evaluation of the operator until encountering a right parenthesis operator ")". For example,

```
MUL(  Num1
ADD   Num2
)
```

is equivalent to result:= result * (Num1 + Num2).

## Functions and Function Blocks

### Function Calls

Function calls can be done using the following forms:

CALC calls a function if the current result (accumulator value) is TRUE (1).

CAL always call the function.

CALCN only call the function if the current result (accumulator value) is FALSE (0).

Function call syntax using CALC is shown below:

**Function Example1:**

```
CALC    ROL(OUT:= VarBit, IN:= BitString, N:= RotateNum)
```

FunctionExample2:

```
CALC        ADD(OUT:= VarNum, Num1, Num2)
```

### Function Block

Function blocks are called using the following form of the call:

CAL always call the function block.

The following are examples of the syntax used for a standard function block, where ctu1 and ton1 are named instance of CTU and TON function blocks.

```
LD      10
ST      ctu1.PV
LD      TRUE
ST      ctu1.EN
LD      In3
ST      ctu1.CU
CAL     ctu1
LD      ctu1.Q
ST      Out3
```

```
LD      TRUE
ST      ton1.IN
ST      ton1.EN
LD      t#30s
ST      ton1.PT
CAL     ton1
LD      ton1.Q
ST      Out2
```

Refer to the **Language Reference** for standard function block descriptions and parameters (PV, EN, CU, Q, etc.).

### Accumulator Relationships

For function blocks, there is no relationship between the accumulator and the function block. The function block is always called, the accumulator is not passed into the function block, and after the function block returns, the accumulator has the same value as it had before the function block was called.

For functions the accumulator has no effect on the function inputs. When using CALC, the function will not be called if the accumulator is FALSE.

When using CALCN, the function will not be called if the accumulator is TRUE. When the function returns, if the return value is a BOOL, the return value is automatically loaded into the accumulator. If the function return value is a non BOOL, the return value can be saved into a variable using the func1 (OUT:=outvar1) syntax. In this case the return value is only saved into outvar1 if the function is actually called (CAL, CALC with TRUE accumulator, or CALCN with FALSE accumulator). If the function return value is a non-BOOL, the accumulator should be automatically loaded with the inverted value of the BOOL system error symbol (RTERROR). If the function is called and an error is flagged by the function, the accumulator will be loaded with a FALSE value after the function returns. If the function is called and no error is flagged by the function, the accumulator will be loaded with a TRUE value.

## Program Examples

The following program segments show examples of using the various Instruction List operators:

```
    LD      1                   (* LOAD  *)
    S       VarBool1            (* SET  *)
    R       VarBool2            (* RESET  *)

    LDN     VarTrue             (* LOAD NOT  *)
    S       VarBool1
    STN     VarBool2            (* STORE NOT  *)

    LD      VarBool1            (* AND  *)
    AND     VarBool2
    ST      VarBool3

    LD      VarBool1            (* AND NOT  *)
    ANDN    VarBool2
    ST      VarBool3


    LD      VarInt1             (* ADDITION OF INTEGERS  *)
    ADD     VarInt2
    ST      VarInt3

    LD      VarInt1             (* GREATER THAN  *)
    GT      VarInt2
    ST      VarBool1

    JMP     Label               (* JUMP  *)
    LD      55
    ST      VarNum

Label:      LD      100         (* LABEL  *)
    ST      VarNum

    LD      VarBool             (* JUMP CONDITIONAL  *)
    JMPC    Go
    LD      12
    ST      VarNum
Go:         LD      250

    LD      VarBool             (* JUMP CONDITIONAL NOT  *)
    JMPCN   Go2
    LD      2877
    ST      VarNum
```

```
        Go2:        LD          0

        (* Nesting* )
        (* Equivalent to the following:* )
        (* VarBool2:= VarBool1 AND (VarInt1 > (VarInt2 + VarInt3)) *)

        LD  VarBool1
            AND(        VarInt1
            GT(         VarInt2
            ADD         VarInt3
            )
            )
            ST          VarBool2

(* GENERAL OPERATIONS EXAMPLE  *)
            LD          Format          (* 0: CONVERT F TO C,  1: CONVERT C TO F  *)
            JMPC        CtoF            (* JUMP TO CORRESPONDING FUNCTION  *)

            LD          .555            (* LOAD THE MULTIPLIER  *)
            MUL(        Fahrenheit      (* THIS EVALUATES TO: (Farenheit-32)*.555)  *)
            SUB         32
            )
            ST          Celsius         (* STORE THE RESULT IN Celsius  *)
JMP         Done                        (* OPERATION COMPLETE, GOTO END OF FILE  *)
CtoF:       LD          32              (* LOAD OPERAND  *)
            ADD(        Celsius         (* THIS EVALUATES TO: (Celsius * 1.8)+32  *)
            MUL         1.8
            )
            ST          Fahrenheit      (* STORE THE RESULT IN Fahrenheit  *)

Done:       LD          1               (* CONVERSION COMPLETED  *)

            LD          Run_Timer       (* LOAD TRUE INTO THE REGISTER  *)
            ANDN        FALSE           (* THIS EVALUATES TO: TRUE AND NOT FALSE  *)
            ST          Enable          (* STORE IN Enable  *)

(* CALL TO FUNCTION BLOCK EXAMPLE  *)
            LD          Enable          (* LOAD Enable INTO THE REGISTER  *)
            ST          ton1.IN         (* STORE REGISTER VALUE IN ton1.IN  *)
            ST          ton1.EN         (* STORE REGISTER VALUE IN ton1.EN  *)
            LD          t#30s           (* LOAD t#30s INTO THE REGISTER  *)
```

```
ST      ton1.PT       (* STORE REGISTER VALUE IN ton1.PT  *)
CAL     ton1          (* CALL ton1, IT WILL RUN FOR 30s  *)
LD      ton1.Q        (* Load Output Boolean Into The Register *)
ST      Enable_Out    (* STORE THE REGISTER VALUE IN Enable_Out  *)
```

# Motion Programming

## Motion Programming Overview

The control system supports both tightly-coupled motion (integrated motion) and loosely-coupled motion (motion direct). One major difference in the two is: in integrated motion, you program motion in RS-274D or structured text; in motion direct, the programming is done in the card's native programming language. For more information refer to:

- Integrated Motion
- Motion Direct
- Refer to a the driver help file for additional information on a specific motion card.

## Motion Card Support

The following lists supported motion cards:

Delta Tau Data Systems, Inc.

- PMAC-PC
- PMAC2-PC
- PMAC2-PC-UltraLite
- PMAC/PMAC2-Lite
- Mini-PMAC

Motion Engineering, Inc.

- SL/DSP : 4-Axis Motion Card
- PCX/DSP: 8-Axis Motion Card

Compumotor

- AT6250 : 2-Axis Motion Card (No Steppers)
- AT6450 : 4-Axis Motion Card (No Steppers)

Galil Motion Control, Inc.

- DMC-1700 : 1 to 8-Axis Motion Card

## Hardware Setup of a Motion Control System

The following lists the general hardware requirements for motion control:

- IBM Compatible PC
- ISA Bus Motion Controller Card
- Cables (Ribbon or Fiber Optic or RJ-45)
- Breakout Boxes (May be Optically Isolated)
- Drives or Amplifiers
- Feedback Devices (Encoders, Resolvers, etc)
- Additional Accessories as Necessary

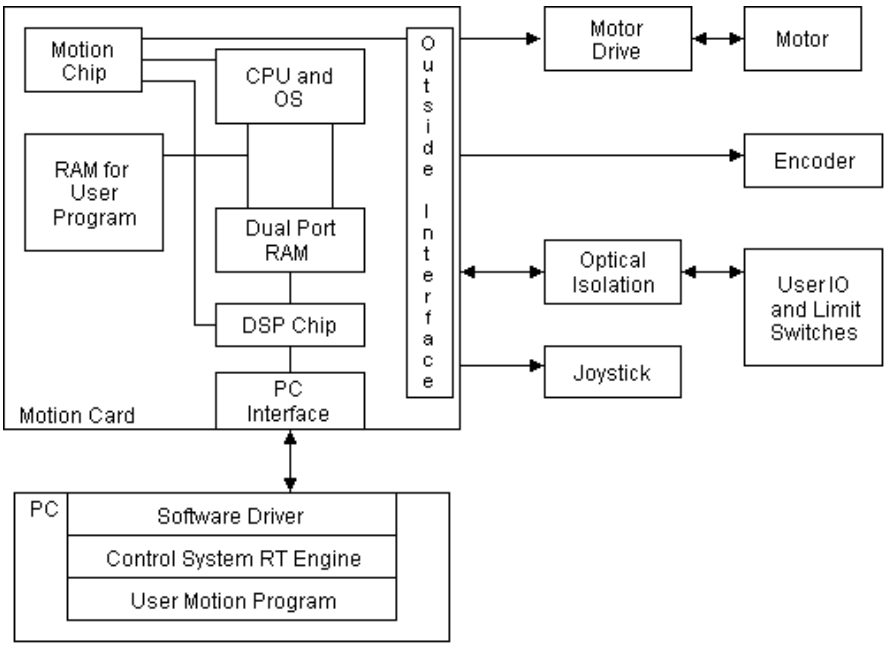
## Software Setup of a Motion Control System

The following lists general software requirements for motion control:

- Windows NT 4.0 Workstation with SP3 or Higher
- ASIC-200 ver. 3.10 or Higher
- ASIC-200 Motion Drivers
- Motion Controller Specific Software:
  - PEWIN32, P1Setup, P2Setup, PMACPlot (Delta Tau)
  - Motion Console (MEI)
  - Motion Architect (Compumotor)
  - WSDK, VB Toolkit, G-code-to-DMC, CAD-to-DMC (Galil)

## Architecture of a PC Based Motion Control System

Refer to the following figure for an example of a PC-based motion control system.




## Relative Roles of ASIC-200 and the Motion Card

| ASIC-200 | Motion Controller Card |
|---|---|
| • Interpret RS-274D and Structured Text Functions and Send Appropriate Commands to the Card<br>• *Does Not* Translate Motion Programs to Card's Native Language<br>• Implement G04 (Dwell)<br>• Send Passthrough Commands without Error Checking to the Card (G66) | • Execute Commands Sent by ASIC-200 (Host PC)<br>• Calculate Motion Trajectory<br>• Close Servo-loop for Position, Velocity, Torque, and/or Current Control<br>• Perform Commutation |

# Integrated Motion

## Integrated Motion Drivers

The following lists driver support for integrated motion:

- All PMAC, PCDSP, and Compumotor ISA Boards.
- Programming support includes Structured Text motion commands and the RS-274D language.

## Integrated Motion Features

The following lists integrated motion control features:

- Ability to synchronize moves with discreet logic.
- Ladder Logic actions using action qualifiers.
- Structured Text mixed with RS-274D.
- Better control over motion.
- Better error reporting.
- Ability to skip or abort motion.
- Ability to run NC Files.

## About the Motion Control Language

The Motion Control programming language is an RS-274D compliant set of text-based instructions that is designed for motion control operations. The language lets you design for 2- and 3-dimensional motions by using parameters such as:

- coordinate positions
- feed rates
- movement between positions with controlled acceleration and deceleration

Motion Control programming consists of a series of single-letter commands, which are followed by numerical parameters to these commands. The commands are organized into individual lines of text, which are called

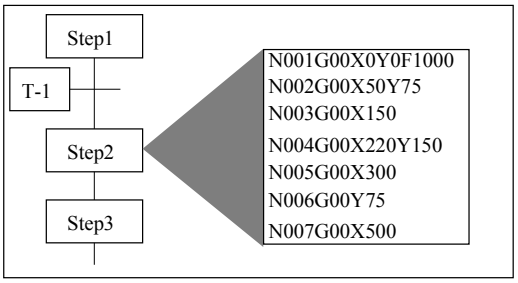
blocks. The blocks form execution units that are executed sequentially. Program execution pauses on each block until all the functions in the block are completed, then program flow continues with the next block.

You can add motion control to an SFC and Embed Structured text into motion control code.

For more information, see "Using Motion Control Statements" on page 137.

## Adding Motion Control to an SFC

When you create the application code for an SFC step, you can choose to use Motion Control code, as illustrated below. When the SFC is executed, the Motion Control code that you incorporate within each step is processed as the step becomes active. You can enter the Motion Control code directly into a step, or you can link a file containing the Motion Control code to the step when you configure the step. The type of information in the file should be in the same format as the type of information entered directly into a step.




For information about how the CONTROL SYSTEM software enhances the RS-274D specification, see "Software Enhancements to RS-274D" on page 136.

## Software Enhancements to RS-274D

The control system software provides the following enhancements to the RS-274D specification.

- You can embed Structured Text assignments and expressions within the Motion Control code. See "Embedding Structured Text into Motion Control Code" on page 158.
- You can use step actions to synchronize I/O operations with Motion Control execution. You can also use motion qualifiers to synchronize action logic with motion.
- The control system software provides motion symbols that are mapped to the I/O. For example, axis.CMDPOS is a real number symbol that contains the current commanded position of an axis.

- M flags are associated with the Motion Control M codes and are set on or off depending on the M code being used. This lets you monitor the M flags from other programs and allow other actions to begin after an M code is encountered during program execution.
- You can improve your control over program flow by using WHILE and IF-GOTO commands with the Motion Control commands.
- You can call a macro (subroutine) for execution by using a preparatory command G65.

# Using Motion Control Statements

## Using Motion Control Commands

The control system software converts RS-274D commands into the appropriate commands specifically generated for the servo-motion card that you have selected.

Motion commands must be structured in a specific format. See "Motion Control Block Format" on page 138. The motion commands are:

| Command | Function |
|---|---|
| A | Rotary A motion command in predefined engineering units. |
| B | Rotary B motion command in predefined engineering units. |
| C | Rotary C motion command in predefined engineering units. |
| D | Tool function for selection of tool compensation. |
| F | Feed rate function or speed command in predefined engineering units. |
| G | Preparatory function. See "Using G Codes" on page 140. |
| I | Interpolation parameter or thread lead parallel to X axis. |
| J | Interpolation parameter or thread lead parallel to Y axis. |
| K | Interpolation parameter or thread lead parallel to Z axis. |
| M | Miscellaneous function. See  on page 142. |
| N | Block number (optional) |
| S | Spindle speed function. |
| T | Tool Function. Range 01-32. |
| X | Linear X motion command in predefined engineering units. |
| Y | Linear Y motion command in predefined engineering units. |
| Z | Linear Z motion command in predefined engineering units. |

### *Motion Control Block Format*

The format for a Motion Control block (single line of code) is described below.

**N** If you use a sequence number, it must be first in the block. Optional for the control system.

**G** The preparatory function(s) G must follow N.

**X** The linear dimension words follow G. Specify the X axis first.

**Y** The linear dimension words follow G. Specify the Y axis second.

**Z** The linear dimension words follow G. Specify the Z axis third.

**A** The rotary dimension words follow G. Specify the X axis first.

**B** The rotary dimension words follow G. Specify the Y axis second.

**C** The rotary dimension words follow G. Specify the Z axis third.

**I** The interpolation words follow the dimension words. Specify the X axis first.

**J** The interpolation words follow the dimension words. Specify the Y axis second.

**K** The interpolation words follow the dimension words. Specify the Z axis third.

**D** The selection of tool compensation must follow K.

**F** If you specify a feed rate that applies to more than one axis, F must follow the last dimension word (and interpolation) to which it applies.

**T** The tool function selection follows S.

**M** Any miscellaneous function(s) that you specify must be last in the block, just ahead of the end of block character.

**End of Block** Indicate the end of a block with the carriage return / line feed character. Make sure that there are no extra spaces, tabs, or other characters between the last command and the End of Block.

### *Motion Control Block Examples*

#### Example 1

```
N009G01X-3.0Y-7.0Z+1.0F95
```

- N009 Ninth block in the program.
- G01 Positions a tool to the next point along a straight line path.
- Sets the X axis position.
- Sets the Y axis position.
- Sets the Z axis position.
- F95 Sets the feed rate at 95 units.

**Example 2**

```
N011G02X+0.5Y+1.0I0.75J0.0
```

- N011 Eleventh block in the program.
- G02 Positions a tool in a circular motion.
- Sets the X axis position.
- Sets the Y axis position.
- I0.75 Sets the X axis position for the arc.
- J0.0 Sets the Y axis position for the arc.

**Example 3**

```
N003G70X+1.3Y-7.0Z+2.1M08
```

- N003 Third block in the program.
- G70 Sets mode for programming in units of inches.
- Sets the X axis position.
- Sets the Y axis position.
- Sets the Z axis position.
- M08 Causes coolant number 1 to turn on.

## Using G Codes

The control system software supports RS-274D G codes to control many motion control operations. The G code support is driver dependent. Not all G codes are supported by all drivers. If a driver is marked with a bullet (•), it supports that G code.

The available G codes are listed in the following table:

| Code | Description | Notes | PMAC | PCDSP | Compu motor |
|---|---|---|---|---|---|
| G00 | Rapid point to point motion | 1) G00 motions must either have an axis.RAPID speed set or a feedrate on the block for the motion to occur.<br>2) Axes may not arrive at the endpoint at the same time. | • | • | • |
| G01 | Coordinated Linear motion | The speed of the motion is controlled and all the axes move in a coordinated manner. | • | • | |
| G02 | Circular Motion, Clockwise | The axes are moved in a coordinated circular motion clockwise when viewed from the top. | • | First two axis only | |
| G03 | Circular Motion, Counter-Clockwise | The axes are moved in a coordinated circular motion counter-clockwise when viewed from the top. | • | First two axis only | |
| G04 | Dwell | Execution of the program is suspended for a programmed length of time. The duration is specified by an F word in the same block. i.e. G04F2.5 = delay for 2.5 seconds. | • | • | • |
| G05 | Spline move | This is a PMAC style SPLINE1 MOVE. Refer to G05 Spline Move Notes for more information. | • | | |
| G09 | Exact Stop | This can be used to insure that axes moves are complete prior to evaluating a parametric expression. Example:<br>G01X10F100<br>G09<br>{Parametric Expression} | • | | |

| Code | Description | Notes | PMAC | PCDSP | Compu motor |
|---|---|---|---|---|---|
| G17 | XY plane selection | | • | | |
| G18 | ZX plane selection | | • | | |
| G19 | YZ plane selection | | • | | |
| G40 | Cancels Radius Compensation | | • | | |
| G41 | Cutter diameter compensation (left) | | • | | |
| G42 | Cutter diameter compensation (right) | | • | | |
| G43 | Tool Length Offset (Plus) | Offset is applied to the normal axis to the plane. | • | | |
| G44 | Tool Length Offset (Minus) | Same as G43 | • | | |
| G45 | Tool Offset Increase | Applies only if system is configured for “Lathe” setting. This setting is estblished from the axis group dialog in the config file. | • | | |
| G46 | Tool Offset Decrease | Same note as G45 | • | | |
| G47 | Tool Offset Double Increase | Same note as G45 | • | | |
| G48 | Tool Offset Double Decrease | Same note as G45 | • | | |
| G52 | Local Offsetting Coordinate Zero Point | Application Specific. | • | | |
| G53 | Motion in Machine Coordinate System | | • | • | |
| G54 | Workpiece Coordinate System 1 | | • | | |
| G55 | Workpiece Coordinate System 2 | | • | | |
| G56 | Workpiece Coordinate System 3 | | • | | |
| G57 | Workpiece Coordinate System 4 | | • | | |
| G58 | Workpiece Coordinate System 5 | | • | | |
| G59 | Workpiece Coordinate System 6 | | • | | |
| G61 | Exact Stop Mode | | • | | |
| G64 | Cutting Mode | | • | | |
| G65 | Macro Call | | • | • | • |
| G66 | Pass thru function | Driver specific | • | • | |

| Code | Description | Notes | PMAC | PCDSP | Compu motor |
|---|---|---|---|---|---|
| G70 | Inch Mode | Defined during configuration, not available during runtime. | Not in Runtime | Not in Runtime | |
| G71 | Metric Mode | Defined during configuration, not available during runtime. | Not in Runtime | Not in Runtime | |
| G90 | Absolute Positioning Mode | | • | • | |
| G91 | Incremental Positioning Mode. | | • | • | |
| G92 | Position Preset | Axis Commanded positions and Relative positions ( axis.CMDPOS ans axis.RELPOS) are set to the values specified in the block. | • | • | |
| G93 | Inverse Time Feed Mode | | • | | |
| G94 | Feed-Per-Minute Mode | | • | Feed-per-min only | |

## G05 Spline Move Notes

G05 supports a PMAC style SPLINE1 move. The syntax of G05 is as follows:

```
G05 F<time in milliseconds>
X100 Y200 A40 B90
X50 Y205 A60
```

While using G05, the following restrictions must be observed:

1. The G05 F<milliseconds> command must appear on a line by itself.
2. The G05 command can be terminated by simply going into any other mode (G00, G01, G02, or G03).
3. A move before SPLINE1 mode's first move and the move immediately after SPLINE1 mode's last move will NOT be blended. This is a PMAC limitation and cannot be resolved within ASIC-200.
4. It is highly discouraged to change the F<milliseconds> value in the middle of a spline move. It will not have any effect on the ASIC-200 program, but it will be interpreted as the next feedrate for a non-SPLINE1 move. For example, avoid the following:

```
G05 F5000
X300 Y100
F2000
X200 Y230
G01 X0 Y0 F30
```

5. In single step mode using a PRGCB, SPLINE1 moves will be executed like a linearly interpolated (G01 style) moves.

## Using M Codes

M Codes are user-defined operations supported in RS-274D and SFC. Valid M codes are M0 to M99. M codes execute in RS-274D and signal supporting logic in SFC or RLL programs to execute. For each M code, corresponding control system symbols Mflag0 to Mflag98 exists. Even numbered M codes turn off the M flags, while odd numbered codes turn on the corresponding flag. For example:

| RS-274D Statement | Result |
|---|---|
| M10 | Mflag10 set false |
| M11 | Mflag10 set true |
| M96 | Mflag96 set false |
| M97 | Mflag97 set true |

**Notes:**

1. Special purpose M codes must be on a line by themselves. This includes M00, M01, M03, M04, M05, M58, and M59.
2. M codes on a line with motion execute before the motion.

### *Predefined M Codes*

The control system software predefines several M codes for internal operations, these cannot be used to turn on or off M flags:

| Code: | Description: |
|---|---|
| M0 | Program Stop |
| M1 | Optional Program Stop |
| M2 | Program End |
| M3 | Spindle positive |
| M4 | Spindle negative |
| M5 | Spindle stop |
| M30 | Program End and Rewind |
| M99 | Macro Function End |

### *Wait and Continue M Code*

The control system software has the ability to wait on RS-274D M Codes. Once the M code processing is done, the control system software lets the application program wait for your logic to inform the system that the operation has completed and for execution to resume with the next sequential RS-274D block. To enable this feature, you must check the "Wait on All M Codes" box in the Motion Options Configuration Page.

These control system symbols apply to the Wait and Continue M codes:

| | | | | Cleared (C) by/ Set (S) by | |
|---|---|---|---|---|---|
| **Variable Name** | **Type** | **Use** | **Read (R) & Write (W)** | **Control S/W** | **User** |
| AxisGroup.MWAIT | BOOL | Waiting for M code processing | R | S/C | ____ |
| AxisGroup.MCODE | INT | Value of M code | R | S/C | ____ |
| AxisGroup.MCONT | BOOL | M code processing complete - resume execution | R/W | C | S |

## Using the Define M Flag Symbols Feature

If you want to the control system software to generate 46 global symbols of type BOOL, select the "Define M Flag Symbols" option box available in the config file. The symbols are generated when this option is selected and the configuration is saved and activated. These symbols appear in the Symbol Manager, and their symbol names are be Mflag6 through Mflag96, counting by even numbers.

## Using the Wait on All M Codes Feature

If you want you application program to suspend execution of RS-274D until your M code logic indicates that the action is complete, select the "Wait on All M codes" option box. The remainder of the program can then be executed. When an M code is executed, the motion program is suspended and the axis_group.MWAIT flag is activated until the application program sets the axis_group.MCONT flag, which causes the motion program to resume and the axis_group.MWAIT and axis_group.MCONT flags to be reset by the controller. The application programmer is responsible for supplying logic that sets the axis_group.MCONT flag after the axis_group.MWAIT flag is activated.

The axis_group.MCODE is an integer value the contains the value of the M code. The axis_group is the name of the axis group to which the M code is associated. Define axis group names in the project configuration file.

### Using the Do Not Process M Codes Feature

The control system software supports the special M code functionality as described in the RS-274D specification. Clicking this check causes the application program to ignore the following M codes:

- M3 (spindle positive)
- M4 (spindle negative)
- M5 (spindle stop)

## Predefined Integrated Motion Control Symbols

The control system software provides predefined symbols that you can use to monitor and control some of your motion control operations. You link the symbols to the appropriate functions when you configure the I/O.

The predefined symbols are grouped as follows:

- Axis Output Symbols
- "Axis Group Output Symbols" on page 146
- "Axis Input Symbols" on page 147
- "Axis Group Input Symbols" on page 148

**Note:** Not all predefined axis symbols are supported by all motion drivers. Check the motion driver help for supported axis symbols.

### Axis Output Symbols

| Symbol | Data Type | Function |
|---|---|---|
| axis.ACTPOS | real | Contains actual position of axis. |
| axis.ACTVEL | real | Contains actual velocity of axis. |
| axis.CMDPOS | real | Contains commanded position of axis. |
| axis.TPOS | real | Contains current move target position of axis. |
| axis.AXSFE | real | Contains following error of axis. |
| axis.A | Boolean | Indicates if axis is enabled. |
| axis.IP | Boolean | Indicates if axis is in position. |
| axis.MC | Boolean | Indicates if axis motion is complete. |
| axis.ESTPO | Boolean | Indicates if axis ESTOP is activated. |
| axis.RELPOS | real | Contains relative position of axis. |
| axis.HOME | Boolean | Indicates axis is at the HOME position. |
| axis.STATUS | real | Indicates axis status - bit definitions depend on driver. Refer to the driver help. |
| axis.CMDSPD | real | Contains commanded speed of axis. |

## Axis Group Output Symbols

| Symbol | Data Type | Function |
|---|---|---|
| AxisGroup.ESTPO | Boolean | Indicates if emergency stop (ESTOP) is activated. |
| AxisGroup.TOOL | integer | Contains active tool offset. |
| AxisGroup.INTPL | integer | Contains active interpolation mode for group.<br>0=point to point<br>1=linear<br>2=circular clockwise<br>3=circular counter clockwise. |
| AxisGroup.CIR | Boolean | Indicates if circular interpolation is activated (clockwise or counter clockwise). |
| AxisGroup.PTP | Boolean | Indicates if point to point interpolation is activated. |
| AxisGroup.LIN | Boolean | Indicates if linear interpolation is activated. |
| AxisGroup.CIRCW | Boolean | Indicates if clockwise circular interpolation is activated. |
| AxisGroup.CIRCCW | Boolean | Indicates if counter clockwise circular interpolation is activated. |
| AxisGroup.DWL | Boolean | Indicates if dwell is activated. |
| AxisGroup.DWLTIM | integer | Contains dwell time, counted down in ms. |
| AxisGroup.ENG | Boolean | Indicates if English (inches) units are activated. |
| AxisGroup.METRIC | Boolean | Indicates if metric units are activated. |
| AxisGroup.ABSDIM | Boolean | Indicates if absolute programming is activated. |
| AxisGroup.INCDIM | Boolean | Indicates if incremental programming is activated. |
| AxisGroup.WAIT | Boolean | Indicates program is suspended for tool change. |
| AxisGroup.CONT | Boolean | Indicates if program has resumed after tool change. |
| AxisGroup.MWAIT | Boolean | Indicates waiting for M code processing. |
| AxisGroup.MCODE | integer | Contains value of the M code. |
| AxisGroup.PLANE | integer | Active plane for circular interpolation.<br>0=G17<br>1=G18<br>2=G19 |

| Symbol | Data Type | Function |
|---|---|---|
| AxisGroup.CUTMOD | integer | Active cutting mode.<br>0=Cutting mode (G64).<br>1=Exact stop mode (G61). |
| AxisGroup.FEDMOD | integer | Active feedrate mode.<br>0=Feedrate programming IPM, DPM (G94).<br>1=Inverse time programming (G93). |

## Axis Input Symbols

| Symbol | Data Type | Function |
|---|---|---|
| axis.JP | Boolean | Causes jog in plus direction. See "Using the .JM and .JP Axis Input Symbols" on page 147. |
| axis.JM | Boolean | Causes jog in minus direction. See "Using the .JM and .JP Axis Input Symbols" on page 147. |
| axis.HLD | Boolean | Causes axis motion to stop until bit is reset. |
| axis.STP | Boolean | Causes axis motion to stop. |
| axis.JSPD | real | Sets jog speed for axis. |
| axis.JINCR | real | Sets jog increment for axis. |
| axis.JTYPE | integer | Sets jog type for axis (0=home, 1=cont, 2=incr). |
| axis.SOVR | real | Sets speed override for axis (100.0=100%). |
| axis.TOOLOFF[x] | real | An array of tool offsets for axis. x = 0-9. See "Using the .TOOLOFF Axis Input Symbols" on page 150.<br>(Not for MEI.) |
| axis.FIXOFF[x] | real | An array of fixture offsets for axis. x = 0-5.<br>See also, "Using the .FIXOFF Axis Input Symbols" on page 149.<br>(Not for MEI.) |
| axis.RAPID | real | G00 speed. |

### *Using the .JM and .JP Axis Input Symbols*

The axis.JP and axis.JM axis input symbols are Boolean symbols that cause an axis to jog in the plus and minus directions, respectively. These symbols have the following requirements:

- The axis.JP and axis.JM symbols are not functional until you assign values to the axis.JSPD and axis.JTYPE axis input symbols to specify the jog speed and type.
- The axis referenced by the axis.JP and axis.JM symbols must be either in a group with no other axes; or if other axes are in the group, they cannot

be under the control of another motion control program when a jog command is issued to the axis referenced by the axis.JP and axis.JM symbols.

**To send an axis home**

1. Set .JTYPE = 0.
2. Set .JSPD = homespeed.
3. Activate either .JP or .JM.
4. Wait until .HOME = TRUE for more than 1 second.
5. Reset .JP or .JM.

## Axis Group Input Symbols

| Symbol | Data Type | Function |
|---|---|---|
| AxisGroup.ESTP | Boolean | Activates emergency stop (ESTOP). The axis.ESTP axis input symbol is a Boolean that causes all defined axes to stop motion. Note that this includes all axes in all groups, regardless of which axis is referenced by the symbol. |
| AxisGroup.HLD | Boolean | Causes group motion to stop till bit is reset. |
| AxisGroup.STP | Boolean | Causes group motion to stop. |
| AxisGroup.SOVR | real | Overrides speed for group (100.0=100%). |
| AxisGroup.MCONT | Boolean | M code processing is complete. |
| AxisGroup.TOOLRAD[x] | real | Value used for tool radius compensation (G41 and G42). x = 0-31. For more information, see "Using the .TOOLRAD Axis Group Input Symbols" on page 149 |
| AxisGroup.TOOLLEN[x] | real | Value used for tool length compensation (G43 and G44). x = 0-31. For more information, see "Using the .TOOLLEN Axis Group Input Symbols" on page 150. |
| AxisGroup.RESTP | Boolean | Resets ESTOP until bit is reset.<br>NOTE: this input should only be pulsed on, if left on the system will not respond to other inputs.. |
| AxisGroup.OPSTP | Boolean | Activates optional stop (M01). |
| AxisGroup.TSTRUN | Boolean | Activates test run mode. |

### *Using the .FIXOFF Axis Input Symbols*

The preparatory commands G-55 to G-59 select fixture offsets. G-54 cancels fixture offsets. The axis.FIXOFF axis input symbols contain the offset values used by these commands. The relationship between the commands and the symbols containing their respective offsets is shown in the following example.

| G Command | Symbol |
|---|---|
| G-54 | axis.FIXOFF[0] |
| G-55 | axis.FIXOFF[1] |
| G-56 | axis.FIXOFF[2] |
| G-57 | axis.FIXOFF[3] |
| G-58 | axis.FIXOFF[4] |
| G-59 | axis.FIXOFF[5] |

Assign values to the array elements in a Structured Text Step that precedes the Motion Control Step containing the G commands.

### *Using the .TOOLRAD Axis Group Input Symbols*

The preparatory commands G-41 and G-42 apply cutter or radius compensation to the tool path. Both commands require the D command be used in the same block. The D command specifies which axisGroup.TOOLRAD[ ] element to select. The axisGroup.TOOLRAD axis group input symbols contain the actual compensation values used by the D commands. The relationship between the D commands and the symbols containing their respective compensation values is shown in the following example. Preparatory commands are G codes. For more information, see "Using G Codes" on page 140.

| D Command | Symbol |
|---|---|
| D-0 | axisGroup.TOOLRAD[0] |
| D-1 | axisGroup.TOOLRAD[1] |
| D-2 | axisGroup.TOOLRAD[2] |
| . | . |
| . | . |
| . | . |
| D-29 | axisGroup.TOOLRAD[29] |
| D-30 | axisGroup.TOOLRAD[30] |
| D-31 | axisGroup.TOOLRAD[31] |

Assign compensation values to the array elements in a Structured Text Step that precedes the Motion Control Step containing the G and D commands.

Code G40 cancels compensation.

### *Using the .TOOLOFF Axis Input Symbols*

This offset is applied if tool setting, as defined in the configuration axis group dialog window, is set to lathe.

The preparatory commands G-45 to G-48 select tool offsets. All four commands require the D command be used in the same block. The D command specifies which axis.TOOLOFF[ ] element to select. The axis.TOOLOFF axis input symbols contain the actual offset values used by the D commands. The relationship between the D commands and the symbols containing their respective offset values is shown in the following example.

| D Command | Symbol |
|---|---|
| D-0 | axis.TOOLOFF[0] |
| D-1 | axis.TOOLOFF[1] |
| . | . |
| D-8 | axis.TOOLOFF[8] |
| D-9 | axis.TOOLOFF[9] |

Assign tool offset values to the array elements in a Structured Text Step that precedes the Motion Control Step containing the G and D commands. Example:

```
{X.TOOLOFF[1] := 0.25}
{Y.TOOLOFF[1] := 1.1}
{Z.TOOLOFF[1] := 1.25}
G90F100
G45X1Y1Z1D1
X2Y2Z2
X5Y5Z5
M30
```

### *Using the .TOOLLEN Axis Group Input Symbols*

The preparatory commands G-43H and G-44H apply tool length offset to the tool path. The offset is applied to the normal axis to the plane. H specifies the which axisGroup.TOOLLEN element to select. The axisGroup.TOOLLEN axis group input symbols contain the actual compensation values used by H. The relationship between H and the symbols containing their respective compensation values is shown in the following example. Preparatory commands are G codes. For more information, see "Using G Codes" on page 140. Example

```
G17
G90G01
X0Y0Z0F100
{Machine.TOOLLEN[1] := 1.25 }
G43Z1H1F50
X5Y5Z5
M30
```

| D Command | Symbol |
|---|---|
| H0 | axisGroup.TOOLLEN[0] |
| H1 | axisGroup.TOOLLEN[1] |
| H2 | axisGroup.TOOLLEN[2] |
| . | . |
| . | . |
| . | . |
| H29 | axisGroup.TOOLLEN[29] |
| H30 | axisGroup.TOOLLEN[30] |
| H31 | axisGroup.TOOLLEN[31] |

Assign compensation values to the array elements in a Structured Text Step that precedes the Motion Control Step containing the G commands. G40 cancels compensation.

# Configuring Motion Options

The control system software contains these features that help you configure motion options:

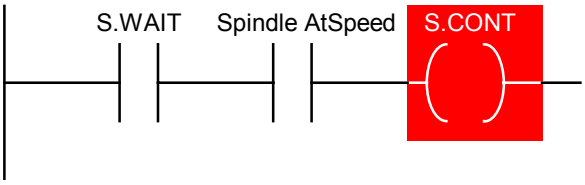

- "Using the Suspend on Spindle Commands Feature" on page 151
- "Using the Suspend on Tool Changes Feature" on page 152
- "Using the Define M Flag Symbols Feature " on page 144
- "Using the Wait on All M Codes Feature" on page 144
- "Using the Do Not Process M Codes Feature" on page 145


## Using the Suspend on Spindle Commands Feature

If you want an RS-274D program to suspend motion execution whenever spindle (S, M3, M4 or M5) commands are executed, select the "Suspend on Spindle Commands" option box. This option lets the application program suspend RS-274D execution until your application logic indicates that the spindle command has completed. The remainder of the program can then be executed. When a spindle command is executed, the motion program is suspended and the S.WAIT flag is activated until the application program sets the S.CONT flag, which causes the motion program to resume and the

S.WAIT and S.CONT flags to be reset by the controller. The application programmer is responsible for supplying logic that sets the S.CONT flag after the S.WAIT flag is activated and the spindle command has been completed.

**Example:**




### Using the Suspend on Tool Changes Feature

If you want an RS-274D program to suspend motion execution whenever a tool (T) command is executed, select the "Suspend on Tool Changes" option box. This option lets the application program suspend RS-274D execution until your application logic indicates that the tool change has completed. The remainder of the program can then be executed. When a tool change is executed, the motion program is suspended and the axis_group.WAIT flag is activated until the application program sets the axis_group.CONT flag; which causes the motion program to resume and the axis_group.WAIT and axis_group.CONT flags to be reset by the controller. The application programmer is responsible for supplying logic that sets the axis_group.CONT flag after the axis_group.WAIT flag is activated and the tool change has completed.

The axis_group is the name of the axis group to which the tool is associated. Define axis group names are defined in the project configuration file.

## Using Program Flow Control in Motion Applications

As an enhancement to the RS-274D specification, the control system software lets you use the WHILE and IF-GOTO commands with other Motion Control commands to enhance program flow control. See:


- "Using the WHILE Command" on page 152
- "Using the IF-GOTO Command" on page 153


### Using the WHILE Command

The commands within the WHILE loop are executed repeatedly until a **< Boolean expression >** evaluates to FALSE. Then, program flow continues with the next RS-274D command that follows the END_WHILE command.

### *WHILE Command Format*

WHILE (*Boolean_expression*) DO
    *RS-274_commands*
END_WHILE

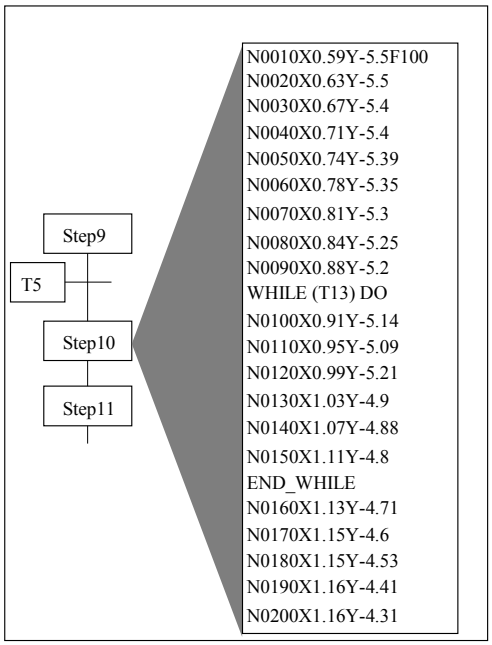
Where:

*Boolean_expression* is any Structured Text expression that evaluates to a Boolean value

*RS-274_commands* consists of normal RS-274D commands.

### *WHILE Command Example*

The following is an example of the WHILE command.




When step10 is active and T13 is TRUE, continuous motion pauses at the WHILE command. Blocks N0100-N0150 execute continually until T13 evaluates to FALSE. Then, block N0160 executes.

## Using the IF-GOTO Command

When program flow reaches the IF-GOTO command and the < Boolean expression > evaluates to TRUE, program flow continues with the RS-274D command corresponding to the value contained in < Block Number >. If the < Boolean expression > evaluates to FALSE, program flow continues with the RS-274D command following the IF-GOTO command.

**Format:**

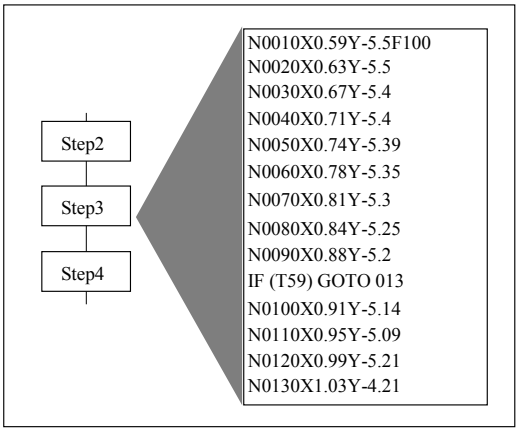
IF *Boolean_expression* IF-GOTO *Block_Number*

Where:

| | |
|---|---|
| *Boolean_expression* | is any Structured Text expression that evaluates to a Boolean value |
| *Block_Number* | is the line number of the next block to execute. The value in < Block Number > must match the line number of the RS-274D command exactly, including all leading zeroes and not including the `N` designator. |

### *IF-GOTO Example*

The following is an example of the IF-GOTO command.




When step3 is active, program flow continues at block N0130 when T-59 is TRUE. Otherwise, blocks 100-130 execute.

## Using the G65 Macro Calls with Motion

As an enhancement to the RS-274D specification, the control system software lets you call macros, sometimes called subroutines, by using the G65 command within RS-274D motion code. There are two simple steps to programming a G65 subroutine:

1. Design your macros; see "Designing the Macro" on page 154.
2. Call the macros for execution; see "Calling the Macro for Execution" on page 155.

### Designing the Macro

You can place the macro anywhere within your program. The G65 macro calling function can appear before or after the macro.

The structure of the macro has the following syntax:

```
O< x >
< RS-274 commands >
M99
```

Where:

| | |
|---|---|
| < x > | is an integer identifier for the macro to be called |
| < RS-274 commands > | is the macro code, consisting of normal RS-274D commands |
| M99 | is an M code block that declares the end of the macro |

The RS-274D commands between the *O*< *x* > identifier and the M99 code block are only executed when called from the G65 command. In other words, program execution skips over the code between the O-word block and the M99 block unless a G65 block has explicitly called it.

Here is an example of an .SFC program in the control system software that demonstrates G65 functionality:


## Calling the Macro for Execution

You call a macro within a G65 block. Use the following command syntax to call the macro for execution:

```
G65P< label >L< loop >
```

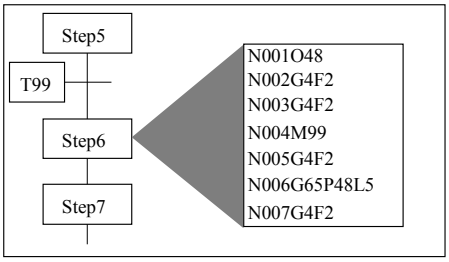
Where:

| | |
|---|---|
| < label > | is the integer identifier for the macro. This integer identifier must match the identifier in the O-word. |

< loop > is an integer that indicates the number of times to execute the macro

The < label > parameter must match the O< label > identifier in the macro. Both parameters < label > and <loop > must be specified. For example, if macro 22 were to be called one time, the syntax is: `G65P22L1`.

The following is an example of a macro call.




When step6 is active, program flow skips the first four blocks, which consist of the macro, and begins executing at block N005. Then block N006 is executed, which calls the macro (blocks N001-N004) a total of five times. After the fifth execution of the macro, program flow resumes at block N007.

# Monitoring and Running Motion Application Programs

This section describes how to use the Jog Panel, Single-Axis and Multi-Axis Status Panel, and how to monitor the axis plot.

> **Note:** These panels only display the first three characters of an axis name. They are also limited to the first eight axes.

## Using the Jog Panel

Use the Jog Panel to home and jog configured axes. The jog axis panel contains buttons that let you select the:

- jog axis
- jog type (home, continuous, or incremental)
- jog speed
- jog increment

The selected axis is displayed at the top of the Jog Panel and is jogged or homed in the desired direction by pressing and holding the jog+ or jog- button. If the jog+ or jog- button is released, the jog or home is cancelled. For incremental jogs, the axis stops when the increment is completed. To jog another increment, release and press the jog+ or jog- button. The Jog Panel is automatically set up for the number of configured axes.

The Jog Panel also displays:

- absolute position
- commanded position
- following error and velocity status for the selected axis

An axis fault indicator shows when a fault has occurred on the selected axis. When an axis fault occurs, the indicator turns red and the fault button is enabled. If no fault has occurred the indicator is green.

To view detailed information on the axis fault, press the fault button. When the axis fault is cleared the indicator turns green.

## Monitoring Axis Plot

Use the View\Axis Plot menu command to view axis information plotted with respect to time. This command activates a status box that plots selected status information for a selected axis over time. The selected axis and status information is displayed at the top of the status box.

To change the selected axis and/or status information, press the button with the desired axis and/or status information. Use the Zoom In, Zoom Out, Shift Up, Shift Down, Speed Up, and Slow Down buttons to position the axis plot to the desired location. Multiple copies of axis status plot can be activated at the same time.

## Using the Single Axis Panel

To view single action motion status, use the View/Single/Axis Status menu command. The Single Axis Status Panel displays the status for a single axis. The Single Axis Status Panel is automatically set up for the number of configured axes.

To display the status for an axis, select the axis by pressing the desired axis button on the panel. The title of the selected axis is displayed at the top of the panel.

The status displayed for the selected axis consists of:

- absolute position
- commanded position
- following error
- velocity

An axis fault indicator shows when a fault has occurred on the selected axis. When an axis fault occurs, the indicator turns red and the fault button is enabled. If no fault has occurred the indicator is green.

To view detailed information about an axis fault, press the fault button. When the axis fault is cleared the indicator turns green.

You can activate multiple copies of the single axis motion status to view complete axis status on more than one axis at the same time.

### Using the Multi-Axis Status Panel

To view multi-axis motion status, use the View\Multi-Axis Status menu command. The Multi-Axis Status Panel displays the specified status of all configured axes at the same time. The specific status that is displayed is selected from any of the following:

- absolute position (POS)
- commanded position (CMD)
- following error (FE)
- velocity (VEL)

Select desired status by pressing the specific status button on the panel. The title of the selected status is displayed at the top of the panel.

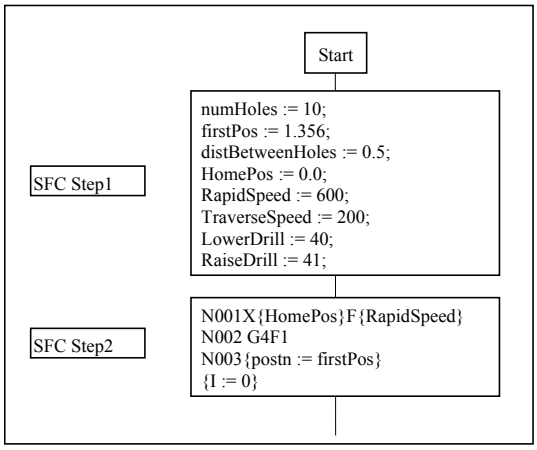
For each axis, a fault indicator indicates when a fault has occurred on that axis. If no fault has occurred the indicator is green. When an axis fault occurs, the indicator for that axis turns red, and the fault button for that axis is enabled.

To view detailed information on the axis fault, press the fault button. When the axis fault is cleared the indicator turns green. The Multi-Axis Status Panel is automatically set up for the number of configured axes.

Multiple copies of the multi-axis motion status can be activated at the same time.

## Embedding Structured Text into Motion Control Code

As an enhancement to the IEC-1131-3 specification, the control system software lets you embed assignments and expressions of the Structured Text language within Motion Control code. For example, in the following figure, block 1 of Step2 causes the X axis to move to *HomePos* at a speed of *RapidSpeed*. The symbols *HomePos* and *RapidSpeed* are assigned values in Step1 by Structured Text assignment statements. Blocks 3 and 4 of Step2 show examples of Structured Text assignment statements that have been embedded in Motion Control code.




## Guidelines for Embedding Structured Text in a Motion Control Step

Follow these guidelines when you embed Structured Text in a Motion Control step:

- Enclose all embedded Structured Text code within braces { } as shown in blocks 1, 3, and 4 of Step2 in the previous figure.
- On a block containing embedded Structured Text, the N command is optional.
- The semicolon, which is usually used to terminate Structured Text code lines, is also optional.
- You can use Structured Text as a parameter for all Motion Control commands except for the N and G commands.
- For example, N016 G4F{pause_time} is a valid block and pause_time is a Structured Text symbol that contains a duration for the F command.
- N100 G{prep_command} is not valid. The Structured Text symbol prep_command cannot contain the parameter for the G command.
- You can use either local or global symbols within the Structured Text expressions. They can be simple symbols or elements of arrays. You can set values within the program itself (Step1 in the figure) or download values using Dynamic Data Exchange (DDE).
- You can use assignment statements in separate blocks (Step2, blocks 3 and 4 in the figure).
- Three Structured Text function blocks are available for issuing motion commands from an SFC Structured Text step:
  - "AXSJOG" on page 161
  - "MOVEAXS" on page 161

- "STOPJOG" on page 161

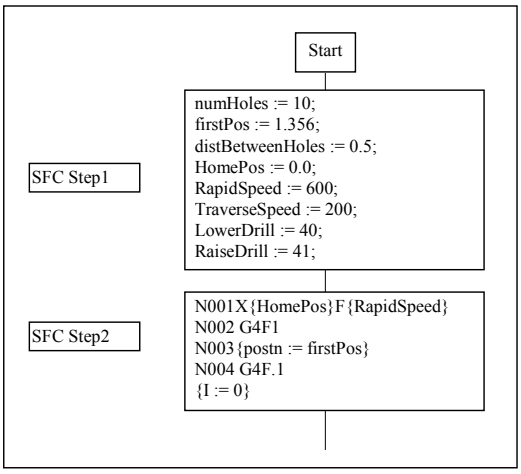
### How the Embedded Structured Text Code is Evaluated

When program flow encounters a block containing Structured Text, program execution and all motion activity pauses briefly. Program flow then continues with the next block that follows the Structured Text.

Expressions are evaluated at the time of execution (when the block is executed) and can represent the result of real-time inputs, such as sensor data. They are evaluated as integers or real numbers as appropriate for the Motion Control word.

Because the control system software has a "read-ahead" capability, Structured Text assignment statements are read and executed before program flow encounters the block containing them. You can control the execution of a Structured Text assignment by placing a G04 command in the block that precedes the block containing the assignment. The "read-ahead" feature does not extend beyond a G04 command.

For example, in the following figure, the assignment statement in the fifth block of Step2 is not executed until program flow encounters the statement. Without the G04 command in the fourth block, symbol I may be set to zero before the second or third block is executed.




## Structured Text Motion Functions

These Structured Text Motion Functions are available:

- AXSJOG
- MOVEAXS

- STOPJOG

## AXSJOG

The AXSJOG function starts a jog command on a specified axis.

**Format:**

AXSJOG(*axis*, JOGDIR:= *direction* , JOGTYPE:= *type*, JOGSPD:= *speed*, JOGDIST:= *distance*);

Where:

| | |
|---|---|
| *axis* | is the axis name (X, Y, Z, etc.) |
| *direction* | is the jog direction (JOGPLUS or JOGMINUS) |
| *type* | is the type of jog (JOGCONT for a continuous job, JOGINCR for an incremental position jog, or JOGHOME for a home position |
| *speed*, *distance* | are real numbers, symbols, or expressions that resolve to real number data types and specify the jog speed and distance. |

**Example:**

AXSJOG (X, JOGDIR:=JOGMINUS, JOGTYPE:=JOGCONT, JOGSPD:=10.0);

## MOVEAXS

The MOVEAXS function starts a move command on a specified axis.

**Format:**

AXSJOG(*axis*, POSTN:= *position*, VEL:= *velocity*, ACCEL:= *acceleration*);

Where:

| | |
|---|---|
| *axis* | is the axis name (X, Y, Z, etc.) |
| *position*, *velocity*, *acceleration* | are real numbers, symbols, or expressions that resolve to real number data types and specify the move position, velocity, and acceleration. |

**Example:**

MOVEAXS (X, POSTN:= positionA, VEL:= velocityA, ACCEL:=10.0);

## STOPJOG

The STOPJOG function stops a jog command on a specified axis.

**Format:**

STOPJOG(*axis*);

Where:

*axis* is the axis name (X, Y, Z, etc.).

**Example:**

STOPJOG (Z);

# Motion Direct

## Motion Direct Overview

**Motion Direct Notes:**

1. For every motion direct program that you write in the motion cards's native language, you can (and should) define 3 bits in the card configuration:
   - Download program
   - Start program
   - Stop program

   You can use these as coil symbols in RLL or set them from structured text commands. If you do not define them, then you do not have control over the motion program.
2. Feedback variables are available. Refer to Predefined Motion Direct Symbols.
3. You have the ability to look at registers and read parameters on the motion card. To do so, you define symbols corresponding to the registers.
4. Depending on the motion card, you can define the I/O in the configuration if the card supports or maps symbols to registers on the card.
5. The control product drivers do not check for any errors in the motion program (because the motion program is written in the card's native language). The driver may generate a log file (located in the project folder) and may also set a bit indicating there is a parse error.

## Motion Direct Driver Support

- All PMAC boards.
- Galil's DMC-1700 board.

- Programming and debugging in the motion card's native language.
- Ability to download, start, and stop programs.
- Comprehensive feedback in ASIC-200.

## Motion Direct Features

- No need to learn a new language or know the idiosyncrasies of a motion card.
- Use all the features of a card rather than artificially limiting them from control system.
- Use existing programs in the card's native language.
- Provide a higher level control with ASIC-200.

## Predefined Motion Direct Symbols

The control system software provides predefined symbols that you can use to monitor and control some of your motion control operations. You link the symbols to the appropriate functions when you configure the I/O.

**Note:** Not all predefined axis symbols are supported by all motion drivers. Check the motion driver help for supported axis symbols.

### Axis Output Symbols

| Symbol | Data Type | Function |
|---|---|---|
| axis.ACTPOS | real | Contains actual position of axis. |
| axis.ACTVEL | real | Contains actual velocity of axis. |
| axis.CMDPOS | real | Contains commanded position of axis. |
| axis.AXSFE | real | Contains following error of axis. |
| axis.A | Boolean | Indicates if axis is enabled. |
| axis.IP | Boolean | Indicates if axis is in position. |
| axis.MC | Boolean | Indicates if axis motion is complete. |
| axis.HOME | Boolean | Indicates axis is at the HOME position. |
| axis.CMDSPD | real | Contains commanded speed of axis. |

## Axis Group Input Symbols

| Symbol | Data Type | Function |
|---|---|---|
| AxisGroup.SOVR | real | Overrides speed for group (100.0=100%). |
| AxisGroup.RESTP | Boolean | Resets ESTOP until bit is reset.<br>NOTE: this input should only be pulsed on, if left on the system will not respond to other inputs.. |

# Running Application Programs

## Runtime Subsystems

The runtime subsystems consist of the Program Manager, Program Execution, I/O Scanner, and the Event Log subsystems. The Run Time icon visually represents the Program Manager, Program Execution, and the I/O Scanner subsystems. The Event Log has its own icon.

The Program Execution and I/O Scanner subsystems are given real-time process priority, which means they are given CPU time before all normal applications as well as mouse update and disk access.

**To start the runtime subsystems, do one of the following**

- Select *Startup Runtime Subsystems* from the Program Editor or Operator Interface Editor *Execute* menu.
- Select *Runtime Engine* from the *ASAP Applications* menu on the Windows *Start* menu.

## Running an Individual Program

To run a program, you must be running under the Windows NT operating system and have the Runtime Subsystems installed and active. If the Runtime Subsystems are not active, a startup prompt for the runtime subsystems appears.

### Running the Active Program

**To run the active program**

- Select *Run* from the Program Editor *Execute* menu.

After an RLL or SFC program begins running, the program display will highlight using the active highlight colors to show the status of the running program.

**To run the active file with debug enabled**

- Select *Run with Debug* from the Program Editor *Execute* menu.

#### To run one step of the active file

- Select *Single Step* from the Program Editor *Execute* menu.

If the step contains more than one command line, the next command line within the step is executed.

#### To run the active file with restart

A program running with restart will automatically start running when the Runtime Subsystems start up.

- Select *Run with Restart* from the Program Editor *Execute* menu.

### Canceling a Running Program

#### To cancel a running program

- Select *Abort* from the Program Editor *Execute* menu.

The program is cancelled and reset to the start of the program.

> **Note:** To cause a program break point in an SFC step, use the Structured Text BREAK function in that step.

## Configuring Programs to Execute Automatically

In some applications, it is necessary to start an SFC program running automatically every time the controller is powered up or every time the Runtime Subsystems are started. You can accomplish the first option by making a batch file with the appropriate commands and adding the batch file to the Windows NT Startup folder. The second option is accomplished using the *Run with Restart* command.

### Starting Programs with a Batch file

> **Note:** Only one SFC application program can be started using this method.

1. From Windows NT, launch a text editor to create a text file. Once created you will need to save the file with a .BAT extension. Enter the following two lines in the text file:

   ```
   start c:\asic\bin\runtime.exe/RUN c:\asic\MyProject\Main.sst
   start c:\asic\bin\oicfg.exe
   ```

   Make sure that the /RUN command above is entered in all CAPITAL letters.
2. Edit the two lines to make sure the path to the \bin directory is correct for your machine.

3. Edit the first line to make sure that the path to your project folder and the name of the SFC program is correct. Use an .SST extension for the SFC file instead of .SFC because this is the Structured Text version of the SFC file that the Runtime compiler needs (instead of the binary SFC file that the Program Editor requires).
4. Using the Program Editor, make sure that the current project and active configuration are correctly selected for this application. Parse the SFC file to make sure that the corresponding .SST file exists and is current.
5. Use the Operator Interface Editor to select the appropriate .OPI file to use at power up.
6. Follow the instructions in your Windows NT help files to add the batch file to your NT Startup folder.
7. Reboot your system to test the configuration.

### Starting Programs with the Run With Restart Command

#### To run the active file with restart

- Select *Run with Restart* from the Program Editor *Execute* menu.

A program running with restart will automatically start running when the control system software Runtime Subsystems start up. A program that has been Aborted or is Faulted will no longer be marked to run with restart.

After an SFC or RLL program begins running, the program display will highlight using the active highlight colors to show the status of the running program.

## Monitoring Power Flow

When an RLL or SFC program is running, the program display will highlight using the active highlight colors to show the status of the running program.

### Active RLL Programs

When an RLL program is running, or an embedded action and/or RLL transition program window is open in a running SFC program, the contacts, coils, and function blocks in the program will begin highlighting. A normally open contact will highlight when the BOOL symbol attached to it is TRUE. A normally closed contact will highlight when the BOOL symbol attached to it is FALSE. A positive transition sensing contact will highlight when a positive transition occurs on the attached symbol. A negative transition sensing contact will highlight when a negative transition occurs on the attached symbol. All output coils will highlight when the attached BOOL symbol is TRUE (coil highlighting reflects the state of the coil symbol). All function blocks will highlight when the function block is active.

**Note:** If the BOOL symbol attached to a contact is TRUE the contact will highlight regardless of its position on a rung. In other words, a contact being highlighted does not necessarily mean the other contacts in front of it are also TRUE.

### Active SFC Programs

When an SFC program is running, any active steps and/or transitions will be highlighted. If an active step is displayed with motion/process commands an active command indicator will be displayed to the left of the command that is currently executing.

## Viewing the Status of Application Programs

**To view program status**

- Select *Program Status* from the *View* menu.

The *Program Status* window is displayed with a list of active programs and their status.

**To view the program**

- Double-click on a program.

**To issue a program command**

- Select the desired program from the list box and push the button with the desired program command: *View*, *Run*, *Abort*, *Stop*, or *Step*.

## Monitoring and Testing Application Programs

### Parsing a Program

In order to parse a program you must be running under the Windows NT operating system.

**To parse the active file**

- Select *Parse* from the Program Editor *Execute* menu.

If any parse errors are encountered, the *Parse Errors Window* appears with the list of error messages. You can double-click on the error in the *Parse Errors Window* to display more information about the error or to highlight the location of the error in the program. Once the program parsing is complete the active file window title is updated with the parsing status.

### Watching and Forcing Symbols

The Watch window displays local and global symbols and their status at runtime.

### To display or hide the Watch window

- Select *Watch/Force Variables* from the *View* menu or click the Watch window icon  .

At runtime, this window displays variables and their current values. You can specify (force) new values for them to help in debugging your program.

**Note:** In the RLL editor, the force state of contact and coil symbols are indicated by colorization (depending on whether the contact or coil is also selected). Default colors are:

Light purple/dark purple – Forced to 0.

Light blue/dark blue – Forced to 1.

## Run with Debug

The Run with Debug program option lets you to execute an SFC program continuously until a BREAK statement is encountered. When the BREAK statement is encountered the program is stopped. From that point you can single step or abort the program.

## Single Stepping a Program

The Single Step Program option lets you to execute an SFC program one complete scan at a time. The program that you are testing must be stopped before you can step it. The mode of any other programs in the project does not matter.

### To Step a Program

1. Select the program in the Program Editor. If the program is running, stop it by selecting *Stop* from the *Execute* menu.
2. Open a Watch window for the program and add any symbols that you need to monitor as the program runs. This step is optional.
3. Select *Single Step* from the *Execute* menu. The program enters the Run mode for one scan, then enters the Break mode. Any I/O controlled by the program is updated during the program scan.
4. Continue single-stepping as long as needed.

## Clearing Fault Mode and Error Conditions

*Reset Estop and clear I/O faults* from the *Execute* menu clears the emergency Estop and any I/O faults. In a system with motion, the axis feedback loops are closed and the commanded position for all axes is set to the current position.

# Program Operation Overview

## Activate Configuration

Activating a configuration causes the following actions:

- Initialize global memory.
- If outputs are configured disabled, enable interface sets outputs to disabled state (force won't work) and starts scanning.
- If outputs are configured enabled, sets default state and starts scanning.

## First Scan with Active Configuration

The following occurs on the first scan:

- Local and function block variables are initialized.
- RT_FIRST_SCAN set high for first scan of first running program.
- Logic is solved.
- I/O is updated and the sequence repeats.

## Power-Down Sequence

The following occurs on the last scan :

- The program does not receive any advance notification of an impending shutdown and cannot take any specific action.
- Interface modules set disable state of I/O according to individual design.

## Normal Operation

The following occurs during normal operation:

- All actions associated within a step are solved once each scan a step is active (depends on the action qualifier).
- Actions are solved before the structured text within a step is solved.

- If a transition associated with a step becomes TRUE, the I/O is updated again after which all the appropriate actions (not pulsed, or inactive delayed, or limited (inactive)) are solved by turning off all output coils except latched coils. Function blocks are not solved. The I/O is updated and then the next active steps are solved the following scan.
- If structured text is used within a step, then the transition is not evaluated until all structured text statements have been completed.
- All structured text statements within a FOR loop are executed one iteration per scan unless a NOWAIT statement is encountered which causes the loop to complete all iterations in a single scan.
- All structured text statements in a step are executed once during the first scan the step is active and are not repeated during subsequent scans.
- All statements within a WHILE loop will continue to be executed until the WHILE condition becomes FALSE.
- The normal execution sequence is: update I/O, solve logic, then repeat.

## Initialization of Variables

- Inputs cannot be initialized; they take their value from the hardware.
- Global memory symbols are initialized when the configuration is activated or re-activated.
- Local memory symbols are initialized each time the program is executed.
- Function block inputs are initialized as local or global symbols according to the type of symbol assigned.
- Function block outputs cannot be written by the user.
- Function block internal variables are local symbols and are initialized as such.

## Program Execution Order

- There is no guaranteed order of program execution; the user must assume simultaneous execution.
- SFC branches: In a simultaneous diverge there is not guaranteed order; the user must assume simultaneous branching.
- SFC actions: There is no guaranteed order; the user must assume simultaneous execution.

**Note:** Refer to On-Line Editing Operation for information on how on-line editing affects program operation.

# Transferring Project to RunTime

## Transferring Project to RunTime Procedure

Depending on your circumstance, you may develop and run your application on the same industrial computer, or you may develop on one computer and wish to transfer the files to another computer to run and operate your application. The following procedure discusses how to do this.

**Note:** Please read this entire procedure and make sure you understand and have all the files you need to transfer before attempting to transfer a project.

**Licensing Note:** You can license your runtime system either before or after you transfer your project. Be aware that if you license your runtime system, and you need to access the development mode, you must use the Temporary Authorization to do so. (The license provides 30 minutes of Temporary Authorization. Additional Temporary Authorization can be purchased.) You must access the development mode to activate the configuration and only if you install the project to a different path than on your development system. Typically, this takes only a minute. However, if the system is unlicensed, you can start in Demo mode to perform this.

**Active X Controls Note:** If your operator interface contains Active X controls, you must have the Active X controls installed and registered on the target before opening the OPI containing them, otherwise the ActiveX controls will be removed from the OPI file when it is opened.

**Note:** Refer to File Types for a complete list and description of project files.

### To transfer your project files

1. All the files you need to transfer are in your project directory. You first need to determine your transfer method: floppy disk, network, or other means.

   **Note:** If you use a (recordable) CD-ROM, be aware that the copied files may be marked read-only when transferred to the target computer.

2. Current project information is stored in a data file (**active.DAT** in the **/bin** folder). If you are transferring project files to another path (for example, from drive D: on the development system to drive C: on the

runtime system), you must account for this. Refer to the following table for files to transfer.

| | Same Path | Different Path |
|---|---|---|
| **New Project Transfer** | Transfer the entire project folder. | Transfer only the source files (SFC, RLL, ST, CFG, OPI, user files) and reparse. |
| **Existing Project Transfer (i.e., update a file)** | Copy over the source and parsed files (e.g., SFC and SST for an SFC program). (Otherwise, you will be out of sync and the system will think you are in on-line edit mode.) | Copy only the source file (and CFG if global symbols have changed) and reparse. |

3. After determining which files to copy and the method, use Windows Explorer to copy the files to the floppy disk or network destination. If using a floppy disk to copy the files, and the total file size is too large, you can use a file compression utility.
4. At the runtime station, copy the files to the local hard disk (or other media) in a project directory.
5. If you have installed your runtime license, you can start the program editor, activate the configuration, and run your programs (and use your operator displays if you have any developed).

Refer to Managing Projects, Activating a Configuration, and Application Programs if you need more information on these topics.

## Installing to a Different Path

If you have installed to a different path (and copied only the source files), you will need to use the temporary authorization to open the project and activate the configuration (since without the **DAT** file, this information has been lost):

1. After starting the Program Editor, select *Temporary Authorization* from the *License* menu. You can then open the project and activate the configuration, etc. This typically only takes a minute.
2. Be sure to reset to the original authorization using *Reset Original Authorization* from the *License* menu.

Alternatively, if you have not installed the license, you can use demo mode.

Refer to the **Getting Started** manual for complete information on Demo Mode, Temporary Authorization, and license information in general.

**Note:** If you have not already licensed the run-time system, you must do so. Refer to the Getting Started manual or the help file for licensing information.

## Other Considerations

You may also wish to do the following:

1. Setup password access level codes. You can create the codes as described in the Getting Started manual, or if you have an existing password code file (in *installationfolder***bin/password.dat**) you can copy it to the runtime only system *installationfolder***bin** folder.
2. Secure HMI access. Refer to the HMI Guide for information.
3. Backup the project, refer to Backup and Restore Project.

# Backup and Restore Project

## Backup and Restore Project Overview

You should backup your project so that in the event of accidental erasure or disk failure, you can quickly restore and begin running your control application.

**Note:** Refer to File Types for a complete list and description of project files.

## Project Backup

To backup your project, you need to copy the following files to a secure location:

Your project folder and all files

*installationfolder*/bin/active.dat

*installationfolder*/bin/oiactive.dat

*installationfolder*/bin/password.dat

*installationfolder*/bin/peactive.dat

Where *installationfolder* is typically c:/program files/ASIC-200. A secure location means a diskette or other removable media or network location. If needed, a file compression utility can be used. For file descriptions, refer to File Types.

In addition, you may have other files that are needed in the application, such as ActiveX controls, that should be backed up as well.

## Project Restore

To restore a project, copy the backed-up files to the their original location.

**Note:** In the event of a hard disk failure, you may also lose the product license and need to reinstall the software. Refer to the installation instructions to re-install the software. Contact your vendor for information on restoring a license.

If you did not back up the **dat** files in *installationfolder*/bin, you must do the following:

1. Re-enter the password level codes.
2. Select the active project. (To do so, you must use Temporary Authorization on a run-time only system.)
3. Select the active **OPI** file.

If you have saved only your source files (**CONFIG**, **OPI**, **RLL**, **ST**, **SFC, IL**) you will need to parse each program.

# File Types

## File Type Descriptions

The following table lists and describes the file types you may find in the project directory.

| File | | Created By | Notes |
|---|---|---|---|
| **Configuration Files** | | | |
| _gprog.bbn | Binary configuration file | Parser | |
| _gprog.st | Structured text version of configuration file | Parser | |
| CFG | Configuration file containing global symbols and IO data | User | 2, 3 |
| **DAT Files** | | | |
| active.DAT | Contains the current active project | Editor | 1, 3 |
| environ.DAT | Environment information (active configuration) | Editor | 3 |
| oiactive.DAT | Contains the current active OPI file | Editor | 1, 3 |
| oienv.DAT | HMI open document information | Editor | 3 |
| password.DAT | Contains password data | Editor | 1, 3 |
| peactive.DAT | Editor settings | Editor | 1, 3 |
| peenv.DAT | Editor open document information | Editor | 3 |
| **Instruction List Files** | | | |
| BBN | Binary file executed by the runtime engine | Parser | |
| IL | Source Instruction List file | User | 2, 3 |
| ILT | Structured Text version of the IL program | Parser | |
| ILX | Online edit cancel information | Parser | |
| IWC | Names of symbols added to watch window | Editor | |
| **Operator Interface Files** | | | |
| OPI | Operator Interface File | User | 2, 3 |

| File | | Created By | Notes |
|---|---|---|---|
| **Relay Ladder Logic Files** | | | |
| RBN | Binary file executed by the runtime engine | Parser | |
| RLL | Source RLL file | User | 2, 3 |
| RLR | Redo info for RLL file | Editor | |
| RLX | Online edit cancel information | Editor | |
| RLU | Undo info for RLL file | Editor | |
| RST | Structured Text version of the RLL program | Parser | |
| RWC | Names of symbols added to watch window | Editor | |
| SAVERLX | Temporary file used for on-line edit | Editor | |
| SAVERST | Temporary file used for on-line edit | Editor | |
| **Sequential Function Chart Files** | | | |
| SAVESFX | Temporary file used for on-line edit | Editor | |
| SAVESST | Temporary file used for on-line edit | Editor | |
| SBN | Binary file executed by the runtime engine | Parser | |
| SFC | Source SFC file | User | 2, 3 |
| SFR | Redo info for SFC file | Editor | |
| SFU | Undo info for SFC file | Editor | |
| SFX | Online edit cancel information | Editor | |
| SST | Structured Text version of the SFC | Parser | |
| SWC | Names of symbols added to watch window | Editor | |
| **STructured Text Files** | | | |
| BBN | Binary file executed by the runtime engine | Parser | |
| ST | Source Structured Text file | User | 2, 3 |
| STT | Structured Text version of the ST program | Parser | |
| STX | Online edit cancel information | Editor | |
| TWC | Names of symbols added to watch window | Editor | |

**Notes:**

1. Located in bin/ folder. All other files are in the project folder.
2. These are the files that you create. You should never attempt to edit any of the other files.

- These are the files that you should back up to protect your work.

# Trace

## Trace Overview

The Trace utility allows you to log the changes in value of a symbol and graph these changes in real time.

## Logging Symbol Data

The first step in tracing is to select symbols and begin logging their data values.

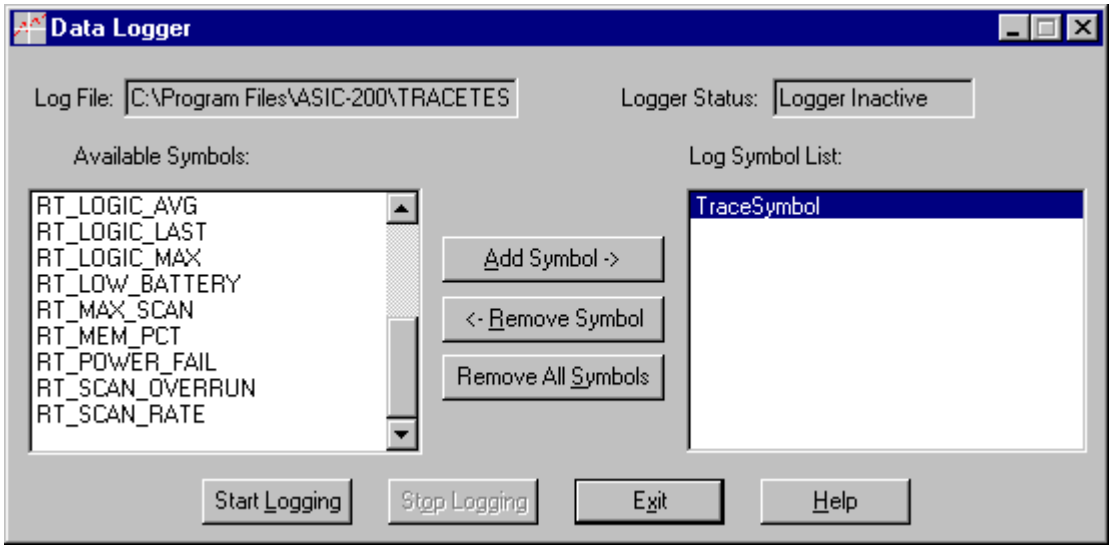
### To log symbol data

1. Activate your configuration.
2. Start the runtime system.
3. Start the Data Logger by selecting T*race Logging* from the *Tools* menu. The Data Logger screen appears similar to the following figure.

- Refer to the table for field descriptions.
- When you have selected the symbols you would like to log (and graph), click *Start Logging* to begin.

  **Note:** When you start logging, the Data Logger icon appears next to the Runtime icon in the system tray. You can click on the icon (with the secondary mouse button) for menu items.
- The symbol data is written to a log file. To view the graph, refer to Trace Symbol Data.

  **Note:** The Data Logger uses a maximum of ten, 100 KB files (1 MB total disk space). Each time you start the Data Logger or when a file reaches the 100 KB limit, a new log file is created (or overwritten). When the tenth file reaches the 100 KB limit, the first file is overwritten with new data, and so on. The log file is in the same path as your configuration file, but has a **log** extension. As each new log file is created, the log file name is appended with a numeral (config1.log, etc.).




| Field | Description |
|---|---|
| Log File | The file to which the logger saves data. It is the same path as your configuration with a **log** extension. You cannot edit the log file path. |
| Logger Status | Indicates whether the data logger is active or not. |
| Available Symbols | Lists available symbols from the current active configuration. |
| Log Symbol List | Lists the symbols to be logged. |
| Add Symbol | To add a symbol to the log list, select the symbol from the available symbols list and click *Add Symbol*. |
| Remove Symbol | To remove a symbol from the log list, select it and click *Remove Symbol*. |
| Remove All Symbols | Click to remove all symbols from the *Log Symbol List*. |
| Start Logging | Click to begin logging symbol values to the log file. This means that for every value change in a symbol, the old and new values are logged. When *Start Logging* is selected, you can no longer add or remove symbols from the log list.<br><br>Logging continues even if the Data Logger is exited. When symbol values are being logged, the data logger icon appears in the tray next to the run-time icon. |
| Stop Logging | Stops logging. |
| Exit | Exits the Data Logger; however, if *Start Logging* is enabled, symbols continue to be logged. |

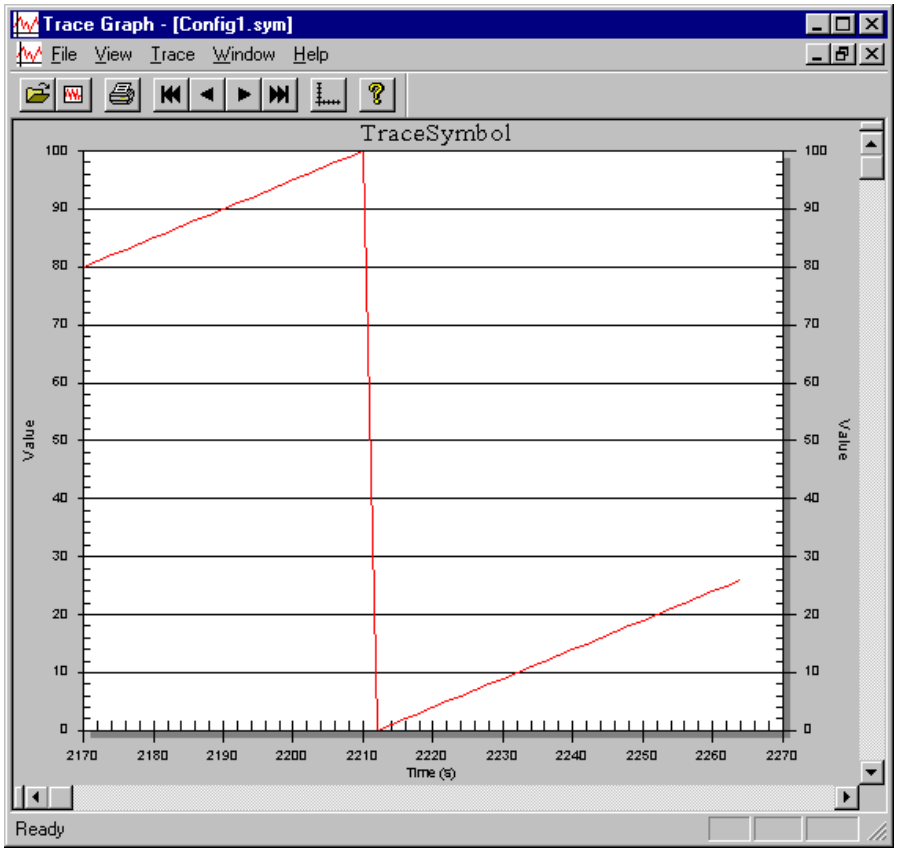
# Trace Symbol Data

Once you have selected symbols to log and started logging, you can select and trace (graph) these symbol data values.

### To trace symbol data

**Note:** Make sure you have activated your configuration, started the run-time system, selected symbols and started the Data Logger (started logging).

1. Select *Trace Graphing* from the *Tools* menu.
2. The Trace Graph window appears as shown in the figure (however a graph will not appear until you select a symbol to graph).



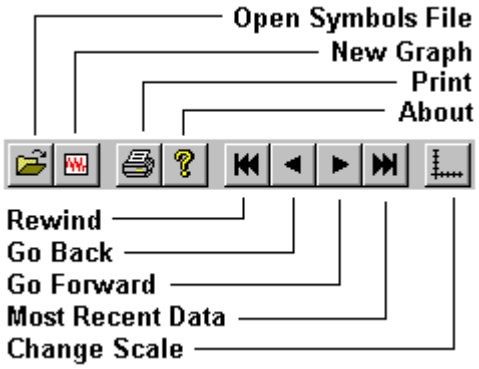
3. Select the file in which the symbols are located that are to be graphed. Do so by selecting *Open* from the *File* menu or use the tool bar  . A *Symbol File to Load* dialog box appears. Navigate to your symbol (**SYM**) file and open it.
4. A *Symbols* dialog appears, displaying the symbols you have selected in the *Data Logger* for the configuration file you opened. Select a symbol to graph.
5. A *Range* dialog box appears. Enter the X and Y limits and a time range to display. Click *OK* to start graphing the symbol, as shown in the figure.
6. You can graph additional symbols from the same configuration file by clicking *New Graph*  .

- You can configure the display of the graph by positioning the mouse pointer in the graph and using the context menu (right-click mouse button). For information on using the graph configuration features, refer to the help file accessed from the context menu.
- Refer to the tables for menu descriptions and to the tool bar figure for all Trace operations.

| File Menu | Description |
|---|---|
| Open | Opens a new Symbol File. |
| Print | Prints selected graph. |
| Print Setup | Standard Windows print setup dialog box. |
| Recent File List | Opens recent symbol files. |
| Exit | Closes Log Graph. |

| View Menu | Description |
|---|---|
| Tool Bar | Toggles display of tool bar on and off. |
| Status Bar | Toggles display of status bar on and off. |

| Trace Menu | Description |
|---|---|
| Change Scale | Displays the Range dialog box to let you change the X and Y limits and time range for the graph display. |
| Go to Beginning | Goes back to the first point graphed. |
| Go Back | Moves display along the X axis. |
| Go Forward | Moves display along the X axis. |
| Go to End | Goes to the end of the graph displaying the most recently graphed data. |




| Tool Bar | Description |
|---|---|
| Open Symbols File | Opens a new Symbol File. |
| New Graph | Graph additional symbols from the same symbol file. |
| Print | Print selected graph. |
| About | Displays about box. |
| Rewind | Goes back to the first point graphed. |
| Go Back | Moves display along the X axis. |
| Most Recent Data | Goes to the end of the graph displaying the most recently graphed data. |
| Go Forward | Moves display along the X axis. |
| Change Scale | Displays the Range dialog box to let you change the X and Y limits and time range for the graph display. |

# On-Line Editing

## On-Line Editing Operation

On-line editing refers to making editing changes to a running program. There are two on-line editing modes: seamless and non-seamless. The on-line editing mode entered depends on the type of editing changes made as discussed in **Rules**.

Seamless on-line editing allows editing changes to be made to a program and have those changes seamlessly reflected in the running program; that is, without disturbing run-time operation. Assume a program is running and the user makes a change to the program: if the change can be seamlessly online edited, the editor goes into seamless online edit mode. The online edit control appears with four buttons active:


**Restart Program** Restarts the program and resets the I/O. It aborts the current program and runs the new program, shutting down I/O in the process.

**Activate Changes** Seamlessly replaces the old version of the program with the new version; I/O scanning continues seamlessly. It parses the changes first if needed. The program is only parsed if the SFC or RLL file date is later than the program which has been parsed. The file is automatically saved if it has been changed.

During Activate Changes, the online edit box is displayed with all buttons disabled (while the changes are being parsed).

If the Activate Change is successful, the online edit box is removed. If parsing the changes is not successful, the

parse error message is displayed and the online edit buttons are enabled again.

**Parse Changes** Parses the new program changes without running them. If the Parse Changes is successful, the online edit box buttons are re-enabled. If a parse error occurs, the parse error message is displayed and the online edit box buttons are re-enabled.

During Parse Changes, the online edit box is displayed with all buttons disabled (while the changes are being parsed).

**Cancel Changes** Converts the source back to the running program and resumes highlighting the active program.

When a program is running and the user makes a change to the program which cannot be seamlessly implemented, the online edit box appears with only the *Restart Program* and *Cancel Changes* button active.

The user is prompted the first time a non-seamless edit is about to be implemented. After the user agrees to this, there is no further notification and the *Activate Changes* and *Parse Changes* buttons are disabled. If a seamless change was made earlier, the non-seamless change will force the program to be restarted.

# Rules

The following paragraphs list the rules for seamless on-line editing. If editing changes are made that cannot be seamlessly updated in the running program, then non-seamless on-line editing operation applies.

## General

Assume editing changes are made to an active program, the changes are saved without activating them (seamless changes or non-seamless changes), and the editor is closed. When the editor is started again, it will go directly to non-seamless online edit mode (the on-line edit box will appear with the *Activate Changes* and *Parse Changes* buttons disabled). This means that seamless online edits are not remembered across edit sessions but the changes themselves are identified by the file dates (SST file compared with SFC file, or RST file compared with RLL file).

## Symbols

You can seamlessly add new global memory variables. To do so, add new global memory variables from the Symbol Manager and click Apply - the new global memory variables seamlessly become active in the run-time engine. Any deletion or modification of global memory variables or

modification or addition of global user structures will require the configuration to be re-activated, aborting all programs and shutting down the I/O. When the user attempts to delete or modify a global memory variable or make changes to the global user structures, a notification appears that this edit will force all programs to be aborted.

- Global and local symbols can be added.
- Deleting or modifying symbols forces the program to restart.
- Modifying local user structures forces the program to restart.

## I/O

- Modification to the I/O drivers will require the configuration to be re-activated.

## RLL Programs

- When editing RLL programs, any change can be made. The full program is re-parsed and the program is seamlessly swapped. Symbols maintain their current value. I/O scanning continues seamlessly.

  **Note:** Positive and negative transition sensing contacts are prohibited from causing a transition the first time the element is evaluated. For on-line edit this means that after the online edit, the first scan will never sense the transition (since for the edited program, this is its initial scan). The place where this can cause an apparent problem is when the element is low and during the scan of the online edit, the element goes high. This transition would not be sensed.

## SFC Programs

- When editing SFC programs, the contents of steps, actions and transitions can be freely changed. The structure of the SFC (the SFC net) cannot be changed (this will force a program restart), including changing the name of an element (step, action or transition). During a seamless online edit, the full SFC (including any macro SFC) is re-parsed, the program segments representing the steps, actions and transitions are seamlessly swapped, the symbols maintain their current values, and steps, actions, and transitions which are active remain active. I/O scanning continues seamlessly.
- For SFC action qualifiers, only the duration can be changed. Any other change will force the program to be restarted.
- For SFC steps which contain Structured Text or RS-274, if the step is complete (the Structured Text or RS-274 has finished executing) it will remain complete with any active actions still scanning. If the step is not

complete (the Structured Text or RS-274 are still executing) the Structured Text or RS-274 will start over from the beginning.

- On-line editing of motion is not supported.

## File Operations

Online Edit and File operation features have some basic incompatibilities. The nature of Online Edit ensures that all variables including File Control blocks maintain their current state during the online edit. During Online Edit, the Structured Text inside of a step is restarted. If a set of Structured Text file commands are executing during an Online Edit, the commands will be restarted but the file will not be closed and reset to the start. The file operations will fail or execute incorrectly after the online edit.

**Example 1**

```
STEP1:
FILE_OPEN (fcb, "test.dat");
WHILE (NOT fcb.EOF) DO
        FILE_READ (fcb, struct1);
END_WHILE;
```

If an online edit is executed during the WHILE loop, STEP1 will be reset and the FILE_OPEN command will fail because the file is already open.

**Example 2**

```
STEP1:
FILE_OPEN (fcb, "test.dat");
STEP2:
WHILE (NOT fcb.EOF) DO
        FILE_READ (fcb, struct1);
END_WHILE;
```

If an online edit is executed during the WHILE loop, STEP2 will be reset and the FILE_READ will work correctly. There will probably be a timing problem in this case because the FILE_READ is not aborted and may still be active when the online edit is executed.

## Structured Text Programs

- Structured Text does not support seamless on-line editing. If changes to the Structured Text file are made while the program is running and the file is saved, the non-seamless on-line editing buttons are enabled.

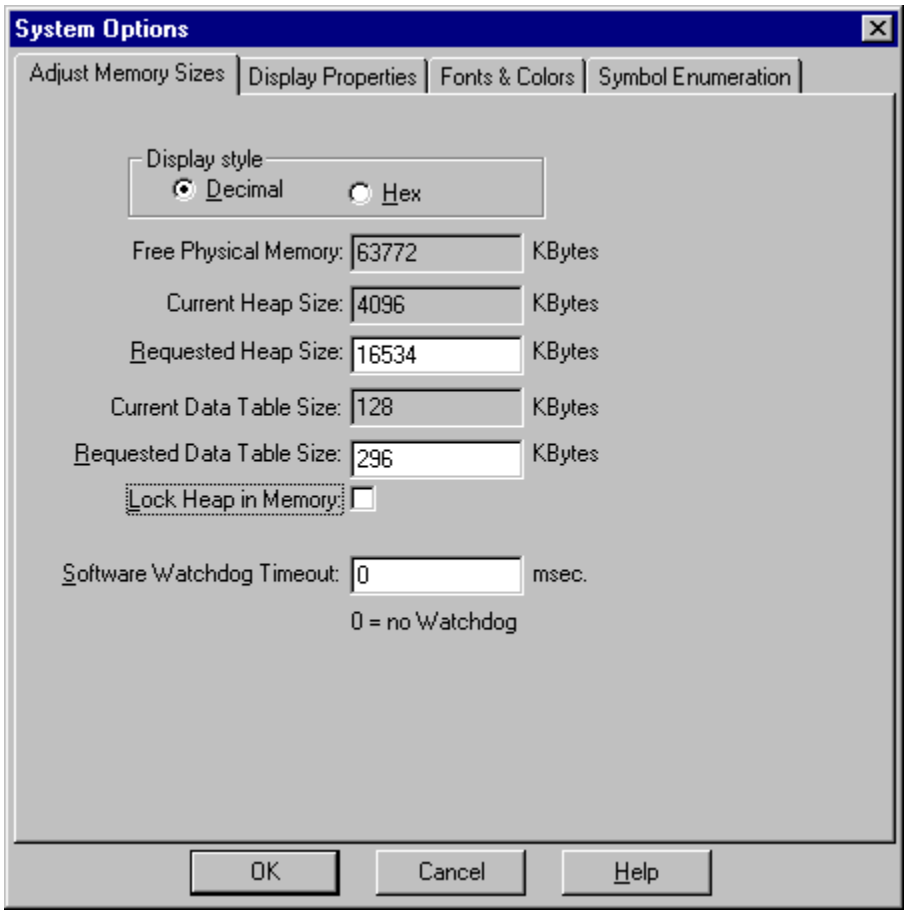
## Instruction List Programs

- If changes to the Instruction List file are made while the program is running and the file is saved, the on-line editing buttons are enabled.

# System Options

## System Options Dialog Box

System options let you set preferences that affect the behavior and appearance of the control system.



## Adjust Memory Sizes Tab

| Field | Description |
|---|---|
| Display Style | Displays memory size options in Decimal or Hex. |
| Free Physical Memory | The current free physical memory available on the computer. This cannot be edited, but depends on the type and number of open Windows applications. |
| Current Heap Size | This shows the current Heap setting and cannot be edited. The Heap is memory allocated (on start-up) for the control system. The Heap is used up by open application programs and symbols. As it is used up, this setting may need to be increased. However, the larger the Heap size, the less memory will be available for other Windows applications |
| Requested Heap Size | If the Heap needs to be increased (or can be decreased), provide a new Heap size here. The Heap can be larger than physical memory, as it will use virtual memory from the swap file. If you run out of control system memory during an editing session, you will get a message suggesting to increase the Heap Size. |
| Current Data Table Size | This is memory used for symbols and depends on the symbol data type. For example, large arrays use more data table memory. If you run out of data table memory during an editing session, you will get a message suggesting to increase the data table size. |
| Requested Data Table Size | If necessary, provide a new data table size. |
| Lock Heap in Memory | If this field is set, it will improve the determinism of the control system. When the control system is started, it will attempt to lock the Heap into physical RAM; otherwise, it can use virtual memory. You are not given an indication of whether locking the Heap in memory was successful or not; therefore, it is best to start the control system run-time system before any other application to allow as much physical memory for the Heap as possible. |
| Software Watchdog Timeout | This option can be used to prevent your application from getting stuck in an endless loop. You can set the timeout to be from 2-times the scan rate up to 10 seconds. The default value of 0 disables the software watchdog timeout.<br><br>In operation, the execution time of each instruction is checked, and if it exceeds the timeout value, a fault is generated. If a timeout fault does occur, and the problem is in an SFC step, you are notified of the step name causing the fault.<br><br>**Note:** This option is for development and debugging purposes only. Do not leave this enabled in normal control operation as it can impact performance. |

## Properties Tab

| Field | Description |
|---|---|
| Runtime Tray Icon | If set, displays the Runtime icon in the tray; otherwise, it is also displayed in the task bar. If it is displayed in the task bar, you can access it using keyboard operations; however, if it is displayed in the tray, it is necessary to use the mouse to access it. |
| Number of Decimal Places | This controls the number of decimal places displayed in the function block details in an RLL program. The more decimal places displayed, the more space is used. |
| Number of FB Symbol Characters | This controls the length of space for displaying characters in the function block details in an RLL program (not necessarily the actual number of characters). A standard character size of an upper case *X* is assumed. If characters take up less space (e.g., lower case *l*), then more characters will appear. |
| Editor Display Update Time | This controls how often the Program Editor is updated; for example, the values in the Watch Window. |
| Show IEC Style Locations | If set, displays I/O points in IEC 1131-3 syntax rather than by symbol name. |
| Display non-fatal error messages | Controls the display of non-fatal error messages. When checked, non-fatal error messages are logged to EVENTS.EXE (Event logger) but it is NOT displayed. When unchecked, the message is both displayed and logged. Non-fatal error messages are messages that are generated by the Runtime Subsystem or by any installed driver. Typically, Non-fatal error messages indicate that a non-critical recoverable error has occurred. |
| Display fatal error messages | Controls the display of fatal error messages. When checked, fatal error messages are logged to EVENTS.EXE Event logger) but it is NOT displayed. When unchecked, the message is both displayed and logged. Fatal error messages are messages that are generated by the Runtime Subsystem or by any installed driver. Typically, Fatal error messages indicate that a severe, unrecoverable error has occurred. The error usually results in the specified subsystem's software being halted. |

**Note:** The reason for disabling error messages is: Some drivers can produce large volumes of non-fatal error messages (PMAC, Modbus+, etc.), which can become annoying when they pop up over your MMI. This allows you to decide if you wants to be informed of fatal and/or non-fatal error messages. If you choose to disable error messages, it is your responsibility to monitor the system for errors.

## Fonts & Colors Tab

| Field | Description |
|---|---|
| Text Editor Options | Refer to Customize Text Editor. |
| RLL & SFC Editor Fonts | Sets the fonts in the program editors. |

**Fonts & Colors Tab**

| Field | Description |
|---|---|
| Colors - RLL & SFC | Sets various editor colors. |
| Reset to Default Colors | Restores all colors to their defaults. |

**Symbol Enumeration Tab**

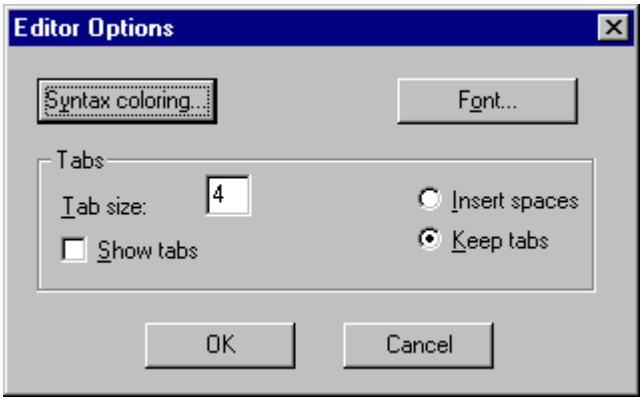
| Field | Description |
|---|---|
| | Enumerations refer to the elements of a complex symbol. That is, a symbol that has elements that can be individually accessed: structures, function blocks, etc. The following check boxes work with the *Show Symbol Enumerations* command on the editor *View* menu. If the associated check box is set and *Show Symbol Enumerations* is enabled, then the respective enumerations are visible in symbol lists. If the check box is reset or *Show Symbol Enumeration* is disabled, the enumerations are not visible in symbol lists.<br>Showing symbol enumerations (depending on the number of symbols) can affect performance as you are using the editors, since more symbols will need to be loaded into the symbol lists.<br>**Note:** Even with enumerations off, if you drag a symbol from the Symbol Manager and drop it, a list box appears with that symbol's enumerations from which you can select the enumeration you want to use. |
| Show Function Block Enumerations | Shows the function block instance inputs and outputs. |
| Show User Type Enumerations | Shows elements within an instance of a user type. |
| Show Axis Enumerations | Shows axis element symbols. |
| Show Axis Group Enumerations | Shows axis group element symbols. |
| Show System Object Enumerations | Shows the variables associated with instances of timers (TMR), PIDs, and PRGCBs. |

# Customize Text Editor

Editor customization allows you to set tab, color, and font options for the editor.

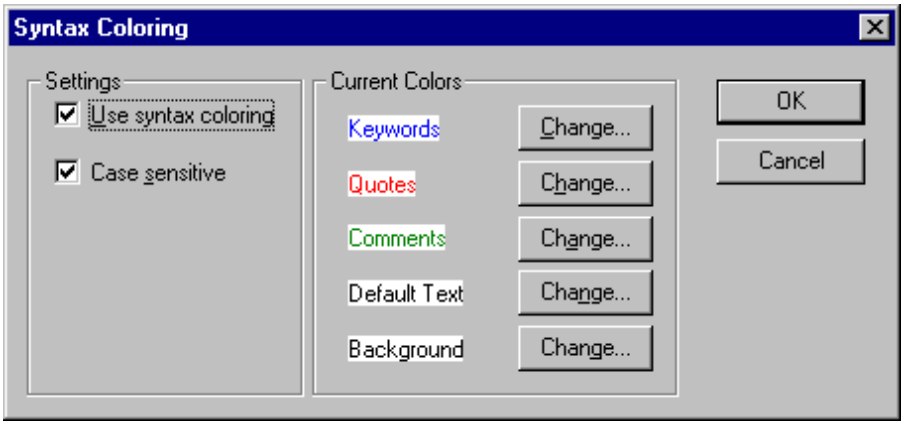
**To customize the editor**

- Choose *System Options* from the *Tools* menu, then choose *Text Editor Options* from the *Fonts & Colors* tab. The *Editor Options* dialog box appears. Refer to the following tables for descriptions of the options.




### *Editor Options*

| Item | Description |
|---|---|
| Syntax coloring | Opens the Syntax Coloring dialog box from which you can set background and various text color preferences. |
| Font | Displays a standard Windows font selection dialog box from which you can set the editor's font characteristics. |
| Tabs | Tab size - the number of spaces for which a tab is equivalent.<br>Show tabs - if checked, presence of tabs are indicated by a tab character (»).<br>Insert spaces - inserts the tab equivalent number of spaces when the tab key is pressed, rather than a tab.<br>Keep tabs - tabs are inserted as tabs, not spaces. |

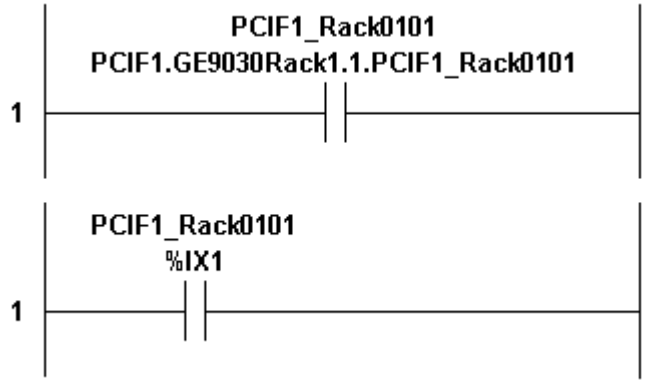



### *Syntax Coloring*

| Item | Description |
|---|---|
| Use syntax coloring | Check this item to use syntax coloring; if unchecked, all text is displayed in the Default Text color. |
| Case sensitive | If checked, keywords are considered case-sensitive within the editor. |
| Current Colors | Clicking the Change button for the respective text type displays a standard Windows color selection dialog box that allows you to set a color for the text or background. |

## IEC Style Locations

The following figure shows an example of normal and IEC 1131-3 style locations for I/O points. (*Display Symbol Locations* must be enabled.) If a symbol is memory mapped variable, then **Memory** appears for the symbol location.




The IEC 1131-3 symbol location syntax is as follows:

| | |
|---|---|
| % | Directly represented variable |
| I | Input point |
| Q | Output point |
| X | BOOL |
| B | BYTE |
| W | WORD |
| D | DWORD |

# UPS Configuration

## Configuring the UPS

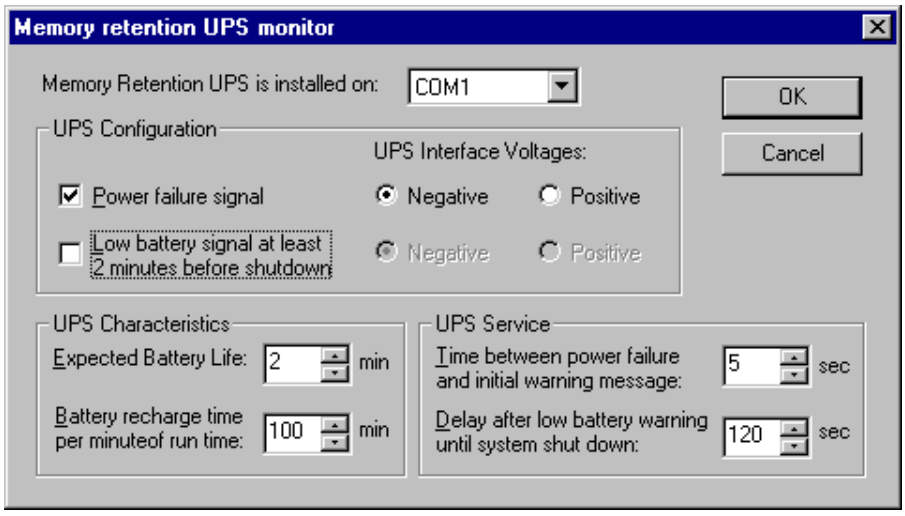
The control system can work in conjunction with an uninterruptible power supply (UPS). The control system can receive signals from the UPS in order to prepare an orderly shutdown. The signals appear in the Symbol Manager as:

| | |
|---|---|
| RT_POWER_FAIL | Power failure detected. |
| RT_LOW_BATTERY | Low battery detected. |

### To use the UPS

- Locate the *ASAP Applications* menu from the Windows *Start* menu and choose *UPS Monitor*.

The UPS Monitor icon  appears in the tray. Configure the UPS by double-clicking on the icon. The *Memory retention UPS monitor* dialog box appears.



| Item | Description |
|---|---|
| Memory Retention UPS is installed on: | Select the serial port to which the UPS is connected. |
| UPS Configuration | Refer to your UPS documentation to fill in these fields. |
| Power failure signal UPS Interface Voltage | Check this option if the UPS provides a power failure signal. If checked, select whether it is a negative or positive voltage. |
| Low battery signal UPS Interface Voltage | Check this option if the UPS provides a low battery signal. If checked, select whether it is a negative or positive voltage. |
| UPS Characteristics | If the UPS does not provide a low battery signal, you can enter UPS characteristics by referring to the UPS documentation. |
| UPS Service | You can set these preferences as needed for your system. |

### To shutdown the UPS

- Right-click on the UPS Monitor icon and choose *Shutdown Memory UPS*.

# Dynamic Data Exchange (DDE)

## About the DDE Interface

The control system software allows a DDE interface option to third-party software to communicate with the control system software through a DDE interface. When the DDE interface is enabled, the control system software accepts external program commands and read/write requests from/to the control system I/O and global memory variables.

With NetDDE, these external program commands and read/write requests can be made from a remote network node. When using NetDDE, the DDE server name must be prefixed by the server network node address.

## DDE Communication with Microsoft's Excel

You can use Microsoft Excel's DDE features to transfer data to and from global symbols. Using Excel you can transfer data from the control system for summary and analysis. You can also transfer data from Excel to the control system for controlling control system program execution or to provide features such as a low level recipe management system.

### Transferring Data to Excel

To transfer the value of a global symbol to a cell in a Excel Spreadsheet, enter a formula like the following into the cell that is to receive the value:

```
=ProgMgr|'_main _main'!VariableName
```

or

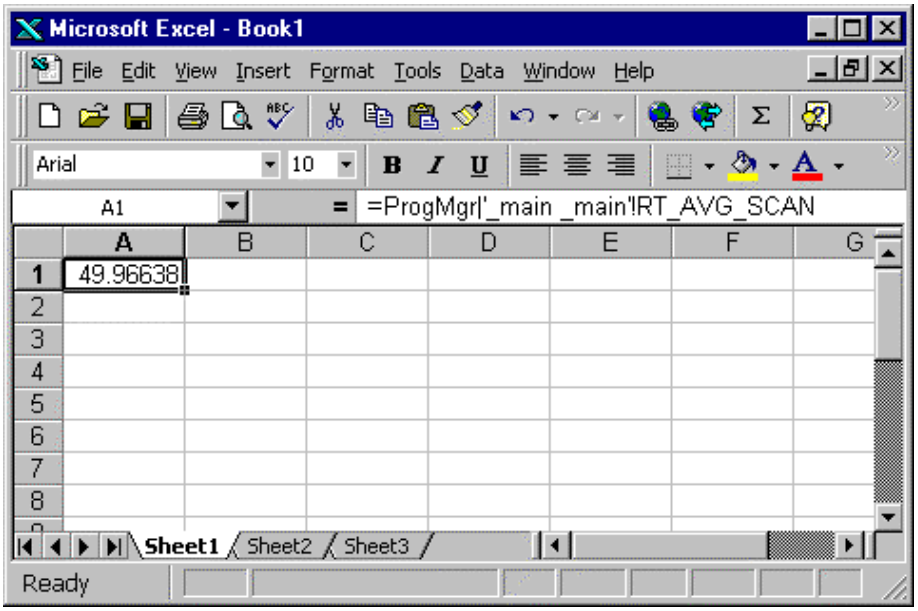
```
=ProgMgr|'_main _main'!'ArrayName[4]'
```

The string '_main _main' is the DDE topic and will allow you to fetch global variables. Make sure that you type one and only one space between the two "_main" as shown in the formula above. Replace the VariableName with the name of the global variable that you wish to transfer. The VariableName must use the same case and spelling as used in the control system application. Array elements can be accessed by using the second format to address the appropriate element in any array, but because of the square brackets ([ ] ), the element name must be enclosed within single quotes.

This formula will establish a Hot DDE link to the control system that will update the value in the Excel spreadsheet whenever the variable in the control system is changed.

### *Excel DDE Example*

The value of the global variable RT_AVG_SCAN is monitored in cell A1.




The formula in A1 is: =ProgMgr|'_main _main'!RT_AVG_SCAN

**Where:**

| | |
|---|---|
| = | is the Excel operator Equal To |
| ProgMgr | is the application name (capitalization is important) |
| \| | is the pipe symbol used to separate the application name from the topic name |
| '_main _main' | is the topic name (Make sure that you type one and only one space between the two "_main" as shown.) |
| ! | is an exclamation mark and is the delimiter between the topic name and the variable name |
| RT_AVG_SCAN | is the name of the symbol being monitored. You can list any global symbol name here. |

This formula is dynamic and updates the value in A1 continually. For more information about entering formulas in an Excel spreadsheet, refer to the Excel user documentation.

## Transferring Data to the Control System

Transferring a value from Excel to a control system global variable requires a DDE transaction routine coded in an Excel macro similar the following:

```
Sub transfer()
        Dim x
        x = Application.DDEInitiate("ProgMgr", "_main _main")
        Set rangeToPoke = Sheets("Sheet1").Cells(7, 11)
        Application.DDEPoke x, "VariableName", rangeToPoke
        Application.DDETerminate x
 End Sub
```

This routine, when executed, will transfer the value of the cell in row 7, column 11 (K7) to the global variable called VariableName. Make sure that you type one space between _main and _main as mentioned above. Replace Sheet1 with the name of your worksheet that contains the value to be transferred.

## Transferring Values to the Control System upon Request

Transferring values from Excel to the control system upon request from the control system requires a DDE transaction routine coded in an Excel macro similar the following:

```
Dim TimeSet As Double
Sub RunMeFirst()
   TimeSet = Now + TimeValue("00:00:05")
   Application.OnTime TimeSet, "Transfer"
End Sub
Sub Transfer()
   Dim x, y
   Dim z As Variant
   x = DDEInitiate("ProgMgr", "_main _main")
   z = DDERequest(x, "Trans1")
   y = Val(z(1))
   If y = 1 Then
      Set rangeToPoke = Sheets("Sheet1").Cells(2, 3)
      DDEPoke x, "Data1", rangeToPoke
      Set rangeToPoke = Sheets("Sheet1").Cells(3, 3)
      DDEPoke x, "Data2", rangeToPoke
```

```
        Set rangeToPoke = Sheets("Sheet1").Cells(4, 3)
        DDEPoke x, "Data3", rangeToPoke
        Set rangeToPoke = Sheets("Sheet1").Cells(1, 3)
        DDEPoke x, "Trans1", rangeToPoke
      End If
      DDETerminate x
      RunMeFirst
    End Sub
```

When executed, this macro will check a global variable called Trans1 every 5 seconds. If Trans1 is set to 1 it will then transfer the value of the cell in row 2, column 3 (C2) to the global variable called Data1, likewise C3 to Data2, and C4 to Data3. It will then transfer C1, which was preset to 0, to Trans1 resetting the transfer request. Make sure that you type one space between _main and _main as mentioned above. Replace Sheet1 with the name of your worksheet that contains the values to be transferred. The time interval may be changed from 5 seconds by changing the TimeValue in the RunMeFirst function.

## Network DDE Communication

A Network DDE Link consists of two parts; the DDE server application and the DDE client application. The DDE server application contains the data to be shared. The DDE client requests the data from the DDE server, thus creating a DDE Link. Network DDE requires the server application to make the data accessible on the network. The server Application creates a Network DDE share, available over the network, which the client application can connect to.

**To create a Network DDE link under Windows NT 4.0**

Use the DDE Share Manager (%SYSTEMROOT% \SYSTEM32 \ DDESHARE. EXE) to create a share on the DDE server:

1. Select *Add a Share* in the SHARES/DDE *Shares* menu.
2. Enter a share name. The share must end with a dollar sign ($), for example DDEShare$.

   **Note:** This share is not a directory. You cannot see it in Explorer.

3. Under *Application Name*, enter the following:

   Old Style: ProgMgr
   New Style: ProgMgr
   Static: ProgMgr

4. Under *Topic Name*, enter the following:

   Old Style: _main _main

New Style: _main _main

Static: _main _main

**Note:** Make sure there is a blank space between _main and _main.

5. Select *Allow Start Application*.
6. Choose *Permissions* and then set permissions for the share the same way you would set permissions on files using Explorer. Click *OK*.
7. Select the share you just created and choose *Trust Share*. Choose *Set*.
8. Enable the two set boxes (*Start Application Enable* and *Initiate to Application Enable*). Choose *OK*.
9. Launch Excel on the client.
10. Create a link to the share that you just created.

    This example assumes a machine name of UNIVERSE.

    Go to any cell in Excel and enter:

    =' \ \UNIVERSE\NDDE$' |' DDEShare$'!'Counter1'

    where:

    Counter1 is a valid ***global*** variable

    **Note:** You may be prompted for a password. If so, enter a valid user account and password which exists in the DDE server's domain or a trusted domain.

    The linked data should appear in the cell when you press enter. If the cell shows N/A, choose *Links* from the *Edit* menu. The link should be listed as follows:

    Source File: DDEShare$

    Item: Counter1

    Type: \ \UNIVERSE\NDDE$

    Status: Automatic

    To update the link, select *Update Now*. The link should activate and the date should be linked to the client spread sheet.

    **Note:** Under Windows 95 - Requires NetDDE.exe to be running. Run NetDDE.exe.

# OLE for Process Control (OPC)

## OPC Server Overview

An OPC server is included in the control system product. OPC stands for OLE for Process Control and provides a standard means of transferring symbol data. Support is for OPC 2.0 and earlier.

## Using OPC

The following provides a general procedure for using the control system's OPC server. Each client will have its own interface, refer to your client application product for specific information.

**To use OPC**

1. Start the run-time system.
2. Open your HMI application that supports OPC.
3. There should be a tag browser or similar feature. Use it to view the OPC servers running on the computer.
4. Choose the server: **ASAPInc.OPC200**
5. You should then have access to a list of tags.
6. Select one or more tags of interest. Make sure the data type is correct, if not and there is a feature to change it, do so.

   **Note:** You may also need to make a group. These are tags that are updated at the same rate.
7. Use the tag by assigning it a symbol name or putting it into a control, as determined by your HMI application.

# Import/Export Configuration

## Import/Export Introduction

The import/export features provide the following capabilities:

- For drivers that do not allow cut and paste capabilities, you can export the configuration; use a text editor to cut and paste the configuration; and then import the configuration back into the control system.
- Offline editing of the configuration. Instead of using the control system to edit configurations, you can use a text editor or CSV compatible utility (such as Microsoft Excel) to edit the configuration. Or, you can write your own dialog based configuration software (using VB, VC++, etc.) to create CSV files. You can then import the CSV into the control system.
- You can create your own application to automate the generation of customer specific configurations.

## Import/Export Support

Currently, the following configuration information types are supported for CSV import and export:

- Global symbols.
- Yaskawa I/O driver.
- GE 90/30 driver.
- PCIM driver.

## Format

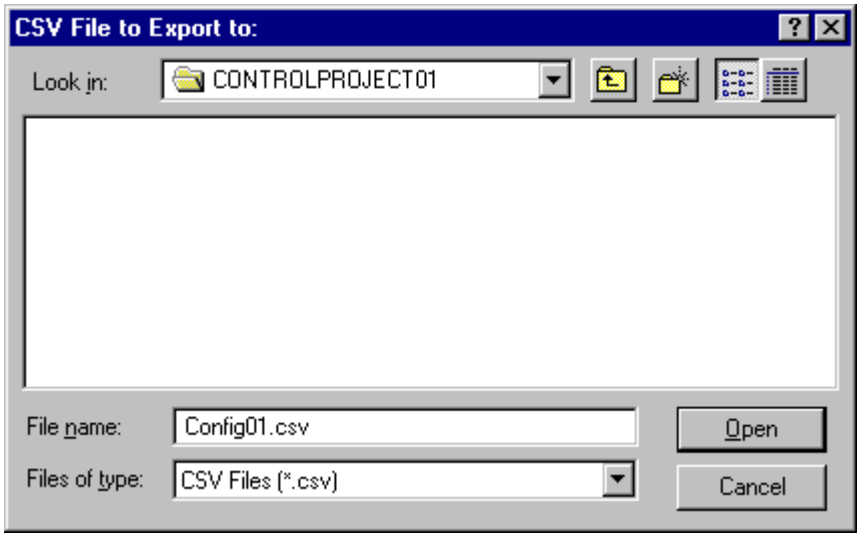
The configuration is exported as a comma separated file that can be read by a text editor or spreadsheet. The format specification is available upon request.

## Exporting a CSV File

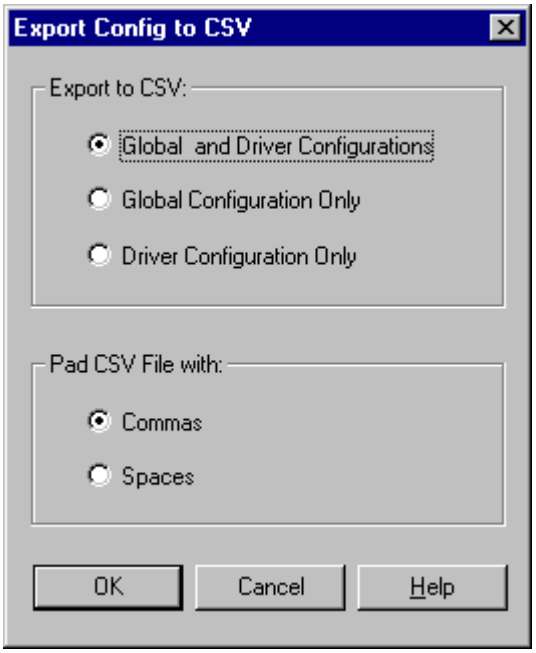
### To export a configuration to a CSV file

1. Open the configuration to be exported.

2. Select *Export Config to CSV* from the *File* menu. A file selection dialog box appears.




3. Select or type a file name for the export and click *Open* to continue. The *Export Config to CSV* dialog box appears.

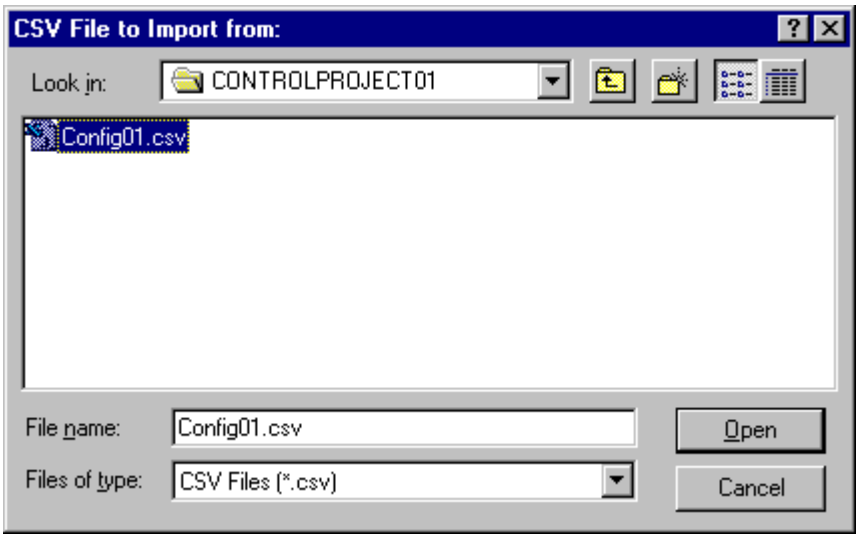


4. Select export options as needed and click *OK* to proceed. The configuration is saved to the designated file.

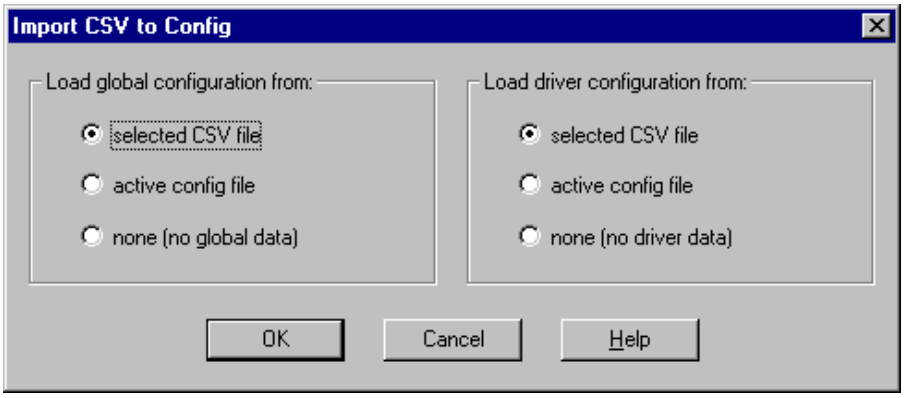
## Importing a CSV File

### To import a CSV file to a configuration

1. Select *Import CSV to Config* from the *File* menu. You are prompted to close the currently open configuration.
2. A file selection dialog box appears.




3. Select or type the file name of the configuration file to be imported and click *Open* to continue. The *Import CSV to Config* dialog box appears.




4. Select import options as needed. You can choose to import either global symbols and/or driver information, use the global symbols or driver information from the active configuration, or start the configuration without global symbols or driver information.
5. Click *OK* to proceed. The configuration data is imported and becomes the active configuration.

# Index

## D



## E



## F



## G



## H



## I

## P



## R

## T



## U



## V



## W



## Z

139837(B)

Xycom Automation, LLC
750 North Maple Road
Saline, MI 48176
***http://www.profaceamerica.com***

Phone: 734-429-4971
Fax: 734-429-1010