GFK-1056

GE Fanuc Manual Series 90-30

State Logic Control System

GE Fanuc Automation

Programmable Control Products

Series 90-30 State Logic® Control System

User's Manual

GFK-1056C *March, 1998*

Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Fanuc Automation assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Fanuc Automation makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

The following are trademarks of GE Fanuc Automation North America, Inc.

Alarm Master	Field Control	Modelmaster	Series One
CIMPLICITY	GEnet	ProLoop	Series Six
CIMPLICITY	Genius	PROMACRO	Series Three
PowerTRAC	Genius PowerTRAC	Series Five	VuMaster
CIMPLICITY 90–ADS	Helpmate	Series 90	Workmaster
CIMSTAR	Logicmaster		

Content of This Manual

Table of Contents

Chapter 1. Preliminaries

Chapter 2. Getting Started

Chapter 3. Option Modules

Chapter 4. State Logic® Control Theory

Chapter 5. Creating a State Logic Control Program

Chapter 6. ECLiPS Programming Features

Chapter 7. Program Instructions

Chapter 8. Perform Functions

Chapter 9. PID Loops

Chapter 10. On - Line Features (Debug Mode and OnTOP)

Chapter 11. Serial Communications

Appendix A Function Keys

Appendix B Language Structural Summary

Appendix C Predefined Keywords

Appendix D Errors

Appendix E ECLiPS Specifications

Index

Related Publications

GFK-0582 ***Series 90 PLC Serial Communications User's Manual***

GFK-0356 ***Series 90-30 Programmable Controller Installation Manual***

GFK-0293 ***Series 90-30 High Speed Counter User's Manual***

GFK-0695 ***Series 90-30 Enhanced Genius Communications Module***

GFK-1084 ***TCP/IP Ethernet Communications for the Series 90-30 PLC User's Manual***

® State Logic is a registered trademark of Adatek, Inc.

GFK-90486 ***Genius I/O System and Communications User's Manual***

GFK-1034 ***Series 90-30 Genius Bus Controller User's Manual***

GFK-0582 ***Series 90 PLC SNP Communications User's Manual***

GFK-0840 ***Series 90-30 Axis Positioning Module APM Standard Mode User's Manual***

At GE Fanuc Automation, we strive to produce quality technical documentation. After you have used this manual, please take a few moments to complete and return the Reader's Comment Card located on the next page.

Contents

Preliminaries 1-1

Items Included with ECLiPS 1-2

Product Registration 1-2

Getting Answers to Your State Logic Questions 1-2

How to Use this Manual 1-2

Manual Organizational Categories 1-3

Setup (Chapters 1 - 3) 1-3

Fundamentals (Chapters 4 & 5) 1-3

Programming (Chapters 6 -- 9) 1-3

On - Line (Chapter 10) 1-3

Serial Communications (Chapter 11) 1-4

Brief Description of the Manual Chapters 1-4

1. Preliminaries 1-4

2. Getting Started 1-4

3. Option Modules 1-4

4. State Logic Control Theory 1-4

5. Creating a State Logic Control Program 1-4

6. ECLiPS Programming Features 1-4

7. Program Instructions 1-5

8. Perform Functions 1-5

9. PID Loops 1-5

10. On - Line Features (Debug Mode and OnTOP) 1-5

11. Serial Communications 1-5

Appendix A - Function Keys 1-5

Appendix B - Language Structural Summary 1-5

Appendix C - Predefined Keywords 1-5

Appendix D - Errors 1-5

Appendix E -. ECLiPS Specifications 1-6

Notation Conventions 1-6

Getting Started 2-1

Introduction 2-2

The State Logic PLC System 2-2

State Logic CPU 2-2

Baseplates 2-4

Power Supply 2-4

I/O Modules 2-4

Serial I/O Devices 2-4

Programming and Online Interface Software 2-5

ECLiPS 2-5

Program Mode 2-5
Debug Mode 2-5
OnTOP 2-6
Setting Up the 90-30 State Logic Software 2-6
Setting Up ECLiPS 2-6
Hardware Requirements 2-6
ECLiPS Installation 2-6
Hardware Key 2-7
Configuring PC Memory for ECLiPS 2-8
Using Expanded and/or Extended Memory 2-8
Setup Option 2-8
Using Conventional Memory 2-9
ECLiPS Security 2-9
Setting Up OnTOP 2-9
Hardware Requirements 2-9
Installation 2-10
Using the ECLiPS Program Files 2-10
OnTOP Security System 2-10
Connecting ECLiPS and OnTOP to the Controller 2-11
Configuring the State Logic Control System 2-12
State Logic CPU Configuration Options 2-12
CPU Model Number 2-12
Automatic Program Execution on Power Up 2-13
Runtime Fault Response 2-13
ECLiPS Port Name 2-13
Configuring System Racks 2-13
Configuring PLC Modules 2-14
Configuring Digital and Analog I/O Modules 2-14
Starting Reference Number 2-14
I/O Names 2-14
High Density Analog Modules 2-15
16 Channel Analog Input Module 2-15
8 Channel Analog Output Module 2-16

Option Modules 3-1

High Speed Counter Module 3-2
Accessing HSC Data 3-2
Configuring The HSC Module 3-2
Starting Reference Number 3-3
Extended Configuration 3-3
I/O Names 3-4
Programmed Configuration Changes 3-5

Enhanced Genius Communication Module (GCM+) 3-5
Configuring the GCM+ 3-5
Communications Coprocessor Module (CMM) 3-7
Sending and Receiving Data 3-7
Configuration 3-8
Sending Data as a Master 3-10
Ethernet Interface Module 3-11
Retrieving Data from the State Logic Control System 3-12
Configuration 3-12
Sending Data Using the Comm_Request Function 3-13
Genius Bus Controller (GBC) 3-14
Global Data 3-14
Datagrams 3-14
Configuration 3-16
Axis Positioning Module (APM) 3-19
Configuration 3-19
Programmable Coprocessor Modules (PCM) 3-21
Accessing CPU Data 3-21
Operational Notes 3-22
PCM Comm_Request 3-22
Configuration 3-23
Serial Communications Module (SCM) 3-24
Module Description 3-24
OK LED 3-25
Reset Pushbutton 3-25
Serial Connector 3-25
WYE Serial Cable 3-26
Cabling 3-28
Configuring the SCM 3-29
SCM Configuration Mechanics 3-32
Recommended Schemes 3-32
Using the *Set_commport* Command 3-32
SCM Configuration Mechanics 3-33
Foreign Modules 3-33

State Logic Control Theory 4-1

State Logic Control Theory 4-2
The Concept of Finite States 4-2
What Makes State Control Logic Different 4-4
A Collection of Tasks is a State Logic Program 4-4

Developing State Logic Programs with ECLiPS 4-4
Tasks - Sequences of States 4-5
States - The Building Blocks of a Task 4-6
Statements - The Command Set for State Descriptions 4-7
Communications 4-8
Scan Overview 4-9
Interaction Between Tasks 4-9
Creating Process Diagnostics 4-11
Creating Diagnostic Routines 4-11

Creating a State Logic Control Program 5-1

Outline the Application 5-2
Identify the Tasks 5-2
Identify The States 5-2
Identify the Statements 5-3
Writing The Program 5-4
Using English Names in the ECLiPS Program 5-4
Naming Tasks 5-4
Naming States 5-5
Naming I/O Circuits, Variables, and Internal Flags 5-5
Statement Structures 5-6
Constructing Statements 5-6
Using Keywords, Synonyms and Filler Words 5-7
Using Variables 5-7
Integer Variables 5-8
Internal Flag 5-8
Floating Point Variable 5-8
String Variable 5-8
Character Variable 5-9
Reserved System Variables 5-9
Program Execution 5-9
Hints for Creating ECLiPS Programs 5-12
Outputs are OFF by Default 5-12
Task Design 5-14
Write Term Considerations 5-15
Calculations and a Scanning Operating System 5-15
Read Term Considerations 5-16
Timer Considerations 5-16
Documentation Hints 5-18
Descriptive Names 5-18

Underscores 5-18
Programming Conventions 5-19
Comments 5-19
Scan Time 5-19

ECLiPS Programming Tools 6-1

State Logic Program Editor 6-2
ECLiPS Editor Screen 6-2
Creating Program Text - Overview 6-2
Add Menu 6-2
List Menu 6-3
Text Menu- Block Operations 6-3
Find Menu - Search and Replace 6-3
Moving Through the Program 6-4
Project Menu 6-4
Project Management 6-5
"Retrieve" 6-5
"Save" 6-5
"New Path/Drive" 6-5
"Make a New Project" 6-5
"Copy" 6-5
Backup 6-6
Backup Project 6-6
Restore Project 6-6
Automatic Backup 6-6
ECLiPS/Controller Version Mismatch 6-6
"Delete a Project from the Disk" 6-6
"Import" 6-6
Documentation - "Print Function" 6-7
"Error Check, Translate, and Download Projects" 6-9
"Task Groups" 6-9
Project Files 6-9
Naming Variables and I/O Points 6-10
Name Data 6-10
Reference Number 6-10
Save Over Halt 6-10
Naming Analog Channels 6-10
Raw Analog Data 6-10
Scaled Analog Data 6-11
Define Menu 6-11
List Menu 6-12

On-Line Program Changes 6-12
Runtime Editing Restrictions 6-13
How to Use the ECLiPS Menus 6-13
Using ECLiPS Hot Keys 6-13
ECLiPS Editor Functions 6-14
How to Use ECLiPS Lists 6-14

Program Instructions 7-1

Program Structure 7-2
Task Groups 7-2
Tasks 7-3
State 7-3
Statements 7-3
Expressions 7-3
Words 7-4
Numeric Data Types 7-4
Variables 7-5
Numeric Variables 7-6
Integer Variables 7-6
Floating Point Variables 7-6
Analog Variables 7-6
Reserved System Variables 7-7
Clock/Calendar Variables 7-7
Fault Variables 7-8
Digital Variables 7-8
%I Digital Inputs 7-8
%Q Digital Outputs 7-9
%G Digital Globals 7-9
Internal Flags 7-9
ASCII Variables 7-10
Functional Instructions 7-10
Turning ON Digital Points 7-11
Assigning Analog Output and Variable Values 7-11
Make Instruction Definition: 7-11
Math-Assignment Instruction 7-11
Set_Bit/Clear_Bit Instruction 7-12
State Transitions 7-12
Changing the Current State within a Task 7-12
Changing the State of Another Task 7-14
Serial Output (Write Instruction) 7-15

PID Loop Control (Start_PID, Stop_PID) 7-17
Change Serial Port Configuration 7-17
Perform Functions 7-17
Conditional Instructions 7-18
Digital Conditional 7-18
Testing Digital Types 7-18
Testing Integer Variable Bits 7-18
Timer Conditional 7-19
Relational Conditional 7-19
State Conditional 7-20
Complex Conditionals 7-20
Serial Input Conditional 7-20
Mathematical Calculations 7-21
Operator Precedence 7-22
System Clock/Calendar 7-22
Differences Between the 311/313/323 and 331/340 Clock/Calendar 7-22
Grammatical Rules 7-23
Filler Words 7-23

Perform Functions 8-1

General Information 8-2
Table functions 8-3
Define_Table 8-3
Entering and Retrieving Table Values 8-4
Initializing Tables 8-5
Copy_Table_To_Table 8-6
Table Uses 8-6
Data Type Conversion 8-7
Copy_Var_To_Integer 8-7
Step by Step Instructions 8-8
Convert Double Integer and Floating Point Data Types 8-10
String Manipulation 8-10
General 8-10
Operations 8-11
E for Extract 8-11
s For store in sub-string 8-11
I for extract and convert to Integer 8-11
i For Convert int eger To ASCII And Store In String 8-12
F for extract and convert to Float 8-12

f For Convert float To ASCII And Store In String 8-12
C for Concatenate 8-12
L for string Length 8-12
M for Match the Given Character with a Character in the String. 8-12
Errors in General 8-12
Communications Request 8-13
Time Counter 8-13
High Speed Counter Data Transfer function 8-15
Shift Register 8-18
BCD I/O Representation 8-19
BCD_In_Convert 8-19
Output_BCD_Convert 8-20
Specialized perform functions 8-21
Display Date and Time 8-21
Get User Input 8-22
User Menu 8-22

PID Loops 9-1

PID Algorithm and Philosophy 9-2
Complex PID Control 9-3
Bumpless Transfer 9-3
Anti-Reset Windup 9-4
Initializing and Starting PID Loops 9-5
Starting PID Loop Execution 9-5
PID Initialization Form 9-5
Loop Tuning and Monitoring 9-7
Loop Tuning Form 9-9
Program Control of PID Loops 9-10
Using the PID Parameters in a State Logic Program 9-10
Using the Command and Status Bits in a State Logic Program 9-11
Accessing PID Parameters from a Remote Device 9-11

Online - Debug Mode and OnTOP 10-1

Online Display Screen 10-2
Project Menu Options 10-2
Run/Halt 10-2
Logging Data 10-3
CCM or SNP Setup 10-3
CCM Set Up Procedure 10-3

SNP Set Up Procedure 10-4
Process Simulation 10-4
Download a Project 10-4
Reset and Clear State Logic CPU 10-4
EEPROM Support 10-5
PLC Clock Functions 10-6
Monitoring Controller Values 10-6
Monitor Tables 10-6
View 10-7
View Program Text 10-7
Current State of Every Task 10-7
Event Queue 10-8
System Status 10-8
Trace 10-9
Display Data 10-9
Fault System 10-9
Fault Table 10-10
Event Queue 10-10
Power and Run Cycle Behavior 10-11
Fault Variables 10-11
CPU Non - Critical Run-time Error Configuration 10-12
Changing State Logic CPU Data 10-13
Change Variable Data 10-13
Change Current State 10-13
Set Clock/Calendar 10-13
Force Inputs and Outputs 10-13
Establishing an Online Session with the Controller 10-14
Establish Communications 10-14
Check for a Controller Project 10-14
Check Project Versions 10-14
Online Hints 10-15

Serial Communications 11-1

Programming Port 11-2
Establishing ECLiPS and OnTOP Communications 11-2
Starting SNP Communications 11-3
Starting CCM Communications 11-3
Communicating with an ASCII Device 11-4
Serial Port Parameters 11-4

Setting the Port Parameters 11-4
Serial Port Parameters 11-6
Programming Serial Communications 11-7
SNP and CCM Communications Protocols 11-8
Protocol Data Types and Reference Numbers 11-8
Displaying Protocol Reference Number Mapping 11-8
%R Register Data 11-9
Accessing Analog Data 11-12
Accessing Digital Data 11-12
SNP Access to Digital Data 11-14
CCM Access to Digital Data 11-14

Keyboard Functions A-1
Function Hot Keys A-1
Other Hot Keys A-2
Miscellaneous Keys A-3

Language Structure B-1
Notational Conventions B-1
Program Hierarchy B-2
Functional Structures B-3
Functional Structures Continued B-4
Conditional Structures B-5
Value Expressions B-6

Standard Predefined Keywords C-1
Conditional Terms C-1
Functional Terms C-2
Operators C-3
Operators Continued C-4
Miscellaneous Keywords C-5

Errors D-1
Translation Errors D-1
Runtime Errors D-7
Non-Critical Errors D-7
Critical Errors D-10

Specifications E-1

Figure 2- 1. State Logic Control System 2-2
Figure 2- 2. OnTOP Security Form 2-11
Figure 3-1. Serial Communications Module 3-25
Figure 3-2. Serial Port Connector Pin Assignments 3-26
Figure 3-3. WYE Cable 3-27
Figure 3-4. WYE Cable Pin Assignments 3-27
Figure 4- 1. State Diagrams 4-2
Figure 4- 2. Task Description in ECLiPS 4-3
Figure 4- 3. Sample Task with Some Elements Labeled 4-5
Figure 4- 4. Five State Task Example with a Single State Highlighted 4-6
Figure 4- 5. ECLiPS State with One complete Statement Highlighted 4-7
Figure 4- 6. Highlighted Communication Functions 4-8
Figure 4- 7. Example Diagnostic Task 4-12
Example ECLiPS State with Four Statements 5-6
Statement examples with Functional Terms highlighted. 5-6
Statement example with Conditional Terms highlighted. 5-6
Figure 5- 1. Program Scan 5-10
Figure 5- 2. Statement Scan 5-10
Figure 5- 3. Program Scan with GO Terms 5-11
Figure 5- 4. Statement Scan with GO 5-11
Using multiple to Tasks to keep an Output ON 5-14
State Logic Program Hierarchy 7-2
Functional Expressions in Bold Type 7-3
Conditional Expressions in Bold Type 7-4
Examples of Multiple Conditional Instruction Expressions 7-4
Examples of Multiple Functional Instruction Expressions 7-4
Figure 9- 1. PID Algorithm 9-2
Figure 9- 2. Cascaded PID Loops 9-3
Figure 9- 3. PID Loop Initialization Form 9-6

Table 1- 1. Notation Conventions....1-6
Table 2- 1. 90-30 State Logic CPU Specification Table....2-3
Table 4- 1. Functional and Conditional Terms....4-8
Table 6- 1. The Hot Keys for FIND Functions....6-4
Table 6- 2. Cursor Control Keys....6-4
Table 7- 1. PLC Fault Variables....7-8
Table 7- 2. Functional Instruction Quick Reference List....7-10
Table 7- 3. Conditional Instruction Quick Reference List....7-18
Table 8- 1. Typical Perform Function Parameter Form....8-2
Table 8- 2. Define Table Parameter Form....8-4
Table 8- 3. Swap Table Parameter Form....8-4
Table 8- 4. Initialize Table Parameter Form....8-5
Table 8- 5. Copy Table Parameter Form....8-6
Table 8- 6. Copy Variable to Integer Parameter Form....8-8
Table 8- 7. String Manipulation Parameter Form....8-11
Table 8-8. HSC Command and Data Format (x = Counter Number, cc = Command Code, dd = data, always 0000 for Type A Counters)....8-15
Table 8-9. Operation Codes, where x = Counter Number....8-16
Table 8- 10. Shift Register Parameter Form....8-19
Table 8- 11. BCD Input Conversion Parameters....8-20
Table 8- 12. Output BCD Conversion Parameter Form....8-21
Table 8- 13. Get User Input Parameter Form....8-22
Table 9- 2. PID Command and Status Bits....9-9
List of State Logic Events Appearing in the Event Queue....10-8
Table 11-1. Serial Port Parameters (Default ECLiPS and SCM Settings in BOLD)....11-6
Table 11-2. State Logic SNP and CCM %R Register Data Types....11-10
Table 11-3. PID Loop Parameter Data....11-11
Table 11-4. State Logic SNP Digital Data Types....11-14
Table 11-5. State Logic Data Types and CCM References....11-14

Chapter 1 Preliminaries

This chapter describes some important details that should be completed or reviewed before using this product, including:

- Product Registration
- Technical Support
- How to Use this Manual
- Chapter Summaries
- Manual Notational Conventions

Please take the time to review all of the items in this short chapter.

Items Included with ECLiPS

Check your ECLiPS package that it has the following items:

1. ECLiPS Disks and this Manual
2. ECLiPS Hardware Key
3. CD ROM providing the GE Fanuc PLC Product Library

If any of these parts is missing, contact your local distributor.

Product Registration

There is an easy FAX registration form inserted at the front of this manual. Fill out this form and FAX or mail it to us immediately upon receipt of this product. By registering your product you are assured of receiving the latest upgrade reports and any technical advisories about the product.

Getting Answers to Your State Logic Questions

When questions arise about the State Logic system look to one of the following sources for answers:

1. ECLiPS and OnTOP have a built in help system that can always be accessed by pressing the <F1> function key on your keyboard. This help system is context sensitive, meaning that the help provided changes depending on the location of the cursor on the screen or the highlighted menu option at the moment you ask for help.
2. This manual contains helpful detailed information about most any aspect of creating State Logic control programs. Use the main index at the back of this manual or the table of contents to locate information in this manual about State Logic programming.
3. For specific information about the Series 90-30 hardware, I/O Modules and Option Modules, see the GE Fanuc manuals describing these products. These manuals can be found on the documentation CD provided with the ECLiPS software.
4. GE Fanuc has personnel specially trained throughout the country to provide customer support for ECLiPS and other State Logic products. Contact your local GE Fanuc representative or call the GE Fanuc technical support hotline at 1-800-828-5747.
5. Your local distributor has had training in State Logic products and is there to help answer your questions.
6. On site and regional training classes are available. Contact your local distributor.

How to Use this Manual

This section explains how to use this manual. First the manual organization is explained followed by a description of each of the chapters.

Manual Organizational Categories

To find specific information in this manual, use the following manual organization categories:

- **Setup (Chapters 1 - 3)**
- **Fundamentals (Chapters 4 & 5)**
- **Programming (Chapters 6 - 9)**
- **On-Line (Chapter 10)**
- **Serial Communications (Chapter 11)**

Setup (Chapters 1 - 3)

The **setup** options are covered in the first three chapters. These chapters explain the overall system architecture and how to set up and configure the software and hardware, including I/O modules. Overall system considerations are covered in this chapter.

Fundamentals (Chapters 4 & 5)

These two chapters explain the concepts of State Logic. The State Logic Theory is explained and an explanation of how to plan a State Logic program.

Programming (Chapters 6 -- 9)

The next group of chapters have information on the details of creating a State Logic control program. These chapters describe how to use ECLiPS and how to use all the program instructions.

On - Line (Chapter 10)

This chapter explains all the features of the ECLiPS Debug Mode, which are also the features of the OnTOP software. All interaction between ECLiPS or OnTOP and the controller is handled using these functions, including monitoring I/O and Current States, forcing I/O circuits, checking faults, inspecting system status, and commanding controller operations.

Serial Communications (Chapter 11)

The last chapter describes the serial connection between the State Logic Control System and other devices through the programming port or the Serial Communication Module(SCM) ports. The serial connection between ECLiPS or OnTOP and the State Logic Controller is covered as well as communications between CPU or SCM and some other serial device. CCM and SNP communications are also covered in this chapter.

Brief Description of the Manual Chapters

1. Preliminaries

Initial inspection, registration, help and manual information.

2. Getting Started

This chapter first shows the general architectural view of the 90-30 State Logic Control System. Setup procedures and configuration for both software and hardware components are explained. The hardware configuration for the CPU and simple I/O modules is explained.

3. Option Modules

This chapter describes the option modules. These are the more advanced specialty function modules available for the Series 90-30 control system. Operation and configuration of these modules are described.

4. State Logic Control Theory

This section provides some basics about the underlying concepts and philosophy of State Logic Control. Regardless of what you may already know about State Logic, **it is extremely important that you read this section carefully.**

5. Creating a State Logic Control Program

This chapter explains how a control application is programmed using ECLiPS.

6. ECLiPS Programming Features

This chapter shows the ECLiPS features designed for creating and modifying State Logic Programs.

7. Program Instructions

This chapter explains how to use State Logic instructions.

8. Perform Functions

The Perform Functions are a series of advanced State Logic Language features. This chapter explains how to implement them in a program

9. PID Loops

This chapter explains how to use PID Loops in the State Logic Program.

10. On - Line Features (Debug Mode and OnTOP)

The On - Line Features are designed to view and modify the information being handled by the State Logic Controller. This chapter explains what features are available and how to use them.

11. Serial Communications

This chapter explains using the ECLiPS Programming Port for serial communications to ECLiPS, common ASCII serial devices, and for CCM communications to graphical user interfaces and other host computers.

Appendix A - Function Keys

Appendix A contains a summary of all the function keys.

Appendix B - Language Structural Summary

Contains a list of all State Logic language structures

Appendix C - Predefined Keywords

List of Keywords that come defined with ECLiPS.

Appendix D - Errors

Contains a list of all the State Logic error messages

Appendix E -. ECLiPS Specifications

List of the ECLiPS system capabilities.

Notation Conventions

Throughout this manual the following notation conventions are observed:

Notation	Example	Description
<Angle Brackets>	<Page Up>	Angle brackets enclose references to keyboard keys
ALL UPPERCASE	MAIN MENU	Indicates a menu title
"Quotes"	"DEFINE"	Indicates Options Listed on a Menu
Italics	*GFK-90486*	Identifies references to other manuals or locations in this manual

Table 1- 1. Notation Conventions

Chapter 2 Getting Started

This chapter describes the Series 90-30 State Logic Control System and its setup. A brief description of the hardware and software components is presented plus a synopsis of how these components are connected and configured.

Introduction

There are two main components of the basic Series 90-30 State Logic Control System, the programmer/online interface software and the Programmable Logic Controller (PLC). The programmer/online interface is a personal computer (PC) executing either the ECLiPS programming and online software or the OnTOP online only software. The following sections of this chapter describe these components and their setup procedures. The following figure depicts a higher level PLC system connected to the programming/online interface:

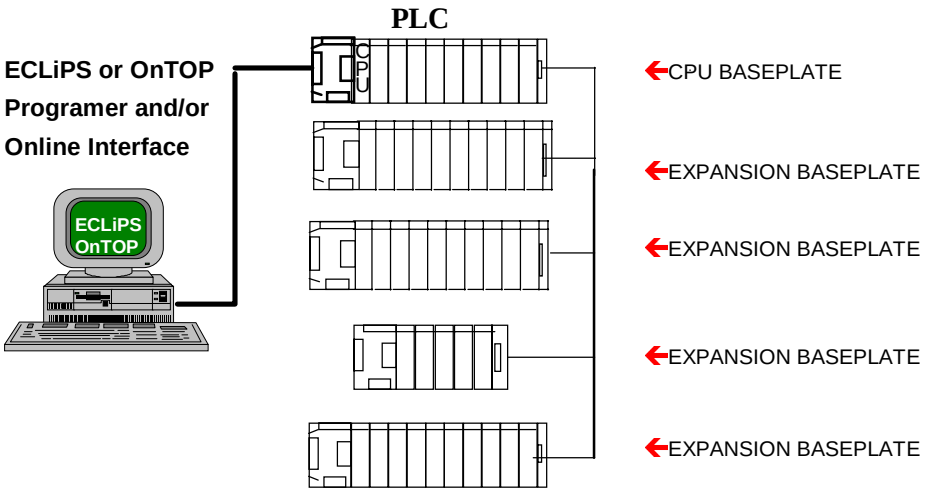


Figure 2- 1. State Logic Control System

The State Logic PLC System

The State Logic PLC System is the hardware that performs all the control and communication functions. The components of the system are the CPU, the PLC baseplates, power supply, and the I/O modules. These components are mostly GE Fanuc products, although third party modules are supported. For detailed information on the hardware or installation procedures, refer to the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

State Logic CPU

The CPU executes State Logic Control Programs, checking inputs and controlling outputs, and sending and receiving communication data. There are five models of State Logic CPUs, the 311, 313, 323, 331, and the 340. The following is a State Logic CPU specification table:

CPU	311	313/323	331	340
Program Memory	10K	22K	48K	98K
CPU Clock Speed	10 MHz	10 MHz	20 MHz	20 MHz
Tasks	256	256	256	256
Task Groups	16	16	16	16
States	2500	2500	3000	3000
States Per Task	254	254	254	254
I/O & Variable Names	3000	3000	3000	3000
Digital Inputs	512	512	1024	1024
Digital Outputs	512	512	1024	1024
Analog Outputs	64	64	128	128
Analog Inputs	128	128	256	256
Internal Flags	500	500	1000	1000
Global Data Bits	1280	1280	1280	1280
Integer Variables	250	250	1000	1000
Floating Point Variables	61	61	497	497
String Variables	8	8	20	20
Characters/String	80	80	80	80
Char Variables	64	64	64	64
Characters / Write	512	512	512	512
Tables	100	100	100	100
Table Memory	1 K	1 K	4 K	4 K
Timers	Unlimited	Unlimited	Unlimited	Unlimited
Timer Resolution	.01 second	.01 second	.01 second	.01 second
Timer-Counters	100	100	100	100
Trace Size	100	100	100	100
PID Loops	20	20	20	20

Table 2- 1. 90-30 State Logic CPU Specification Table

The 311, 313 and 323 CPUs are built into the system baseplate while the 331 and 340 CPUs are a separate module. For more information on the CPU hardware see the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

Baseplates

The baseplates hold all the system modules providing mounting, a distribution of power and communications between the modules. The 311 and 313 systems use only one 5-slot baseplate with the CPU built into the baseplate. The 323 system also has the CPU built into the baseplate, but provides 10 I/O module slots. The 331 and 340 CPUs allow for up to five, 5-slot or 10-slot baseplates, some of which may be remote baseplates, for a maximum of 50 system module slots including the slot for the CPU.

For more information on the functionality and installation of the 90-30 baseplates see the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

Power Supply

Each baseplate of the system has a power supply providing all necessary power for modules installed in the baseplate. There is no configuration required for the power supply.

The power supply module also provides an RS-485 compatible serial port used as the programming port or a connection to other serial devices including CCM devices. This port is active only on the power supply inserted into the CPU baseplate of the system. The port is referred to as the programming port in this manual.

The power supply also comes with a backup battery to maintain CPU RAM memory data when there is no power to the system. On the 331 and 340 platforms, the battery also operates the system clock when power is removed from the system. Only the battery on the power supply in the CPU baseplate is operational.

For more information on calculating power needs or installation of the power supply see the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

I/O Modules

There are six categories of I/O modules: discrete input, discrete output, discrete mixed, analog input, analog output, and option modules. The option modules perform special functions such as motion control, high speed counting, and communications.

For more information on using, wiring, or the installation of the I/O Modules, see the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

Serial I/O Devices

Serial devices are connected to the system through one of the system serial ports. Devices that might be connected are bar code readers, marquee displays, dumb terminals, data acquisition systems, and supervisory and Man Machine Interface (MMI) systems.

Serial devices can be connected to the system programming port after the programming device has been removed. This port provides RS-485 compatible communications using either simple ASCII, SNP, or CCM2 communications protocols.

The 331 and 340 models provide additional serial ports using one or more of the following option modules: SCM, CMM, or PCM. *See the Serial Communications chapter and the Option Module chapter of this manual for more information about using these modules.*

Programming and Online Interface Software

The ECLiPS software provides both programming and online debugging functions for the State Logic Control System. There is also an ECLiPS Lite product, which has all of the functionality of ECLiPS, but which is used only with the 311 CPU. Another Adatek software product for the 90-30 State Logic CPU, OnTOP, provides only the debugging functions.

These products run on an IBM PC/AT or better and connect to the controller using either the COM1 or COM2 serial port.

ECLiPS

ECLiPS is a complete State Logic controls environment providing capabilities for programming, configuring, monitoring, and controlling a State Logic program. The two modes of operation are Program Mode and Debug Mode.

Program Mode

Programming in ECLiPS is done in plain English sentences using the Task/State structure of State Logic as provided by the ECLiPS Program Mode. Programs are entered, checked for errors, and translated for controller execution. Elements of the program, such as I/O points, variables, PID loops, and tables, are also defined and assigned names.

Configuring the modules of the system is also done in Program Mode. When configuring the system, a drawing of the baseplate is displayed and the type of module for each slot is selected. Names for the I/O points of the module and configuration parameters for option modules are defined in the configuration screens.

Debug Mode

The online interface system is the ECLiPS Debug Mode. Since State Logic is a State based system, many of the online tools provide information about the States of the Tasks. A history of State transitions is available using the Trace function and several function display the real time current State values. Of course I/O status and variable values can also be monitored.

The Debug Mode is also used to control the system. The program can be started or halted, data values changed, I/O points forced, and even the current States changed using the functions provided in this mode. *For more information about the Debug mode see the Online Features chapter of this manual.*

OnTOP

OnTOP is a separate software package that has all the capabilities of the ECLiPS Debug Mode. OnTOP is used as an operator interface, maintenance, or troubleshooting tool for situations where it is not necessary for the user to be able to change the program, or where there are many installations and a low cost interface solution is sought. *See the On Line Features chapter of this manual for more information about the OnTOP functions.*

Setting Up the 90-30 State Logic Software

This section describes how to set up the components of the system. Topics explained are:

- ECLiPS Setup and PC Configuration
- OnTOP Setup and PC Configuration
- Connecting ECLiPS to the 90-30 State Logic Controller

Setting Up ECLiPS

This section describes how to get started with the ECLiPS State Logic software. Topics explained are the equipment required, the installation, the hardware key, the memory configuration, and the password protection.

Hardware Requirements

1. IBM PC Compatible
2. 286 processor or better required -- 386 processor recommended
3. 640K RAM -- 4MB of Expanded and/or Extended Memory Recommended
4. 3MB of Hard Disk Space
5. 3.5 inch floppy disk drive
6. Any parallel printer (Optional)
7. Color or monochrome monitor
8. Serial Port

ECLiPS Installation

To install ECLiPS, insert disk 1 into drive A or B and make this drive the current logged drive. Type **INSTALL** and hit <Enter>. The installation program directs you when to insert the other disk into the floppy drive. The installation program displays a message when the installation is complete.

To run ECLiPS, make sure that the directory where ECLiPS is installed, \ECLIPS\CPU30\, is the current directory. Now type **ECLIPS** to start the program.

Note

DO NOT make this directory part of the system PATH and start ECLiPS from another directory. Always log into the directory where the ECLiPS files are stored to start ECLiPS.

Before running ECLiPS, make sure that the CONFIG.SYS file in the root directory of your boot drive has set the FILES and BUFFERS to at least the following minimums:

FILES=20

BUFFERS=20

The contents of the Config.sys file can be checked with the command **Type Config.sys** when logged into the root directory that is used to boot the computer. If the lines FILES = xx and BUFFERS = xx are not present, or are set for less than 20, use a text editor to change the CONFIG.SYS file.

Hardware Key

ECLiPS disks are not copy protected and may be installed repeatedly, but **the license agreement specifies that ECLiPS is to be used on only one computer at a time.**

ECLiPS requires that a hardware key be plugged into the parallel port for normal operations. ECLiPS does function in a limited demo mode when the hardware key is not installed. Operation of ECLiPS without the hardware key is allowed by the license only for demonstration purposes to evaluate the product. DO NOT attempt to use ECLiPS for control operations without having the key installed.

To install the hardware key, plug it into the parallel port (LPT1) and tighten the security screws. The port must be a fully compatible IBM Printer Adapter Card. The key does not restrict or change the use of the parallel port. Any device that normally connects to the parallel port may be connected to the key and all information is transmitted through the key just as if the key were not present.

Note

Devices that use the parallel port such as a bi-directional device, have damaged some hardware keys on some types of personal computers. Devices such as; CD-ROM, Tape Back-UP Systems, or Lap-Link may cause damage to the hardware key. Connect only one-directional devices such as printers to the key when it is installed in the parallel port.

If another software product requiring a similar hardware key is installed on the same computer as ECLiPS, plug the ECLiPS key directly into the parallel port and connect the other key to the ECLiPS key. Any other device may then be connected to the second key.

If there is a need to move ECLiPS to another computer, the distribution disks may be used to install ECLiPS on the other computer and the hardware key moved to the new computer. To return to the original computer, move the key to the first computer.

Note

When using ECLiPS in the Microsoft Windows NT operating system, special drivers are required for accessing the hardware key. Call technical support for information about using ECLiPS in this operating system.

Configuring PC Memory for ECLiPS

The configuration options for ECLiPS and the host personal computer are described in this section.

Using Expanded and/or Extended Memory

ECLiPS makes use of Expanded and Extended memory, if they are available with appropriate memory managers installed. By using expanded and/or extended memory, ECLiPS can be configured for the full number of States and Variables and executes faster. The Expanded memory and memory manager must be version LIM 3.2 or higher, and the Extended memory and memory manager must be version XMS 2.0 or higher.

EMM386.SYS can be used for Expanded memory and HIMEM.SYS can be used for Extended memory. Four megabytes of memory is enough to provide the maximum capacity and speed for ECLiPS operations.

Setup Option

The first ECLiPS screen, the MAIN MENU, shows the "SETUP" option, which is used to configure the ECLiPS system. The two parameters that can be selected are the number of States and the number of I/O and variable names allowed in the program. These selections are for numbers in the range of 500 to 3000 selected in increments of 500.

The maximum limits cannot be selected for both the number of States and number of Names for I/O and variables. The maximum total number for States plus names is 4500. If the ECLiPS computer does not have enough memory for the selected limits, ECLiPS displays an out of memory error message and allows the Setup limits to be selected again.

There are two reasons to select the smallest size limits for the ECLiPS setup options. First ECLiPS requires less free conventional memory with lower defined limits. The second reason is that ECLiPS does some operations faster when using the smaller limits such as translating a project. For slower computers use the minimum settings for best performance.

If there is not enough memory available to load ECLiPS, the Setup screen is displayed to reconfigure ECLiPS for a smaller number of States and/or variables. After these quantities are changed, ECLiPS is restarted.

Using Conventional Memory

Another way to deal with memory problems for ECLiPS, is to free up conventional memory in your PC by eliminating device drivers and Terminate and Stay Resident (TSR) programs. To eliminate these items change CONFIG.SYS and AUTOEXEC.BAT files in the computer. If you are using DOS 5 or better, loading DOS HIGH also frees up additional conventional memory. To use the maximum settings for either the number of States or Variables, DOS must be loaded into the high memory area.

ECLiPS Security

ECLiPS may be configured for password protection. Once a password is established, ECLiPS requires that password be entered before performing any operations. Use this security protection to restrict access to the State Logic CPU and the ECLiPS program files. The password protection only limits entering the protected ECLiPS package and does not limit access to the State Logic CPU by another installation of ECLiPS.

Protection from unauthorized project access by another installation of ECLiPS is a natural attribute of the system. Since ECLiPS cannot connect with the State Logic CPU unless it can access the same version of program files as the control program running in the State Logic CPU, no one can interact with or inspect the State Logic CPU program without having a disk copy of the correct version of the project files.

Setting Up OnTOP

This section describes how to get started with the OnTOP software package. The topics discussed are hardware requirements, installing the software, using the ECLiPS program files, and setting up the security system.

Hardware Requirements

1. IBM PC compatible
2. 286 processor or better required -- 386 processor recommended
3. 640K RAM -- 4MB of Expanded and/or Extended Memory Recommended
4. 1.5 Mbytes of Hard Disk Space
5. 3.5 inch floppy disk drive
6. Any parallel printer (Optional)
7. Color or monochrome monitor
8. Serial Port

Installation

To install OnTOP, place the disk into a floppy drive making that drive the currently logged drive, then type INSTALL. There is no copy protection for OnTOP, but **the license agreement specifies that OnTOP can be used on only one computer at a time**. The installation program displays a message when the installation is complete. OnTOP is installed in the \ECLIPS\CPU30\ directory. To start OnTOP, make this directory the currently logged directory and type ONTOP at the DOS prompt. Do not put this directory in the system PATH and start OnTOP from another directory.

Before running OnTOP, make sure that the CONFIG.SYS file in the root directory of your boot drive has set the FILES and BUFFERS to at least the following minimums:

FILES=20

BUFFERS=20

The contents of the Config.sys file can be checked with the command **Type Config.sys** when in the root directory that is used to boot the computer. If the lines FILES=xx and BUFFERS=xx are not present, or are set for less than 20, use a text editor to change the CONFIG.SYS file and reboot the computer.

Using the ECLiPS Program Files

OnTOP uses the program files created by ECLiPS for on-line interaction with the State Logic CPU. These files provide OnTOP with the actual State Logic program and variable definitions so that information about the operation of the State Logic CPU can be displayed. Use DOS functions to make these files available to the OnTOP software package.

The files that have I/O configuration, names, variable definitions, etc. use the project name with extension PRJ. If the project name were PRESS then the definitions would reside in the file PRESS.PRJ.

The files that contain the program information have extensions TG0 through TGF (TGx indicates the Task Group number). The ECLiPS program may have up to sixteen Task Groups. The first task group is saved in file with extension TG0 and the second in the file with extension TG1, etc., up to TGF.

These ECLiPS files must be on a disk that OnTOP can access, before OnTOP can connect with the State Logic CPU. If OnTOP cannot find the files to match the State Logic CPU program, it asks for another path that it can search to find the files.

To download the program using OnTOP, the actual translated program must be available to OnTOP. The extension of the file containing the translated project uses the project name with the extension (.PSM). This file is created by ECLiPS whenever the project is translated.

OnTOP Security System

The OnTOP Security System is designed for machine operation safety. Through this feature unauthorized access to the control of the machine is prevented. The security system allows up to 4 levels of security for OnTOP. Levels 1 - 3 are customized to limit access to six different functions.

Level number 4 allows unlimited access to all the features in OnTOP, but still the State Logic Program cannot be changed.

To set up the Security System, run ECLiPS from the same drive as OnTOP. Select "Security" from the ECLiPS Program Mode menu and fill in the form. If ECLiPS does not normally execute on the same computer as OnTOP, load ECLiPS temporarily on this computer just to set up the passwords for OnTOP.

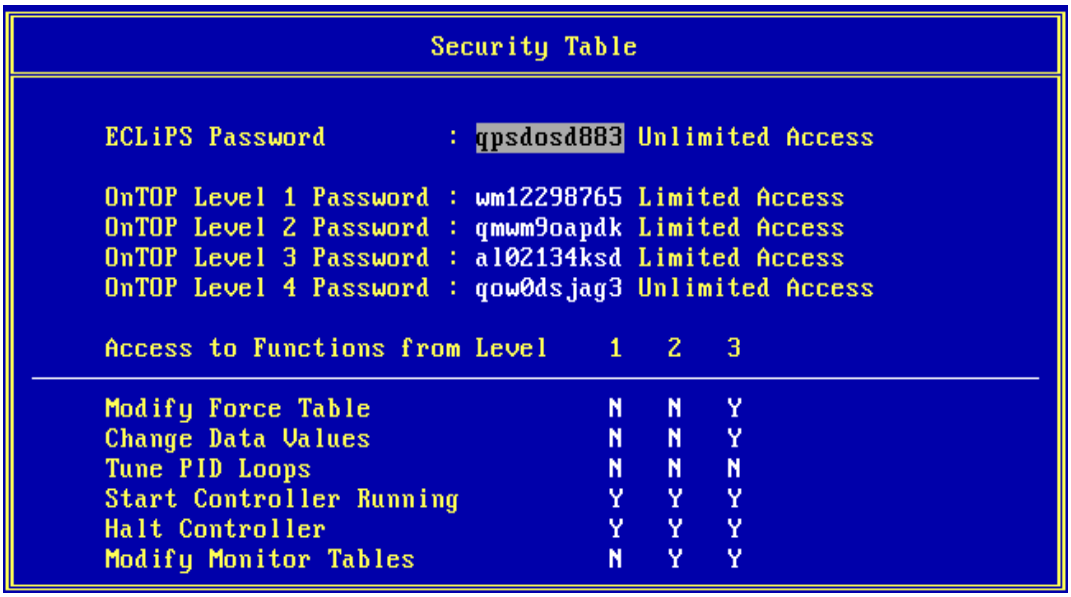


Figure 2- 2. OnTOP Security Form

Use the arrow keys to move the highlighted cursor to the OnTOP Level 1 Password block in the security form. Enter the password. Move the cursor to Level 2, 3, 4 and enter the appropriate passwords. Move the cursor to the Function Access Table and enter a "Y" or an "N" to activate or deactivate the function listed for each level of security access. Also use this form to make any changes to an existing security system.

Connecting ECLiPS and OnTOP to the Controller

Both ECLiPS and OnTOP connect to the State Logic CPU through the 15-pin serial port located on the power supply for the CPU rack (baseplate). One serial cable option is the mini-converter kit which comes with a serial cable and adapters that are required to connect the PC to the controller. *See the Serial Communications Chapter for more information on serial cables.*

When using the mini-converter kit, plug the 15-pin to 9-pin adapter, part number HE693SNP232, to the serial port on the 90-30 CPU rack power supply. Plug one end of the serial cable to this adapter and the other end to COM1 or COM2 of the computer running ECLiPS or OnTOP.

No configuration of the serial ports is required to communicate normally. The State Logic 90-30 PLC programming port configuration parameters may be changed from instructions in the State Logic program if it is necessary to connect that port to another serial device. Be aware that changing the parameters may require the port to be reset before communicating with ECLiPS or OnTOP again. *See the serial communications chapter for information about communicating with other serial devices.*

Configuring the State Logic Control System

The 90-30 State Logic Control System is configured in the ECLiPS Program Mode. The items that are configured are the State Logic CPU, I/O modules, option modules, and system racks. This section describes configuring the CPU, system racks, and the I/O modules. The option module configuration is described in the chapter describing the option modules.

All the configuration options are accessed through the "System Configuration" option from the DEFINE menu in Program Mode. Selecting this option displays a representation of the CPU rack (baseplate), showing each of the slots. To configure a module for a specific slot, highlight the slot using the arrow keys and press <Enter>.

The rack configuration options are only available for 331 and 340 CPU configurations. To configure the system racks, select the CPU slot, press <Enter>, and select the “Modify Rack Sizes” option from the menu displayed. Select the size for each of the racks in the system. After more than one rack is configured, use the <Page Up> and <Page Down> keys to configure other racks in the system.

The configuration and I/O names completed for another project can be reused by selecting the "Import" option from the PROJECT menu. The configuration can also be printed by selecting the “Rack/Slot Configuration” option from the PROJECT PRINT form.

All configuration information is sent to the State Logic CPU each time the State Logic program is downloaded. If a change is made to the configuration, the project must be translated and downloaded again for the changes to take effect.

State Logic CPU Configuration Options

To configure the system CPU, highlight the CPU slot and press <Enter>. Choose the "View/Modify Configuration" option to display the CPU configuration form. Each of the options on this form are explained in the following sections.

CPU Model Number

The first option of the CPU configuration form is the Model Number. Press any key and a list of CPU numbers is displayed. Select the appropriate model number from the list.

When changing the CPU model from a 331 or 340 to a 311, 313, or 323, the module types and variable and I/O numbers are checked for validity for the new less capable model. If there are invalid modules configured or a variable or I/O reference is too large the change is aborted.

The exception is the analog output, %AQ data type. All analog outputs are deleted when downgrading the CPU model configuration. A warning option is displayed before the analog outputs are deleted.

Automatic Program Execution on Power Up

The CPU can be configured to automatically run the program on power up. When set to run on power up, no operator actions are required to have the program start running when power is applied to the system.

Note that Tasks can be configured to start in the State that was last active, by following the Task name in the program with the keyword "Start_In_Last_State." Also, variables and tables can be configured to hold their values over a power cycle. This option is specified when the table or variable is defined.

PID loops and time counters do not execute automatically when the program starts, even though they might have been running when the program stopped. The PID loops must be started by the program. The time counters must be assigned and restarted by program execution.

Note

The CMM, PCM, and SCM modules take from 10 to 15 seconds to power up. Systems configured to run automatically on power up should delay interaction with these modules for at least 15 seconds after power is applied to the system.

Runtime Fault Response

There are two categories of runtime errors, critical and non-critical. The system can be configured to halt or just send a message and continue program execution when these errors occur. Enter 'D' in the form to continue running and 'H' to halt the program. *See Appendix D for a list of runtime errors. Information on the fault handling system can be found in the Online Chapter.*

ECLiPS Port Name

The serial port that comes with the power supply is the interface to the ECLiPS software and is therefore referred to as the ECLiPS Programming Port. The default name is Eclips_Port, but can be changed using this CPU Configuration form. This name is used with the READ and WRITE program instructions for serial I/O operations.

Configuring System Racks

Only the 331 and 340 CPUs support more than one rack and different rack sizes in the system. Once the 340 or 331 model CPU is selected, and then the "View/Modify Configuration" option for the CPU slot is selected again, two different options are displayed; “Modify State Logic Controller Params”, which displays the CPU configuration form and “Modify Rack Sizes”, which configures the system racks.

The racks of the system are numbered with rack 0 being the CPU rack. Select the rack size for each rack in the system.

Configuring PLC Modules

Specific modules are selected for slots in the rack(s) of the system. To configure the modules that are inserted into the PLC slots, select the “System Configuration” option from the DEFINE menu. A picture of the PLC slots is displayed.

To configure a slot in the system, highlight the appropriate slot, press <Enter>, then select the “Configuration” option. Select from a list of module categories and then from the list of available modules types in that category.

After a module is selected for a slot, configuration information is entered in the forms that follow. Many modules only require the starting reference number , while other modules especially the Option Modules require more extensive configuration. *Configuration of the option modules is described in the Option Module chapter.*

For more information on the Series 90-30 modules and their configuration parameters, refer to the Series 90-30 Programmable Controller Installation Manual, GFK-0356.

Configuring Digital and Analog I/O Modules

To identify the type of module used in a slot, select the "Configuration" option from the SLOT OPTIONS menu while that slot is highlighted. Next select is the signal type for the module (Digital Input, Analog Output, etc.). Now select the module part number from the list provided. The Generic options are provided for GE Fanuc modules that may be developed in the future. Next enter the Starting Reference Number in the blank provided.

Starting Reference Number

Each I/O circuit is identified by a State Logic name and a reference number. A reference number is preceded by a per cent sign (%) followed by letters indicating the I/O type:

%I - Digital Input

%Q - Digital Output

%AI - Analog Input

%AQ - Analog Output

The circuits of each I/O module are represented by consecutive reference numbers, one for each circuit. The starting reference number identifies the number for the first circuit of the block. For example, a 16 circuit digital input module might use %I17 to identify its first circuit. The modules circuits for this block are numbered %I17 - %I32. ECLiPS automatically enters the lowest unused number, in the Starting Reference Number blank, but that number can be changed to any legal number not previously assigned.

I/O Names

After the module has been configured, the “I/O Names” option on the SLOT OPTIONS menu displays names and other configuration data for the module circuits. This form has blanks for

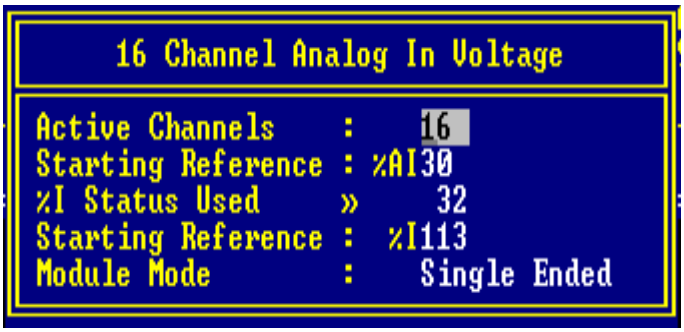
other configuration data depending on the data type; outputs can be configured to be retentive over a run cycle and analog circuits can be scaled. This form may be used to enter new data or just to display the names and data configured in other screens.

High Density Analog Modules

Release 3.0 supports two high density analog modules; the 16 channel input and the 8 channel output modules. The following sections describe the configuration of these modules.

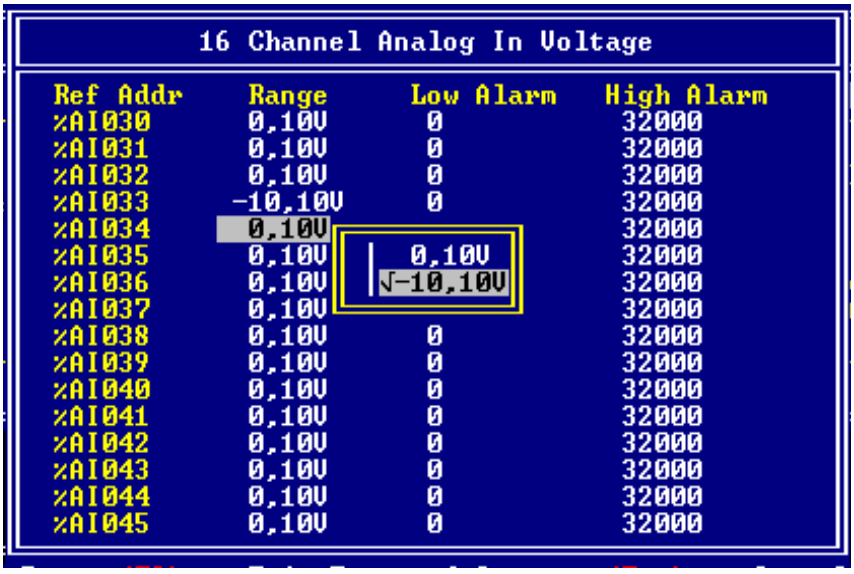
16 Channel Analog Input Module

The 16 channel analog input modules have more features than the other analog modules and therefore have more configuration selections. After selecting a 16 channel analog input module, a configuration form is displayed to configure reference numbers for analog channels and status bits plus a mode field.



Entering the exact number of Active Channels, saves program scan time, if not all 16 channels are used. Fill in the form then press <F9> to save the entries.

Now another form is displayed to configure the range and alarm values for each circuit of the module.

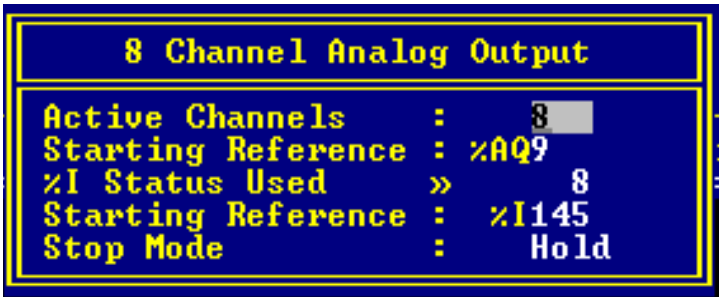


Pressing any key with a range value highlighted, displays a list of options. The Low Alarm and High Alarm options are raw values not scaled engineering units. Fill in this form and press <F9> to save the entries.

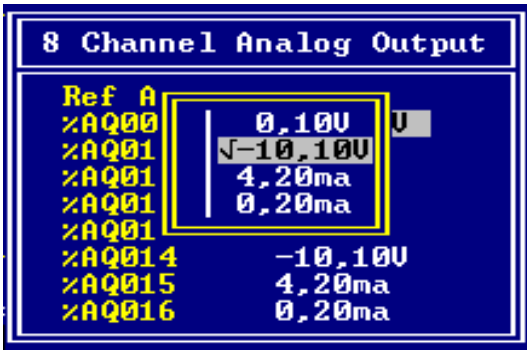
Now that the module is configured, the "Extended Configuration" option on the SLOT OPTIONS menu displays the range and alarm form. The "View/Modify Configuration" option displays reference number form. The "I/O Names" option is used to access analog channel names and scaling data and %I status bit names. After "I/O Names" is selected, choose whether to view analog names or status bit names.

8 Channel Analog Output Module

The first form displayed after selecting the 8 channel analog output module is used to select the reference numbers for the analog channels and the %I status bits, and the Stop Mode.



Entering the exact number of channels used, if not all 8 are used, saves scan time. Complete the form and press <F9> to save. The next form displayed is used to select the range for each one of the output channels.



Pressing any key when the desired range is selected, displays a list of valid options. Press <F9> to save when the form is completed.

Now that the module is configured, the "Extended Configuration" option on the SLOT OPTIONS menu displays the range form. The "View/Modify Configuration" option displays the reference number form. The "I/O Names" option is used to access analog channel names, retentive status, and scaling data and also the %I status bit names. After "I/O Names" is selected, choose whether to view analog names or status bit names. The form displaying analog names provides blanks for specifying scaling configuration data.

Chapter 3 Option Modules

The Option Modules are special function or “intelligent” modules. This chapter discusses configuration, operation, and interface with the program and State Logic CPU. The option modules are:

- **High Speed Counter (HSC)**
- **Enhanced Genius Communication Module (GCM+)**
- **Communications Module (CMM)**
- **Ethernet Interface Module**
- **Genius Bus Controller (GBC)**
- **Axis Positioning Module, 1 or 2 axis (APM)**
- **Programmable Coprocessor, 300, 301, and 311 models (PCM)**
- **Serial Communications Module (SCM)**
- **Foreign or Third Party Modules**

To find detailed information about the GE Fanuc modules and their configuration, see the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

High Speed Counter Module

The High Speed Counter (HSC) is used to process rapid pulse signals of up to 80 kHz, communicate pulse data to the CPU, and provide output circuits that can be configured to respond directly to the pulse data without interaction with the controller. This module is supported by all State Logic CPU models.

Typical applications for this module are:

- Flow Meter Values
- Velocity Measurement
- Motion Control
- Material Handling

See the Series 90-30 High Speed Counter User's Manual, GFK-0293 for detailed information about this module.

Accessing HSC Data

There are three types of data values passed between the State Logic program and the HSC, register values, status bits, and command bits. Register values sent from the HSC to the controller are referenced in the State Logic program as analog inputs (%AI), the status bits which provide information about the counter module are referenced as discrete inputs(%I) , and command bits which are used to control different counter functions are referenced as discrete outputs(%Q). There are 16 status bits and 16 control bits and 15 register values available for each HSC module.

Some of the register data returned by the HSC is double integer data requiring 2 words or 4 bytes of memory. The double integer %AI values are represented as floating point numbers in the State Logic program. Some Type 'B' and 'C' data is double integer, but all Type 'A' counter data is integer.

Note

The IEEE 32 bit floating point data format used by the State Logic Control System represents accurate integer data up to 33,554,432. Values above this number start to lose accuracy, since there are some numbers that can no longer be represented by this floating point format. Therefore, the HSC should be configured so that it does not generate values in excess of 33,554,432. For example, the accumulator should be configured to reset to zero before reaching the range where accuracy is lost.

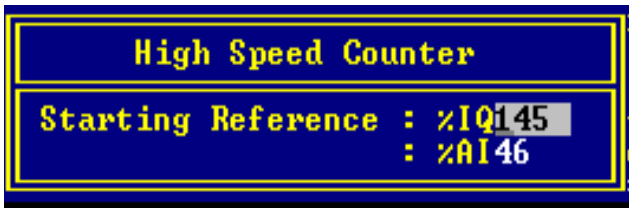
Configuring The HSC Module

There are several configuration options and screens for the HSC:

1. Starting Reference Numbers
2. Extended Configuration
3. Names

Starting Reference Number

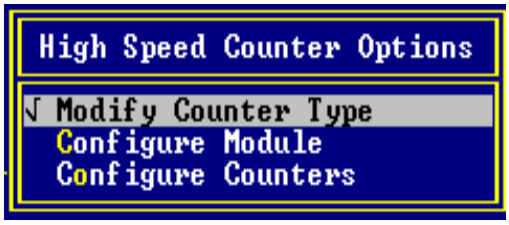
After the HSC module is selected from the list of option modules, the first screen accepts the discrete and analog starting reference numbers. The %I and %Q data uses the same reference number. The blank for this value is labeled as %IQ.



This form is also displayed when the "View\Modify Configuration" option is selected from the SLOT OPTIONS menu.

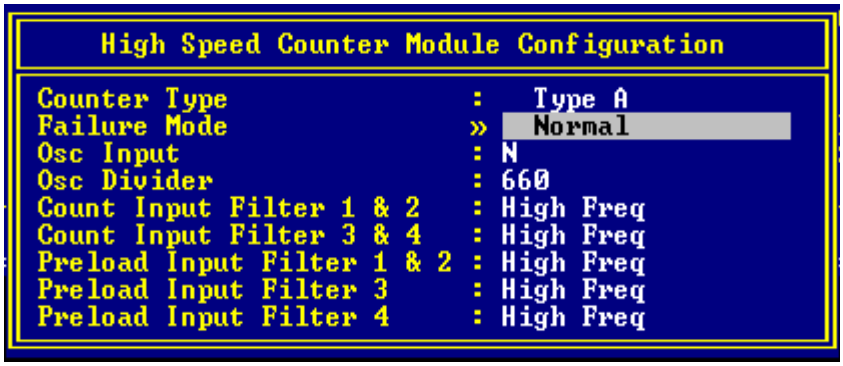
Extended Configuration

The "Extended Configuration" option on the SLOT OPTIONS menu displays another menu providing access to counter type selection, configuration options for the entire module, and configuration options for the individual counters.



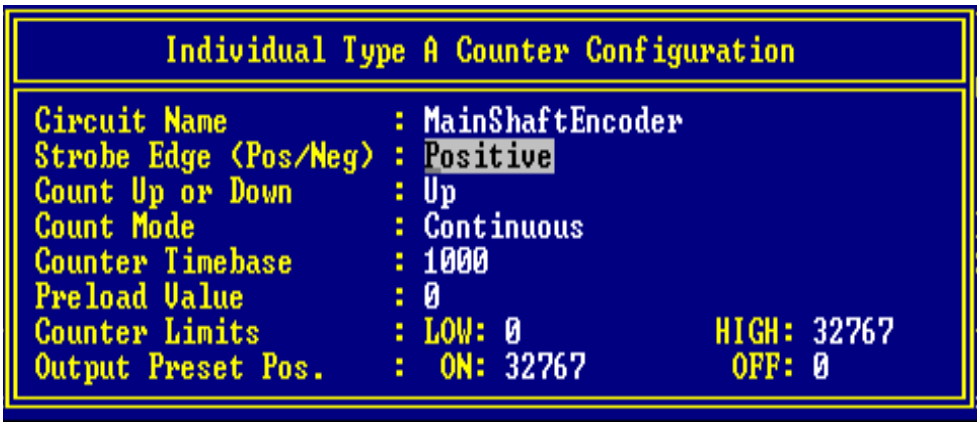
The counter can be type A, B, or C. Select the type for your application.

The "Configure Module" option displays a screen of detailed configuration options for all of the counters. The options display differs depending on the counter type selected.



The “Configure Counters” option is used to configure options for each individual counter. The first form provides blanks to enter names for the accumulator for each counter. This is an important screen, since it provides access to more individual counter configuration options.

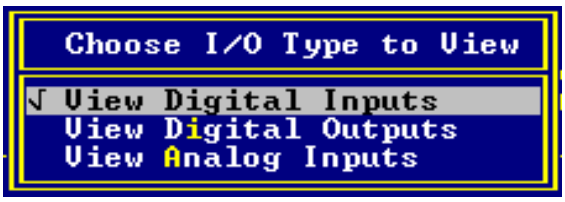
To configure the individual counter options, press the <Alt + F9> key with the desired counter name highlighted.



The configuration options displayed on this screen vary depending on the type of counter being configured.

I/O Names

The “I/O Names” option on the SLOT OPTIONS menu enables names to be viewed and changed for all of the I/O points transferred between the CPU and the HSC. First select the type of name to be displayed.



The names for each of the data types are displayed in addition to the retentive status of the digital outputs.

Programmed Configuration Changes

The HSC can be reconfigured from the State Logic program as the system is operating. There are two different Perform Functions that can execute this operation. One is the **Send_HSC_Data** Perform Function and the other is the **Comm_Request** Perform Function. Both of these functions have the same capabilities.. Use the **Send_HSC_Data** Perform Function, since it is easier to use *See the discussion of these function in the Perform Function chapter of this manual and also the Series 90-30 High Speed Counter User's Manual, GFK-02*93.

Note

The **Comm_Request** Perform Function supports only the register data type, which is number 08. Also, the NO WAIT Com_Request is not supported; only the WAIT Com_Request can be used.

Enhanced Genius Communication Module (GCM+)

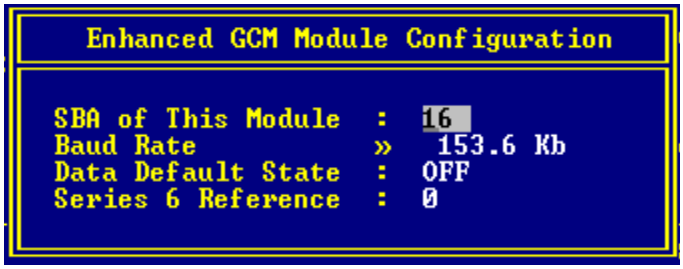
The Enhanced Genius Communications Module (IC693CMM302) provides automatic global data communications between the Series 90-30 PLC and up to 31 other devices on a Genius bus. The GCM+ can receive global data broadcast by other devices and/or broadcast data to the other devices. For more detailed information refer to the *Series 90-30 Enhanced Genius Communications Module, GFK-0695.*

Configuring the GCM+

To configure the GCM+ module follow these steps:

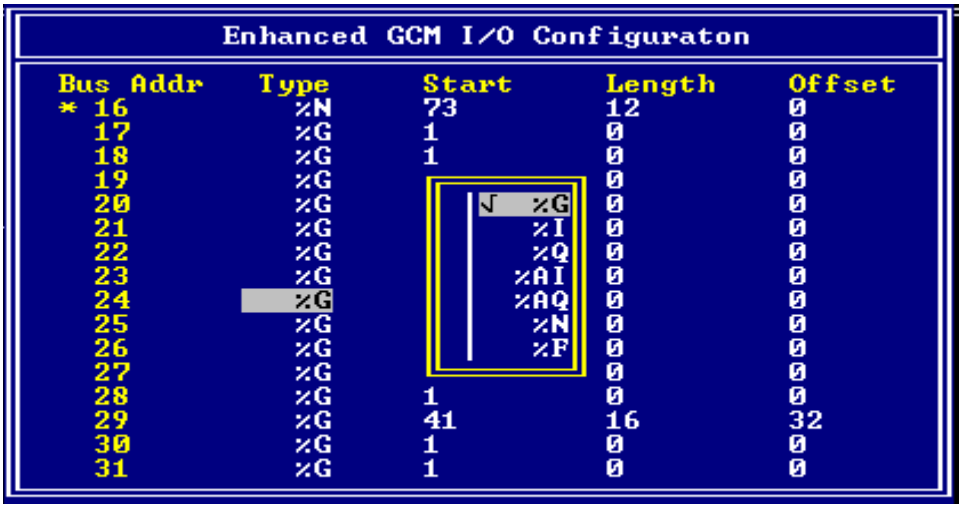
1. Highlight Appropriate Slot
2. Select the GCM+ from list of "Option Modules"
3. Fill in the Starting Reference Number blank for the module status bits on the first configuration screen displayed.

4. Fill in Module Configuration Form for all of the module options.



5. Select "I/O Names" option from the SLOT OPTIONS menu to name the %I Status Bits being used in the program.

6. Fill in the I/O Transfer Form.



The left most column in this form represents the Genius Bus Address (SBA) of the device broadcasting global data. The SBA marked with an asterisk is the SBA of this GCM+, and the configuration in this row identifies the data type and location of data being broadcast by this GCM+. All the other rows specify the data type and storage location of data received by this GCM+ from the device at the SBA specified for this row.

Only half of the SBAs are displayed at a time. Press <Page Up> or <Page Down> to access the rest of the SBAs.

Pressing any key when one of the **Type** column entries is highlighted, displays a list of possible data types. The digital types are %G, %I, and %Q. The integer variables are type %N; floating point variables are type %F; and analog types are %AI and %AQ**.**

Note

The %AI and %AQ data is always the unscaled raw data as it is received from the analog I/O module. Another device cannot change analog output data directly through the GCM+ using the %AQ reference number. Send the data to an integer or floating point variable and then transfer the value to the %AQ in the control program.

Caution

Forced analog input data is not accessible using the %AI references. Instead of the forced value, the actual value at the module is transmitted as global data. To access forced analog input data, assign the %AI value to a numeric variable in the program and transfer the variable instead of the %AI.

The **Start** column specifies the beginning reference number of the data sent or received. For data being received this number specifies where the incoming data is stored.

The **Length** column specifies the amount of data transferred. This value represents the number of bits for the digital types, the number of words for integer variables and analog types and the number of double words for the floating point data type.

The **Offset** column is used for data being received only and is an optional configuration value. The offset allows the GCM+ to receive only a portion of a broadcast message. The offset value always specifies the number of bytes. The number of bytes specified by the length plus the offset must never exceed 128, since this value is the maximum amount of data that can be transmitted via global data.

The form displayed above shows that this GCM+ has an SBA of 16, indicated by the asterisk, and is broadcasting 12 integer variables, starting at integer variable number 73. This GCM+ is also receiving 16 bits of data being broadcast from the device at SBA 29. The data received starts 32 bytes after the start of the data being broadcast by the device at SBA 29. This data is being stored in 16 %G bits starting at %G41.

%G bits can be either inputs or outputs, depending on the configuration of the GCM+ and the Genius Bus Controller (GBC) configuration screens. %G bits that are being broadcast by a GCM+ or by a Genius Bus Controller as global data are treated as outputs. All other %G bits are inputs.

%G bits that are outputs are always OFF unless turned ON by the program, just like the %Q bits are always OFF unless turned ON somewhere in the program or forced ON. Therefore, if a %G bit is turned ON by some external device, that bit is ON for only one scan. The State Logic operating system immediately turns it back OFF.

Note

Although there may be up to eight total GCM+s plus GBCs in a system, there may be only two ranges of %G outputs. If more than two of these devices have %G outputs, make the output ranges contiguous so there are only two ranges.

Note

The GBC+ does not transmit or receive any data when the State Logic program is in simulation mode.

Communications Coprocessor Module (CMM)

The CMM module (IC693CMM311) provides two serial ports to interface between the State Logic PLC and serial devices such as Man Machine Interfaces (MMI), Graphical User Interfaces (GUI), and other Level 2 devices. The CMM supports master, slave, and peer-to-peer capabilities for the CCM protocol, master and slave for the SNP protocol, and slave only RTU protocol.

For more detailed information on the Communication Coprocessor Module see the Series 90 PLC Serial Communications User's Manual, GFK-0582.

Sending and Receiving Data

State Logic provides a more extensive set of data types than traditional PLCs. The extra data types are the current State of a Task, string variables, character variables, and scaled analog data

in floating point format. Although the protocols supported by the CMM do not support this extended set of data types, the State Logic system does provide access to this data through the CMM by allowing the data to be referenced as traditional register data.

Note

The CMM module does not transmit or receive any data when the State Logic CPU is in simulation mode.

ECLiPS prints an SNP mapping of all of the State Logic data. Select the "SNP Protocol Listing" option on the ECLiPS Print Project Data form to print out this information. The "CCM Protocol Listing" option is used only for the programming port on the power supply, and not for the CMM or the PCM modules. Use the "SNP Protocol Listing" option for all CMM protocols, CCM, RTU, or SNP. *See the SNP section in the chapter on serial communications in this manual for more information about using communication protocols with State Logic systems.*

Normally the CCM protocol cannot access analog data, but State Logic provides CCM access to analog data by mapping that data to register locations. If the analog channel is not scaled, the register location holds the unscaled value in the first of two registers reserved for each analog channel.

The SNP protocol provides another access to analog data using %AI and %AQ references. The %AI and %AQ data is always the unscaled raw data, use the %R register analog references to access scaled analog data.

Note

To change analog output values, use the register location for the analog data. Analog outputs cannot be changed through the CMM module by using the %AQ reference.

Caution

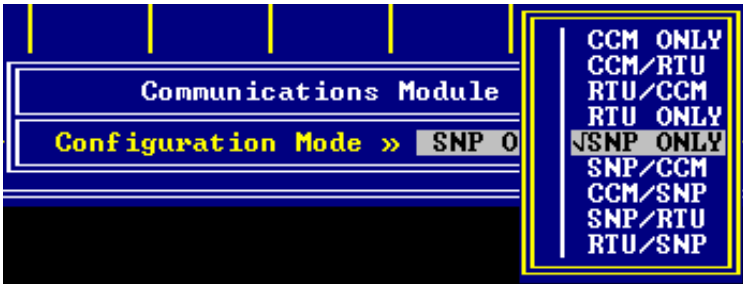
Forced analog input data is not accessible using the %AI references. Instead of the forced value, the actual unscaled value at the module is accessed. To access forced analog input data use the register references for the channel.

Configuration

To configure the CMM follow these steps:

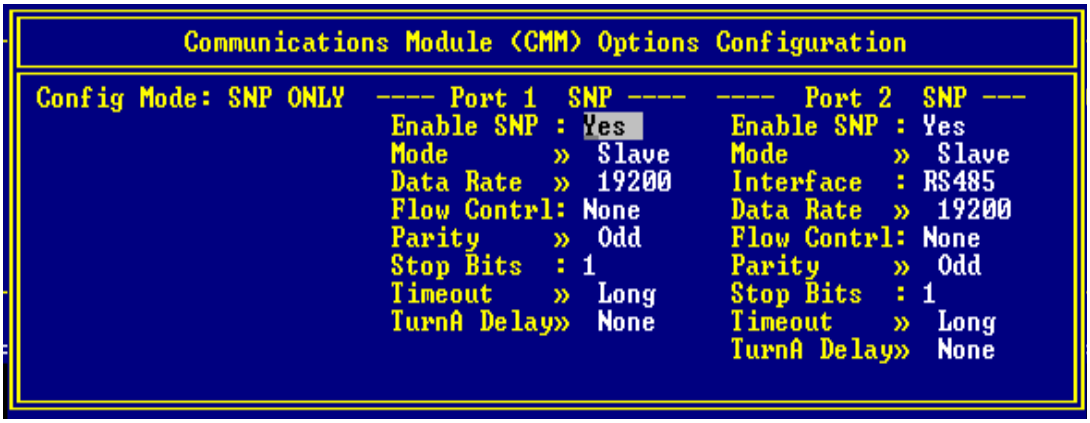
1. To configure the CMM, first select a slot in rack 0. The CMM must be installed in rack 0 and requires the CPU model to be a 331 or 340
2. Next select its part number from the list of Option Modules.

3. The first screen is used to select the protocol for each of the serial ports.



The protocol listed before the slash is the protocol for port 1, and the last one is the protocol for port 2. Select the desired option and press <F9> to save.

4. The next screen is a form for setting the configuration options for each of the ports. Fill in this form to complete the configuration.



Note

The information in the Serial Communications chapter on using CCM protocol to transfer digital data applies only to the programming port and the ports on the SCM module. Use the SNP descriptions to access CPU data through the CMM, Ethernet, GBC, and PCM modules.

5. There are two ways to set up the SNP ID number when the SNP protocol is used on the CMM module.

    1. Set up the programming port SNP ID number, from the Debug Mode, PROJECT menu options. Then transfer the ID to the CMM by either pressing the reset button or power cycle the system.

    2. Change the SNP ID by sending the CMM module the Comm_Request which assigns the SNP ID number.

Sending Data as a Master

When acting as a master through the CMM, the State Logic program uses The Comm_Request perform function to send messages. *See the Perform Functions chapter of this manual for more information on the Comm_Request Perform Function.*
The following programming example uses the CMM port as a CCM master, requesting data from a CCM device with a CPU ID of 2.

State Logic Program Example:

```
State: FirstInit
   Stat_Loc=0.      ! Set status location to zero prior to sending commreq

!!!!  Names word1 thru word12 are defined as consecutive integer variables
   word1=6          ! Length of Data Block
   word2=0          ! Wait/No Wait status  always 0 for State Logic
   word3=8          ! Memory Type of Status Word always 8 (Integer Variable) for State Logic
   word4=45         ! Location of status word (stat_loc) zero based
   word5=0          ! Always 0 with State Logic
   word6=0          ! Always 0 in State Logic
   word7=6101       ! Command Number - First Word of the Data Block Command for the Comm Request
   word8=2          !  Target CPU ID Number - Second Word of data block
   word9=1          ! Target Memory Type
   word10=1         ! Target Memory Location
   word11=9         ! Number of points
   word12=101.      ! Source Memory address

   Go to WriteComm State.

  State: WriteComm
! This State sends the Comm Request

Perform Comm_Request with
    Rack=0,
    Slot=3,
    Task_ID=1,
    Data_Block=word1.

    Go to CheckStatus State.   ! Go to State that checks Stat_Loc status variable
```

The next example is a program segment that uses the CMM port as an SNP master. The Comm_Request function is sending an SNP attach message.

State Logic Program Example:

```
State: SetPort1
   Stat_Loc=0.     ! Set status location to zero prior to sending commreq

!!!!  word1 thru word11 are defined as consecutive integer variables
   word1=7         ! Length of Data Block
   word2=0         ! Wait/No Wait status  always 0 for State Logic
   word3=8         ! Memory Type of Status Word always 8 for State Logic
   word4=45        ! Location of status word (Stat_Loc) zero based
   word5=0         ! Always  0 for State Logic
   word6=0         ! Always  0 for State Logic
   word7=7200      ! First Word of the Data Block  Command for the Commreq
   word8=0         !   address of remote plc
   word9=0         !   address of remote plc
   word10=0        !   address of remote plc
   word11=0        !   address of remote plc
   go WriteComm.

  State: WriteComm
! This State sends the Comm Request

  Perform Comm_Request with
    Rack=racknum,
    Slot=slotnum,
    Task_ID=PortNumber,          ! CMM Port Number being Used
    Data_Block=word1.

   go CheckComm.   !  Go to check status variable Stat_Loc
```

See the Series 90- PLC Serial Communications User's Manual, GFK-0582, for more information on the CMM Comm_Requests.

Note

The State Logic CPU uses only NOWAIT Communication Requests. The WAIT Communication Requests are not supported. Also the type of the data sent or received must be register, data type number 08.

Note

The CMM takes from 10 to 15 seconds to power up. Systems configured to run automatically on power up should delay interaction with the CMM for at least 15 seconds after power is applied to the system.

Ethernet Interface Module

The Ethernet Interface Module (IC693CMM321) provides an interface between the State Logic PLC and an Ethernet network. Data is transferred between the CPU and the Ethernet network using this module.

Note

Program downloads and online functions using the Ethernet Interface Module are not currently supported.

Retrieving Data from the State Logic Control System

State Logic provides a more extensive set of data types than traditional PLCs such as current State of a Task, string variables and analog data scaled to floating point precision. Although the protocols supported by the Ethernet Interface Module do not specifically support this extended set of data types, the State Logic CPU does provide access to this data by allowing these values to be referenced in the connecting device as register data.

To see how the State Logic data are mapped to register locations, use the “SNP Protocol Listing” option on the ECLiPS Print Project Data form. This print out displays the data types and reference numbers to use when accessing State Logic data using the Ethernet Module. *See the SNP section in the chapter on serial communications in this manual for more information about using communication protocols with State Logic systems.*

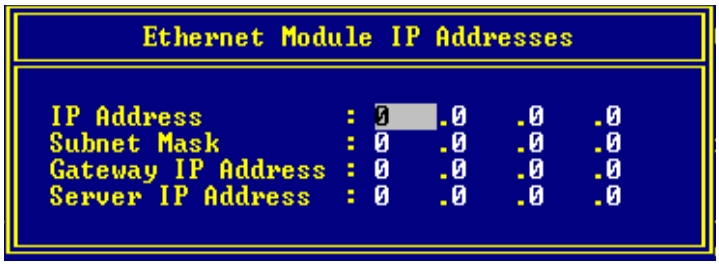
Note

The Ethernet module does not transmit or receive any data when the State Logic controller is operating in Simulation Mode.

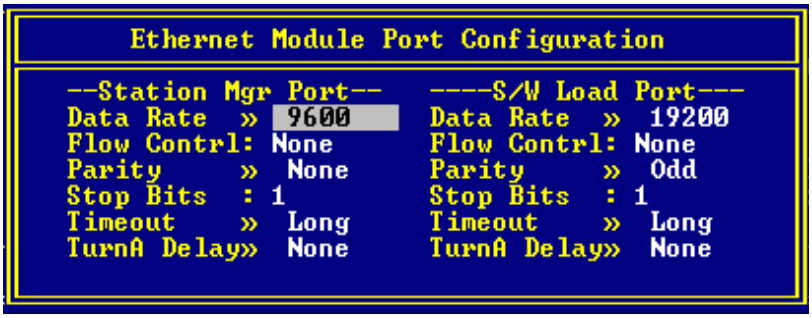
Configuration

To configure the Ethernet module, follow these steps:

1. Select its part number from the list of option modules.
2. The first screen provides a blank to enter the starting reference number of the %I status bits for this module.
3. After this number is entered, select either the "IP Addresses" or "Port Configuration" option from the next menu.
4. The IP Address form:



5. The Port Configuration form:



6. Select the "I/O Names" option from the SLOT OPTIONS menu to enter or view the names of the %I status bits for LAN or Channel status of the module.

See the TCP/IP Ethernet Communications for the Series 90-30 PLC User's Manual, GFK-1084, for more information about this module.

Sending Data Using the Comm_Request Function

When the PLC system is used as a server, all communications are initiated by a client device. Responses to requests for information are handled automatically and therefore require no programming.
A system used as a client, initiates communications by sending messages generated by the control program. Messages are sent from the State Logic CPU through the Ethernet Module by executing a Comm_Request Perform Function in the State Logic program. *See the Perform Functions chapter in this manual for more information on the Comm_Request Perform Function.*

The following program example sets up channel 1 of the Ethernet module to write 9 integer variable values to a remote PLC every 3 seconds. The variable values sent start at integer variable 101 and are sent to the register location 201 in the remote PLC.

State Logic Program Example:

```
State: InitWords
   count=1
  Stat_Loc=0

!!!!!  The names Word1 thru Word23 are defined as consecutive integer variables
   word1 = 17   ! command data length
   word2 = 0    ! wait/no wait  always 0  - State Logic support only NOWAIT Comm_Requests
   word3 = 8    ! status word data type always 8 for State Logic
   word4 = 10   ! status word offset zero based -   Integer Variable 11 named Stat_Loc
   word5 = 0    ! reserved always 0
   word6 = 0    ! reserved always 0
   word7=2004       ! command to set up a write channel
   word8=1         ! Channel number 1
   word9=0         ! Number of write repetitions (write indefinitely)
   word10=3        ! Time units for write period (3 = seconds)
   word11=3        ! Number of time units (update every 3 seconds)
   word12=100      ! Timeout for Each Write (1 second)
   word13=8        ! Local PLC data type (always int for State Logic)
   word14=101      ! 1 based offset Location of data to write from  (Local PLC integer 101)
   word15=8        ! Target device data type - can be any type (8 for integer data)
   word16=201      ! 1 based offset location to write data to (Target PLC Register 201)
   word17=9        ! Number of units to write 9 integers
   word18=1        ! Address type  (always 1)
```

```
  word19=4        ! Address length in words (always 4)
  word20=6        ! Next 4 words ADDRESS OF DEVICE TO SEND DATA TO  (6.9.6.111)
  word21=9
  word22=6
  word23=111.

  Go to the WriteComm State.

State: WriteComm    ! This State sends the Comm_Request
 Perform Comm_Request with
   Rack=0,                  !  Rack 0
   Slot=3,                 !  Slot 3
   Task_Id=0,               !  Always 0 for Ethernet
   Data_Block=word1.          !  First Word of Data Block

  Go to the CheckComm State.   ! Go to a State that checks the success of the Comm_Request
```

Note

Memory type references for all State Logic data whether client or server must by number 8 (Register). Also only the NOWAIT Comm_Request is supported in the State Logic CPUs. The WAIT version is not supported.

See the TCP/IP Ethernet Communications for the Series 90-30 PLC User's Manual, GFK-1084, for more information about this module.

Genius Bus Controller (GBC)

The Genius Bus Controller (GBC) (IC693BEM331) allows the PLC to control and communicate over a Genius Bus. Using this module, the State Logic CPU can interact with Genius blocks and Field Control Devices, send datagrams, and broadcast and receive global data.

The State Logic program interacts with the I/O points on the Genius bus just as it does with the 90-30 rack I/O. Each point is given a name and a reference number. That reference number is tied to specific devices on the Genius bus during the configuration process.

Global Data

Global Data is data which is automatically broadcast by the GBC or GCM+ every scan. All other global data devices on the bus are capable of receiving this data. Sending and receiving global data requires no programming. Once configured, the system transfers and receives the data automatically.
In the configuration section above, step 9 describes setting the system up to broadcast global data. Step 10 explains how to receive global data from another controller on the bus. *See the Genius I/O System and Communications User's Manual, -GFK-90486, for more information about Global Data.*

Datagrams

Datagrams are messages sent from one device on the Genius Bus to one or more other specified devices on the bus. Datagrams are directed to specific bus addresses in contrast to global data, which is broadcast to all devices.

From the 90-30, only the GBC can send Genius Bus datagrams. The CPU sends and receives datagrams thru the GBC using Comm_Request Perform Functions. *See the Series 90-30 Genius Bus Controller, User's Manual, GFK-1034 and the Genius I/O System and Communications User's Manual, -GFK-90486, for datagram details.*
The following program example demonstrates using the **Comm_Request** Perform Function to send a Read Device datagram. This datagram is requesting 64 words of register data starting at an offset of 201 from the start of this data block. The returned data is stored in 64 consecutive Integer Variables starting at variable number 296.

State Logic Program Example:

```
State: SetupReadDevice
      Status_Variable1 = 0
      Status_Variable2 = 0          ! Check status variables to see when Comm_Request completed

!!!!  The following variables from Cmd_Data_Len to RequestLength are defined as consecutive Integer Variables
       Cmd_Data_Len=13,
       Wait_Nowait=0,
       StatusMemoryType = 8,
       Status_Offset=11,
       Idle_Timeout=0,
       Max_Com_Time=0,

! ************** COMMAND BLOCK ***********************************
      Command_Number = 15
      Device_Number = 11
      Function_Code = 32
      Subfunction_Code = #1E    ! Read Device Datagram - Hexadecimal values specified with #
      Priority = 0
     Datagram_Length = 6     ! Bytes
     ReplyCode  = #1F       !Reply Code
     ReplyType = 8        !Reply stored in integer variable memory
     ReplyOffset = #127      ! Offset start at Integer Variable 296
     ReplyLength = 69       ! Words reserved for returned data
     RequestType = #0800      ! Integer type data requested
     RequestLocation = #C800     ! Send data starting from Integer 201
     RequestLength = #4000.      ! Send 64 words

     Go to SendReadDevice State.

  State: SendReadDevice

  Perform Comm_Request with
   Rack=0,
   Slot=9,
   Task_ID=1,
   Data_Block=Cmd_Data_Len.

  Go WaitOnResponse.                    ! Go to State which inspects status variables
```

Note

The Genius Bus Controller does not transmit or receive any data when the State Logic controller is in simulation mode.

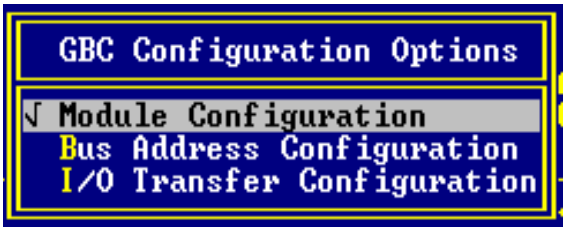
Note

The State Logic CPU uses only NOWAIT Communication Requests. The WAIT Communication Requests are not supported. Also the data type of the data sent or received must be register, number 08.

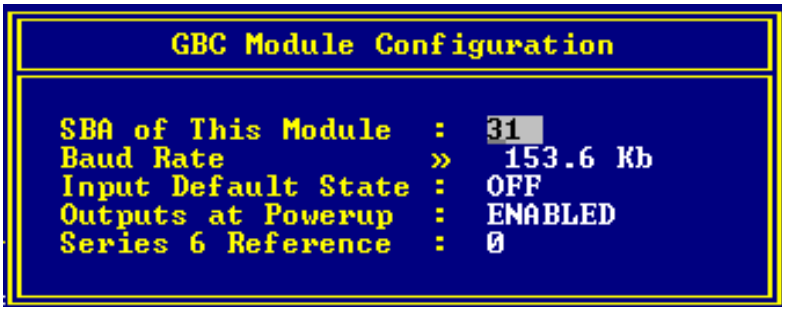
Configuration

To configure the module perform the following steps:

1. Select the GBC part number from the list of Option Modules.
2. The first form displays a blank to enter the starting reference of the %I status bits for this module.
3. The next screen provides three configuration options.

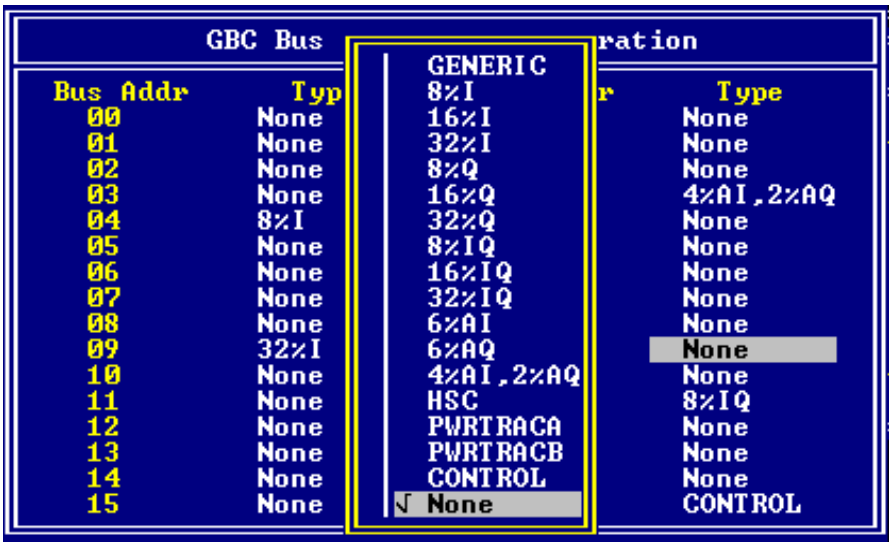


4. The “Module Configuration” option is a form for specifying configuration parameters for the GBC module itself.



Fill in these configuration options.

5. The “Bus Address Configuration” option displays a form to identify the devices on the Genius Bus connected to this GBC.



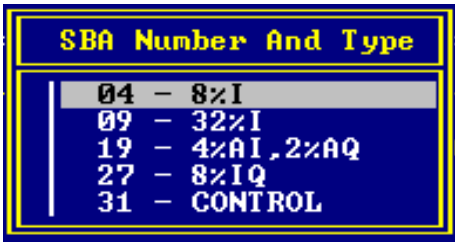
This form lists the devices configured at each of the 32 SBA addresses. To select a device type for an SBA, use the arrow keys to highlight the desired SBA, press any key to display the list of options. Highlight the correct option and press <Enter>.

The form displayed above shows the list of device types. Already configured in this screen is an 8 circuit discrete input block at SBA 4, a 32 circuit digital input block at SBA 9, an analog block with 4 inputs and 2 outputs at SBA 19, an 8 circuit digital input/output block at SBA 27, and the GBC itself at SBA 31, indicated as the CONTROL type. There can only be one CONTROL configured for a GBC.

To change the SBA of this GBC, use the Module Configuration form displayed above. The GBC SBA cannot be changed using the Bus Address Configuration form.

6. Select the "I/O Transfer" option to choose PLC reference numbers for the data associated with each SBA on the Genius Bus.

7. Select the device from the list to configure its starting reference numbers.



8. Enter the starting reference numbers the data types used with this device. The lowest available references are entered automatically into the form, but may be changed to any legal reference number. Even though the lowest values are entered automatically, this screen must be used and the values saved by pressing <F9> for the module to be properly configured.

```
I/O Configuration for SBA 19 - 4%AI,2%AQ

Input1    Ref: %AI1        Length:   4
Input2    Ref:    0        Length:   0
Output1   Ref: %AQ3        Length:   2
Output2   Ref:    0        Length:   0
```

9. The I/O Transfer screen for the GBC itself (CONTROL) is used to configure the system to broadcast global data on the Genius Bus. Press any key to display a list of data types. Fill in the blanks for the starting reference number and the length. This screen demonstrates configuring the GBC to broadcast as global data the data residing in the 16 integer variables starting at integer variable number 41.

```
I/O Configuration for SBA 31 - CONTROL

Input1  Type    %G    Start   0     Length  0
Input2  Type    %G    Start   0     Length  0
Output1 Type    %N    Start   41    Length  16
Output2 Type    %G    Start   1     Length  0
```

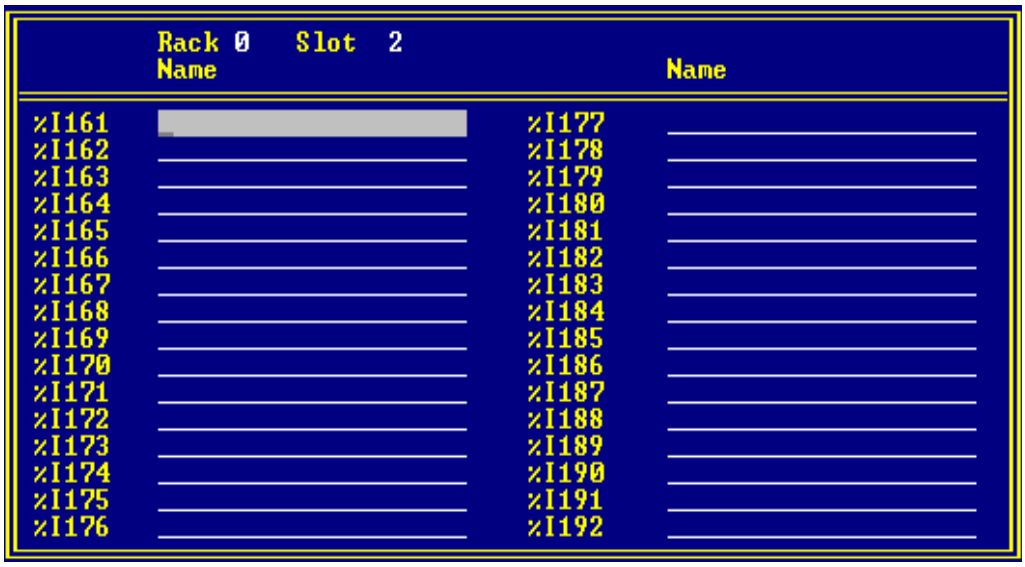
10. The I/O transfer screen for a device configured as Generic can be used to configure the system to receive global data from the device at that SBA. Fill in the input sections for the data type, starting reference number and the length of data for the memory locations that store the received data.

    The following screen demonstrates configuring the GBC to receive global data from SBA 23. The data received is stored in 8 floating point variable locations starting at floating point variable 123. The amount of data transferred for this configuration is 32 bytes, 4 bytes for each floating point variable.

```
I/O Configuration for SBA 23 - GENERIC

Input1  Type    %F    Start   123   Length  8
Input2  Type    %G    Start   1     Length  0
Output1 Type    %G    Start   1     Length  0
Output2 Type    %G    Start   1     Length  0
```

11. The “I/O Names” option on the SLOT OPTIONS menu activates a form for the names of the %I status bits for the GBC being configured.



Enter the names for the status bits in the appropriate blank or just use this screen to view the names entered elsewhere.

See the Series 90-30 Genius Bus Controller, User's Manual, GFK-1034 and the Genius I/O System and Communications User's Manual, GFK-90486, for more information about the GBC.

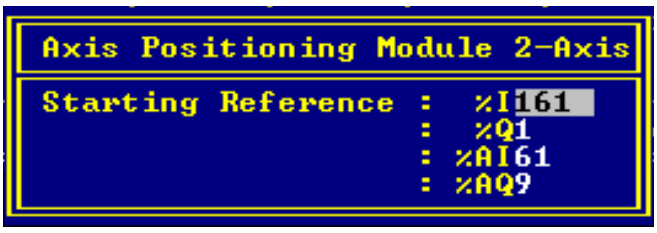
Axis Positioning Module (APM)

The Axis Positioning Modules are fully programmable motion control modules. There are two versions, the 1-axis (IC693APU301) and the 2-axis (IC693APU302). There are several manuals that explain the APM30 in detail, *GFK-0707, GFK-0664, and GFK-0781.*

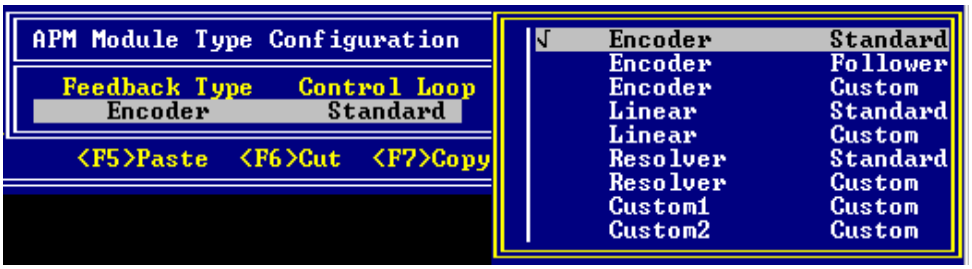
Configuration

To configure the Axis Positioning Modules follow these steps:

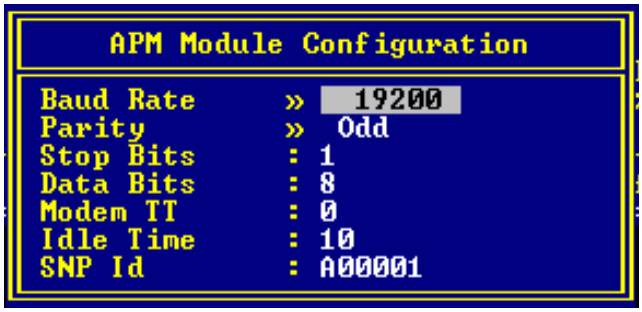
1. Highlight the appropriate slot.
2. Select the APM Module from the "Option Module" List
3. Fill in the Starting References form.



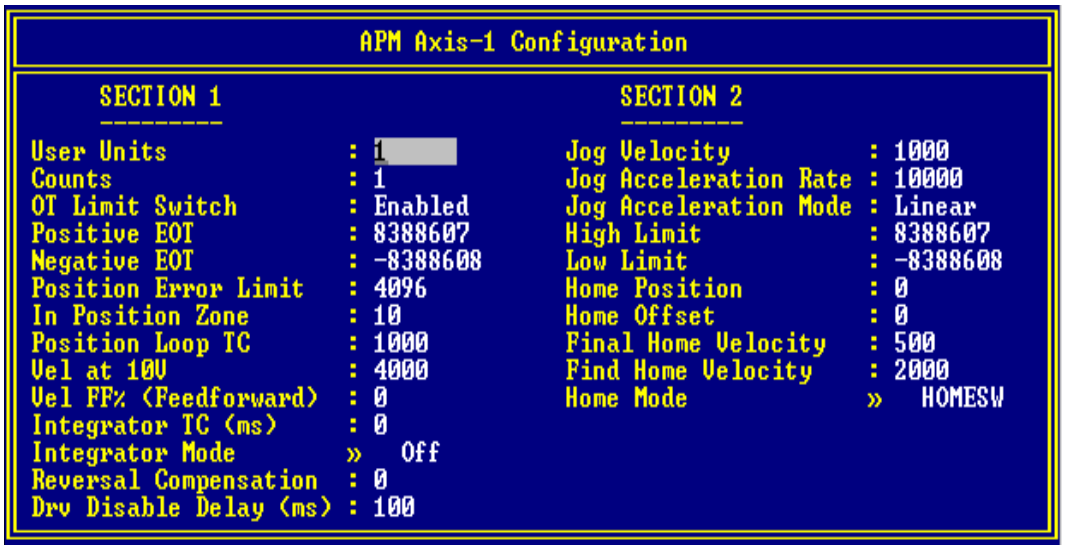
4. Select the "Module Type" from the List



5. Fill in the "Module Setup Data" form



6. Fill in the "Axis Setup Data" form



This screen varies depending on the type of module selected in the option described above. For a 2-axis module, press the <Page Down> key to access the screen for the second axis configuration screen.

7. Select the "I/O Names" option from the SLOT OPTIONS menu to view or add names for the APM I/O data. There are four separate forms, one for each of the data types used by the APM, %I, %Q, %AI, and %AQ.

Note

Some of the %AI and %AQ data transferred to and from the APM uses 2 words or 4 bytes of memory. The data that requires 2 words is represented as a

floating point number in the State Logic Control System. *See the APM manuals for information about which data is 2 word data.*

The IEEE 32 bit floating point data format used by the State Logic Control System represents accurate integer data up to 33,554,432. Values above this value start to lose accuracy, since there are some numbers in that range that can no longer be represented by this floating point format. Therefore the APM should be configured so that it does not require values in excess of 33,554,432.

Note

There is no current support for a program 0 in the ECLiPS programming software. Use the APM programming software to accomplish all programming functions.

Programmable Coprocessor Modules (PCM)

The PCM is a multipurpose module providing two serial ports, a MegaBasic interpreter, and drivers for CCM protocol support for the serial ports. All three 90-30 PCMs are supported by the State Logic Control System, IC693PCM300, IC693PCM301, and the IC693PCM311.

Accessing CPU Data

The PCM provides access to CPU data from the MegaBasic program or using the CCM serial protocol drivers. When accessing State Logic data from the PCM, use the SNP reference numbers used to access data from an SNP device. The SNP mapping of State Logic data is printed by ECLiPS, when the "SNP Protocol Listing" option is checked in the Print Project Data form, which is available from the PROJECT menu.

Analog data is available from either %AI and %AQ references or from %R register references. The %AI and %AQ references address the unscaled data only. Scaled analog data is a 4 byte floating point reference that is accessed using the %R register references. If an analog channel is unscaled, the %R reference stores the integer unscaled value in the low word of the two word reference.

Note

An analog output cannot be changed by using the %AQ reference. Use the %R register reference instead..

Caution

Forced analog input values are not available from the %AI locations. The %AI location returns the actual value from the module not the forced value.

See the chapter on serial communications in this manual for more information about the SNP protocol listing references.

Operational Notes

The PCM takes from 10 to 15 seconds to power up. Systems configured to run automatically on power up should delay interaction with the PCM, including Comm_Request functions, for at least 15 seconds after the program starts execution.

The PCM does not transmit or receive data when the State Logic controller is in simulation mode. The CPU does interact with the PCM when the program is halted and the CPU is not is simulation mode.

If the PCM is sent new configuration data, the PCM generates a reset event. This reset appears in the Fault Table and Event Queue. The reset is an alarm condition and therefore is a non-critical error. This event halts the program, if the CPU is configured to halt when a non-critical error occurs.

If a PCM fault is cleared from the Fault Table and the condition causing the fault is not corrected another fault is generated. The time from the first fault being cleared until the second fault is generated, is about one minute.

PCM Comm_Request

The CPU and the PCM communicate across the backplane. Either the PCM or the CPU can initiate communications. The Comm_Request perform function is used to send data to the PCM from the State Logic program.

The Comm_Request function sending data to the PCM executing a MegaBasic program does not return any fault information automatically. There is no status bit and the Comm_Request status word must be set by a MegaBasic program executing in the PCM.

The following program segment demonstrates using a **Comm_Request** Perform Function to send data to a PCM MegaBasic program. Every 2 seconds this program increments the variable PCM_Data_Block and sends its value to the PCM located in Rack 0 Slot 2.

State Logic Program Example:

```
State: SetUp  !Initializes Comm_Request Command Block data

!!!!  The names DataBlockLength through PCM_Data_Block are consecutive Integer Variables
      DataBlockLength = 2
      WaitNoWait = 0
      StatusMemType = 8    ! All State Logic Com_Requests use register data type only
      StatusMemOffset = 19    ! Zero based offset - Refers to Integer Variable 20
      IdleTimeOutValue = 1000
      MaxCommTime = 1000
      PCM_Data_Block = 1.

      Go to the SendCommReq State.

 State: SendCommReq    ! This State sends the Com_Request

   Perform Comm_Request with
    Rack=0,
    Slot=2,
    Task_ID=3,
    Data_Block=DataBlockLength.
```

Go to the Pause State.

State: Pause
If 2 seconds have passed , make PCM_Data_Block = PCM_Data_Block + 1 and
go to the SendCommReq State.

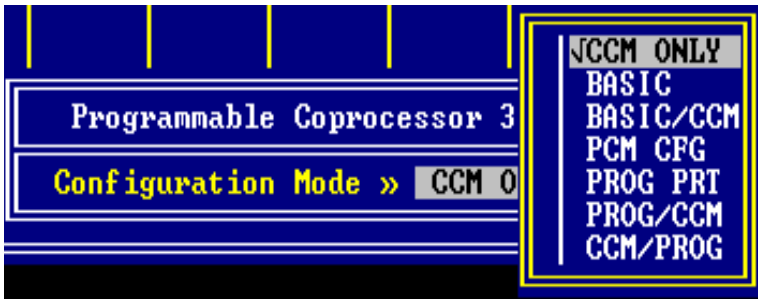
Note

The State Logic CPU supports only NOWAIT Comm_Requests. The WAIT option is not supported. Also, the only data that is accessed with a State Logic Comm_Request is the register data type #08.

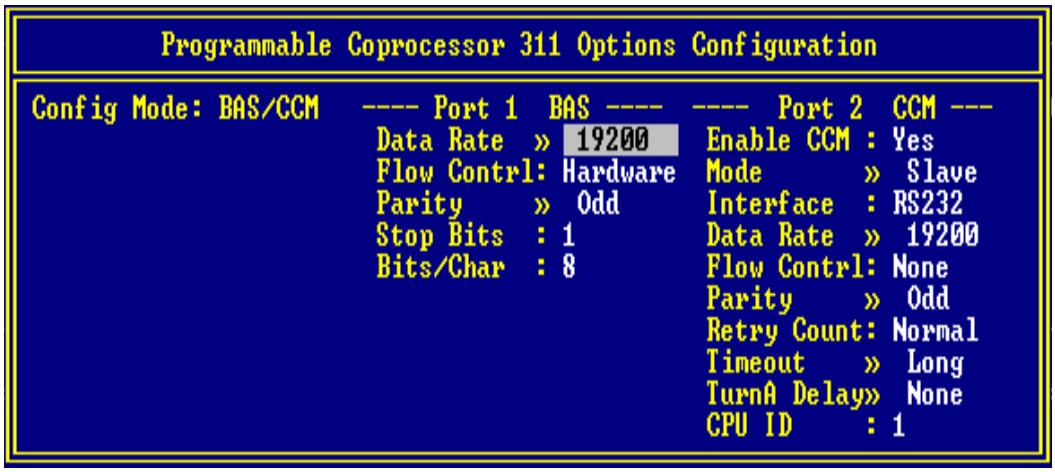
Configuration

To configure a PCM module follow these steps:

1. The PCM must be installed in rack 0, and is only supported by the 331 and 340 State Logic CPUs. Select a slot in rack 0.
2. Select the part number from the list of Option Modules.
3. Select the Configuration Mode from the first screen displayed. Press any key to display a list of the modes.



4. Fill in the form configuring each of the ports.



Serial Communications Module (SCM)

The Serial Communications Module (SCM) provides two additional serial ports for the 90-30 State Logic 331 and 340 CPU systems. One port supports RS-232 and the other can be configured for either RS-232 or RS-485/422 communications. These ports can be configured to be simple ASCII, CCM slave or SNP slave ports. At least one port must be an ASCII port.

All ASCII serial I/O is programmed in the State Logic control program running in the CPU. The State Logic program uses the READ and WRITE instructions to control serial data exchange. There is no separate programming of the SCM itself.

Use SCM version 3 or above when using a version 3 or above CPU. Older versions of the SCM are not compatible with version 3 or above CPUs.

For more information on the ASCII port programming, see the chapter on serial communications in this manual.

Note

The SCM does not transmit or receive data when the State Logic controller is in simulation mode.

Note

SCM takes from 10 to 15 seconds to power up. Systems configured to run automatically on power up should delay interaction with the SCM for at least 15 seconds after power is applied to the system.

If the system is powered up with the CPU low battery condition, the SCM does not receive valid configuration. The system must be power cycled again with a fully powered battery connected to the power supply, before the SCM operates properly.

Module Description

The Serial Communications Module requires no hardware configuration, i.e. no jumpers or dip switches, since all the ECLiPS software manages all of the configuration . The SCM provides one serial connector that supports the two serial ports and also comes with an OK LED, Reset switch, and WYE serial cable.

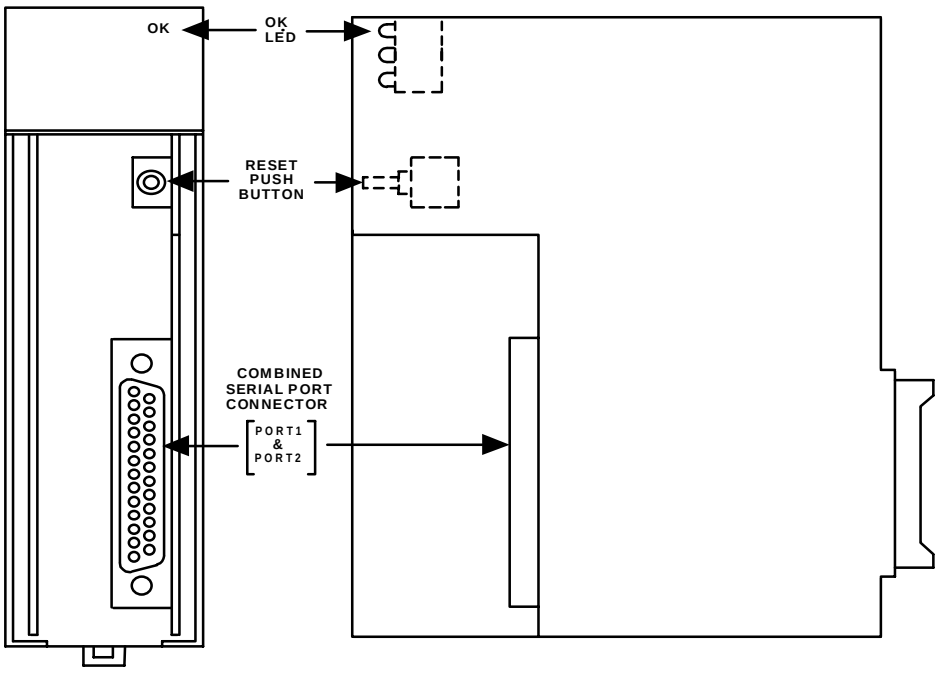


Figure 3-1. Serial Communications Module

OK LED

The SCM turns ON the OK LED after completing an internal self test procedure. The LED remains ON as long as the module is functioning properly. If the OK LED is OFF while power is applied to the system, turn off PLC power and make sure the module is seated properly in the backplane. Turn power back on. If the OK LED remains OFF when power comes back up, then there is a hardware failure in the SCM, and it should be returned for repairs.

There are two other LEDs on this module. These LEDs are not used at this time.

The SCM has a hardware watchdog timer that is periodically reset by the SCM firmware. If the watchdog timer expires, the SCM stops functioning and the LED turns OFF.

Reset Pushbutton

Pressing the Reset pushbutton when the OK LED is on, re-initializes the module. However, if the OK LED is off (indicating a hardware malfunction), pressing the Reset pushbutton will have no effect. See the previous heading, "OK LED."

Serial Connector

The SCM serial connector provides all of the connections for the two SCM serial ports. Separate pins on this connector are assigned for port 1 and port 2 RS-232 communications and for RS-485 communications on port 2.

The SCM ports always use hardware handshaking for flow control. If the connecting device does not support hardware handshaking, connect the CTS line to the RTS line on the SCM side of the serial cable connecting the two devices.

CAUTION

The serial port connector is connected to electrical ground within the PLC. Serial communications must be limited to distances of 50 feet (15 meters) unless ground isolation is provided by an external device. Failure to observe this caution may result in damage to the SCM or the communicating device.

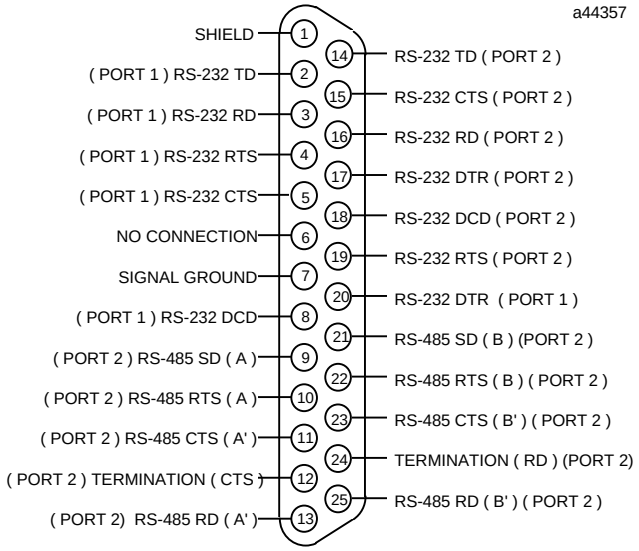


Figure 3-2. Serial Port Connector Pin Assignments

NOTE

If the connecting device does not support hardware handshaking flow control, tie the SCM CTS to the RTS line.

WYE Serial Cable

If only port 1 is used, standard cabling described in this chapter can be used to connect directly to the SCM serial connector. If port 2 is used, a specially built cable or the WYE cable, provided with the SCM, must be used.

The WYE cable is one foot in length and has a right angle connector on the end that connects to the SCM. The other end has a dual connector, one for port 1 the other for port 2.

The purpose of the WYE cable is to provide a connector for each of the two ports separating the signals from the single connector on the module. The WYE cable is designed so that the same standard cables used for port 1 can also be used for port 2.

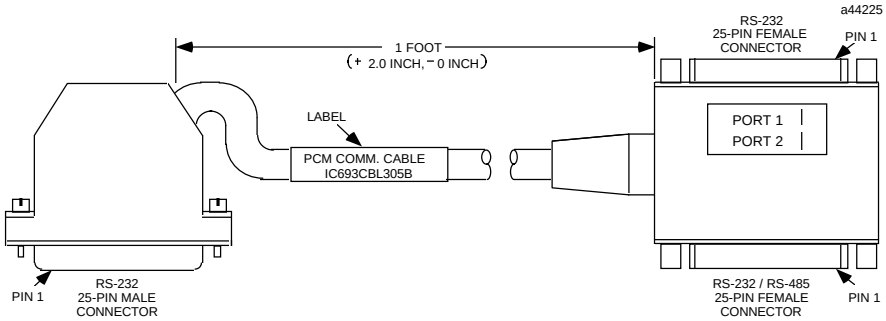


Figure 3-3. WYE Cable

WARNING

The WYE cable should NOT be used with an SCM connected to a multidrop network, because it introduces signal reflections on the cable. Multidrop networks should be cabled directly to the SCM serial connector.

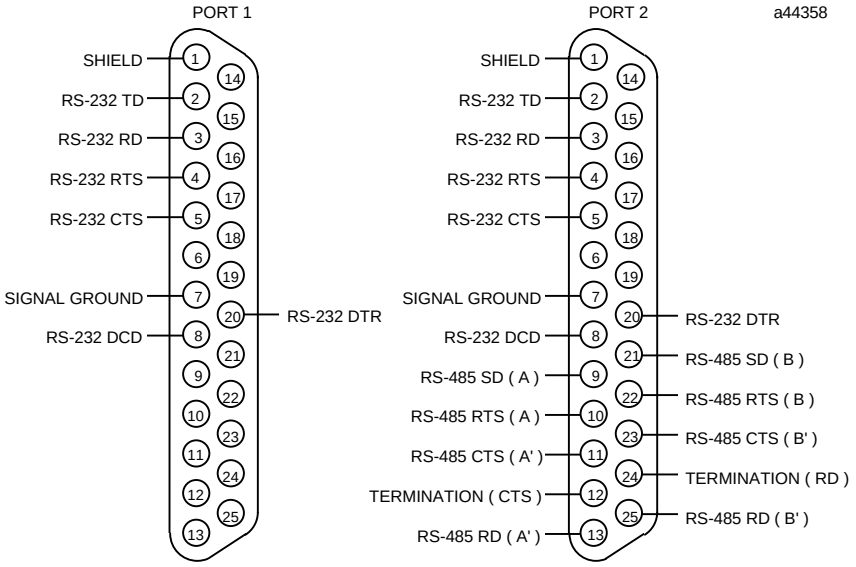


Figure 3-4. WYE Cable Pin Assignments

NOTE

If the connecting device does not support hardware handshaking flow control, tie the SCM CTS to the RTS line.

Cabling

- Cable Connector - Male, Subminiature-D Type, Cannon DB25P (Solder pin) with DB110963-3 Hood; AMP shell 207345-1 and connector 205208-1 with crimp pin, 66506-1, or solder pin, 66570-3; or equivalent standard RS 232C connector.
- Maximum Length -

  - 50 feet (15 meters) for RS-232C.

  - 50 feet (15 meters) for RS-485 without isolation at the remote end.

  - 4000 feet (1200 meters) for RS-485 with isolation at the remote end.
- Overall Shield
- 24 AWG (minimum)

The following cables provide acceptable operation at data rates up to 19.2K BPS and distances up to 4000 feet for RS-485.

Belden	9595	(5 pair, #24 AWG, stranded)
Belden	9184	(5 pair, #24 AWG, solid)
Belden	9302	(5 pair, #24 AWG, stranded)

For shorter distances up to 50 feet, almost any twisted pair or shielded twisted pair cable is adequate.

Caution

Do not use the shield as a signal ground conductor.

When using RS-485, the twisted pairs should be matched so that both transmit pairs make up a pair and both receive signals make up a pair. This procedure eliminates cross-talk problems that could affect the performance of the communications system.

When routing communication cables outdoors, transient suppression devices can be used to reduce the possibility of damage resulting from lightning or static discharge.

Caution

Care should be exercised to ensure that both the SCM and the connecting device are grounded to a common point. Failure to properly ground these devices could result in damage to the equipment.

The SCM is supplied with a 120 Ohm terminating resistor in each RS-485 receiver circuit. If the SCM is located at either end of a multidrop or point-to-point link, these resistors should be in the

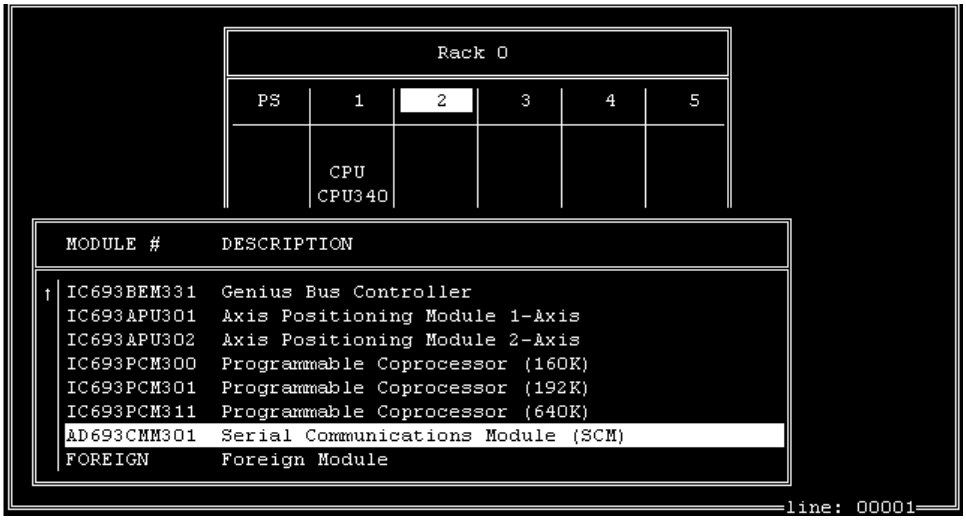
circuit. If the SCM is an intermediate drop on the multidrop link, disconnect the resistors from the circuit by removing their jumpers.

Configuring the SCM

There can only be one SCM in a system and it must be installed in rack 0 slot 2. The SCM also requires a 331 or 340 CPU controlling the system. The following configuration instructions were developed using Version 3.13 of the ECLiPS programming software package.

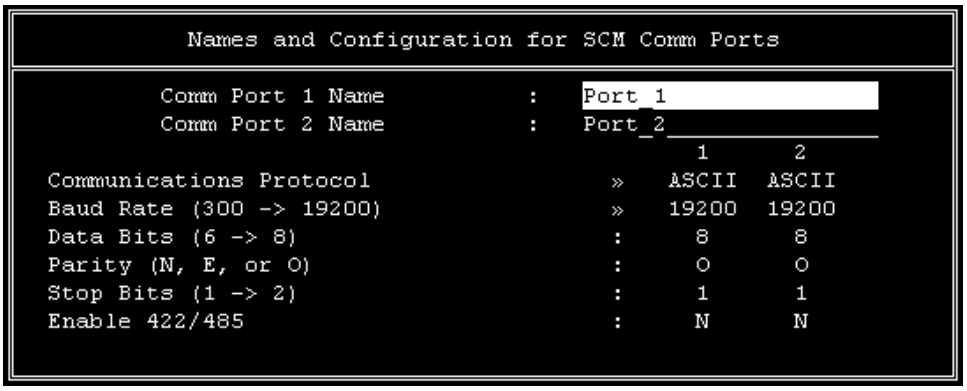
1. Start the ECLiPS program. After displaying its temporary startup screens for a few seconds, it will display its Main Menu.
2. Select (highlight by moving the cursor with the keyboard arrow keys) PROGRAM from the Main Menu and press the Enter key. You will now see the main project programming screen.
3. Press the F3 key. The Project Menu will display.
4. Select DEFINE and press the Enter key. The Selection Topic menu will display.
5. Highlight System Configuration and press the Enter key. The Rack 0 window will display.
6. If this is a new project, a default configuration will display showing a PS/CPU combination in the left-most slot of Rack 0. This is an unnumbered slot and is reserved for the power supply module. However, the default CPU311 CPU is an embedded design, which means that it is built into the rack's baseplate. That is why it is shown in the power supply slot. However, a 331 or 340 CPU module is required to support the SCM module, and it must be configured in slot 1 of the rack. So, the next step will be to change CPUs from the 311 to the 331 or 340 (we'll use the 340 for this example). If this is not a new project and you already have a 331 or 340 CPU configured, you can skip down to step 12.
7. Place the cursor on the slot that contains the CPU, and press the Enter key. The SLOT OPTIONS window will display.
8. Highlight View/Modify Configuration and press the Enter key. The State Logic Controller Configuration window will display.
9. Place the cursor on the Model Number field, and press the Enter key. A small window with CPU selections will display.
10. Select 340 and press the Enter key. The number 340 will now be shown in the Model Number field.
11. Press the F9 key to exit this window and save your changes, if any. If you made any changes, the Save Data Entered? window will display. If so, highlight Yes and press the Enter key. You should now see the CPU340 in slot 1 of the Rack 0 window. The SLOT OPTIONS window should still be displayed.
12. Press the right arrow key to move the cursor to slot 2 of Rack 0. The SCM module must be installed here.
13. If the SLOT OPTIONS window is not displayed, press the Enter key. In the SLOT OPTIONS window, highlight Configuration and press the Enter key.

14. Select Option Modules and press the Enter key. A list of option modules will display as shown in the following picture.



15. Use the down arrow key to select the AD693CMM301 Serial Communications Module (the module may not appear in the list until you scroll down with the arrow key), then press the Enter key. The SCM (CMM301) module should now be shown in slot 2 of Rack 0.

16. The SLOT OPTIONS window should be displayed. If not, press the Enter key. Highlight Configure Comm Ports and press the Enter key. The following screen will be displayed.



This screen allows the name of the ports as well as their communications settings to be changed.

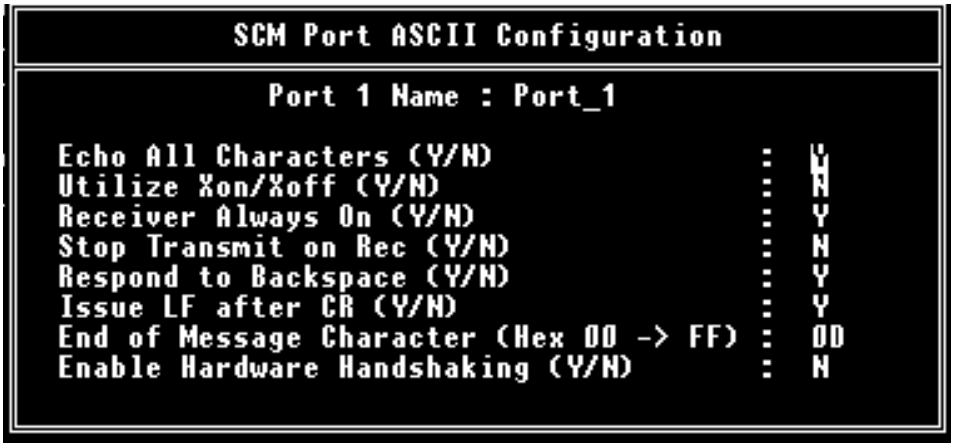
NOTE

In earlier versions (prior to version 3.13) of the ECLiPS software, one of the ports could be configured for either CCM or SNP protocol (at least one port had to be configured as an ASCII port). This was undesirable for certain applications because if power was cycled to the PLC, the protocol setting for a port configured for SNP or CCM would revert to its default configuration (ASCII) with a resulting loss of communications through that port. Therefore, it was decided to no longer support the configuration of SNP or CCM port protocols in the software. So starting with version 3.13, ECLiPS software does not allow the protocol setting to be changed from ASCII. If an SNP or CCM

port must be added to your system, it is recommended that a separate GE Fanuc PCM or CMM module be used for this purpose.

The editable parameters in this window (Baud Rate, Data Bits, Parity, Stop Bits, and Enable RS-485) are explained in the *Serial Port Parameters* table in Chapter 11 of this manual.

17. When finished setting the communications parameters, press the F9 key to exit this window and save your changes, if any. If you made any changes, the Save Data Entered? window will display. If so, highlight Yes, and press the Enter key. You will be returned to the SLOT OPTIONS menu.
18. Select Set Protocol Parameters and press the Enter key. A Protocol Port Select box will display.
19. Select the desired port and press the Enter key. The SCM Port ASCII Configuration window will display as shown in the following picture.



An explanation of the parameters on the SCM Port ASCII Configuration window can be found in the *Serial Port Parameters* table in Chapter 11 of this manual. One parameter needing further explanation is the Enable Hardware Handshaking option. When enabled, this option uses two lines of the port, CTS and RTS, to control the flow of data through the port. The CTS line is controlled by the SCM to limit the flow of data from another device. The RTS line is monitored by the SCM so that another device can control the flow of data from the SCM. If hardware handshaking is not used, tie the CTS line to the RTS line on the SCM end of the serial cable.

20. When finished configuring the settings on the SCM Port ASCII Configuration window, press the F9 key to exit this window and save your changes, if any. If you made changes, the Save Data Entered? window will display. Highlight Yes, and press the Enter key.

SCM Configuration Mechanics

The SCM configuration is downloaded to the PLC when the State Logic program is downloaded. The SCM is actually configured when the program starts running. To make changes to the configuration, repeat these steps and then download the program again.

The port parameters can also be changed by executing the State Logic program instruction, *Set_Commport*. (See the "Using the *Set_Commport* Command" heading in this document.) This function is also used to change the SCM port setups from the State Logic program.

Because the SCM is configured when the program starts running, any program changes to the SCM port configuration are canceled when the program is halted and then restarted.

Recommended Schemes

For maximum SCM operation reliability, the following schemes are recommended:

- Either use a true null modem cable connection for SCM communications ports (jumper pins 4 and 5; and jumper pins 6, 8, and 20 of each side of the dual connector on the WYE cable) or use a hardware handshaking scheme.
- Use the SCM ports as ASCII protocol only. If an extra SNP or CCM port is desired, purchase a PCM or CMM module. Starting with ECLiPS version 3.13, the ability to configure SCM ports to any protocol other than ASCII has been removed due to a reliability issue. This is discussed in more detail in the "Configuring the SCM" heading.
- Use the *Set_commport* command in your state logic programs. See the heading, "Using the *Set_commport* Command."

Using the *Set_commport* Command

The *Set_commport* command is used to specify SCM port settings within users' State Logic application programs. It may be added to user programs using the ECLiPS programming software. This command takes precedence over configuration settings. This means that when a program runs, the settings specified in the *Set_commport* command will be used instead of the configuration settings. This will help ensure the use of the desired SCM port settings even if the configuration settings revert to default values due to a power failure. The following procedure outlines one method of adding the *Set_commport* command to an application program:

1. Configure the port using the ECLiPS programming software. This configuration procedure is covered in an earlier section of this document. If your port is already configured with the desired settings, skip ahead to the next step.
2. Go to the Project screen and place the cursor at the place in the program (preferably near the beginning) where you wish to insert the *Set_commport* command.
3. Select F3 to display the menu.
4. Select List
5. Select Communications Ports
6. Select Comm Port Names
7. Select the desired port, then press the Enter key

8. Select Insert Reconfiguration Data for Port, then press the Enter key. This will place the *Set_commport* command containing the configured port settings into the current position in your State Logic program.

SCM Configuration Mechanics

The SCM configuration is downloaded to the PLC when the State Logic program is downloaded. The SCM is actually configured when the program starts running. To make changes to the configuration, repeat these steps and then download the program again.

The port parameters can also be changed by executing the State Logic program instruction, *Set_Commport*. See the explanation of the *Set_Commport* instruction in the *chapter on serial communications in this manual.*

Note

Because the SCM is configured when the program starts running, any changes to the SCM port configuration executed by program instructions are canceled when the program is halted and then restarted.

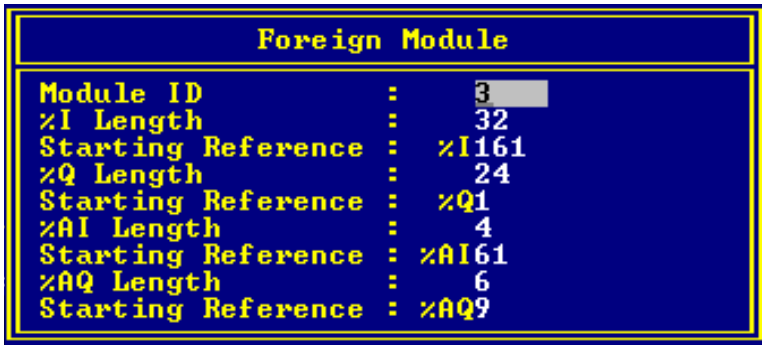
Foreign Modules

The Foreign Module option on the list of Option Modules is used to configure third party modules designed for the 90-30 PLC. These modules are not GE Fanuc products, but are designed to operate in the 90-30 State Logic Control System.

The slot can be configured for %I, %Q, %AI, and %AQ data. Other configuration options are a module ID, and 16 bytes of hexadecimal data.

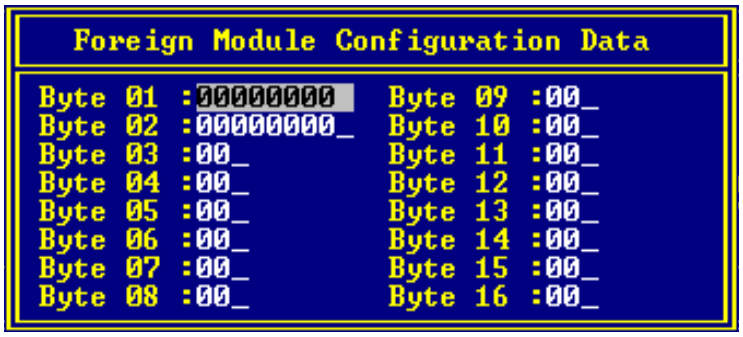
To configure a foreign module follow these steps:

1. Select “Foreign Module” item from the list of Option Modules.
2. Fill in the Starting Reference and ID Number Form. This form specifies the location, amount and type of CPU memory accessed by this module.



The Module ID is a 1, 2 or 3 digit signed integer value specified by the module vendor.

3. Fill in the Module Configuration Data Form.



The first two bytes are binary values (00000000 to 11111111). Bytes 3 through 16 are hexadecimal values (00 to FF).

4. The names for the I/O type used by the module can be entered or just viewed by selecting “I/O Names” option from the SLOT OPTIONS menu. There is a separate form for the names of each of the data types used.

Chapter 4 State Logic Control Theory

This chapter has two main topics. The first part discusses State Logic Control Theory and how it differs from traditional control models. The second part discusses the ECLiPS implementation of State Logic Control.

State Logic Control Theory

State Logic Control has its roots in Finite State Machine Theory, developed by nineteenth century mathematicians. Because its philosophy is a natural fit to real-time systems, Finite State Machines have become the strategy of choice in disciplines, such as electronics and compiler design. This theory has recently been gaining wide spread acceptance in industrial control industry with all major controls producers offering State Logic based control products.

The Concept of Finite States

The basic concept of State Logic is that a process can be defined as a sequence of States. Each State is defined by two components, actions that occur while that State is active and the **transitions** to other States.

In the control world, **actions** are turning ON digital outputs, setting variable and analog output values, sending messages to an operator, etc.

"Turn ON Mixer_Motor."

is an example of an ECLiPS program line describing the **action** of a State.

Transitions are a little more complicated since they are themselves defined by two components, the condition controlling the **transition** and the target State.

"If Part_In_Place switch is ON, go to Start_Conveyor State."

is an ECLiPS program line representing a **transition** of a State. In the control world conditions controlling **transitions** are the status of the digital inputs, the values of variables and analog inputs, elapsed time, etc. The target State is the one which becomes active when the condition is true.

A sequence of states can describe any control application. In pure Finite State Machine science these sequences are each called State Machines. ECLiPS calls each such sequence a TASK.

It is traditional to diagram Finite State Machines with circles and arrows. The **actions** of a State are written inside the circles. The arrows show the **transitions** with the condition component of the transition written next to the arrow. The following unlabeled State diagrams show two simple Finite State Machines or Tasks.

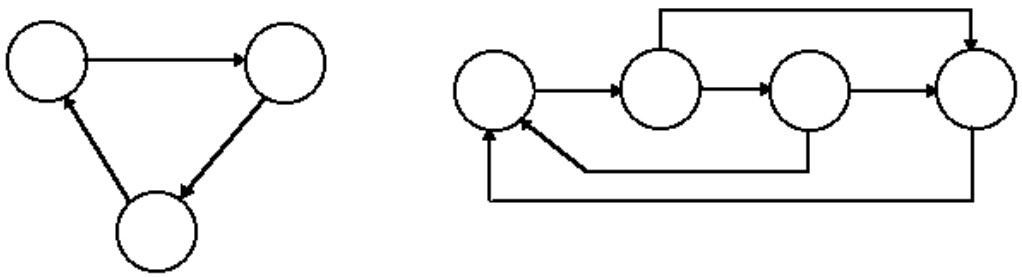


Figure 4- 1. State Diagrams

The Task may transition from one State to any other State in the Task depending on how the State instructions are specified by the system designer. The target State of all transitions is always pre-defined. A State description may describe several different State transitions based on differing input information. Each Task is always in one and only one State at any time, and the transfer from one State to another does not consume any time.

```
Project: CHEMICAL PROCESS
Task: Make_Compound_5

State: PowerUp
  If the Manual_Switch is on and Start_Pushbutton
  is pressed go to the Add_Water State.
  Go to Add_Water if Auto_Switch is on.

State: Add_Water
  Run Pump_1.
  If Tank_Gage equals 35 gallons,  then go to the Add_Chemicals State.

State: Add_Chemicals
  When the Chemical_Management Task is in the Mixing State,
  go to Pump_2_On State.  When Tank_Gage equals 39 gallons,
  send "Tank Filled" to operator_panel and go to the Mixing State.

State: Mixing
  If hour is past 8 AM, start the exhaust_system.
  Start Main_Mixer. If 20 seconds pass and the Mixer_Monitor
  is less_than 100 rpms, go to the Wait_3 State.
  Go to the Cooking State after 90 seconds.

State: Wait 3
  Write "PROCESS SHUT DOWN BECAUSE MATERIAL IS TOO THICK".
  Go to PowerUp State when Reset_Button is pushed.
```

Figure 4- 2. Task Description in ECLiPS

An **IMPORTANT** point is that Finite State Theory does not create or invent TASKS. TASKS are already an inherent part of every process to be controlled. Programming with a state control language is merely the act of describing the process.

What Makes State Control Logic Different

Both State Logic and traditional methods of control test the condition of the inputs and internal data to decide how to control a system. The fundamental difference of State Logic is its inherent ability to also use the current condition (State) of the process in making control decisions. Traditional methods of control artificially simulate different States with internal contacts or data values. Consider the following States:

```
State: Ready_For_Cutting
  Turn on the Cutter_Ready_Light.
  When the Cut_Push_Button is pressed, go to the Engage_Cutter State.
```

```
State: Engage_Cutter
Start the Cutter_Blade.
When the Cut_Complete_Detector is tripped, go to the Raise_Blade State.
```

These States describe a situation where the only time that the cutter should be activated from the push button, is when the machine is ready for the cutting operation. State Logic inherently allows the system designer to take the State of the process into the decision. By making the only transition to the Engage_Cutter State be in the Ready_For_Cutting State, the designer limits the time that the Cut_Push_Button has any affect on the Cutter_Blade.

In these States the only time the cutter is started is when the Engage_Cutter State is active. Traditional approaches allow for ingenious methods to simulate the States of the process to protect from an inadvertent pressing of the Cut_Push_Button at the wrong time. These traditional methods add considerably to the complexity of the system design, especially in intricate systems.

Because a State Machine model reflects sequence of operation over time, the model embedded in the controller matches the actual model the real world process follows. This model makes it possible to define the control system by describing the process.

Because the model matches the real world, program development and modification is always simpler and easier to understand. Program developers can more easily build advanced diagnostics for the process into the program because the control program is a precise model of the process and it is easy to detect when that normal behavior is not followed.

A Collection of Tasks is a State Logic Program

Finite state machines or tasks define sequential operations. Processes though usually have more than one sequence of operations executing concurrently. State programs are usually a collection of Tasks matching the actual real physical Tasks that are inherently part of the process under control. The State Logic control program is a collection of Tasks which execute concurrently.

Developing State Logic Programs with ECLiPS

Developing State Logic programs can be characterized as entering a description of your control system into a **template.** The template is the Finite State Machine model built into the State Logic CPU in the control hardware. ECLiPS is a tool and framework for entering that description into the template. ECLiPS will provide all of the commands and tools you will need to "load" virtually

any control application into the State Logic template. The primary element of the template's structure is the TASK. Each Task can be subdivided into an unlimited number of States. The I/O related activity and State change rules are described in each State with a collection of STATEMENTS. Statements are the ECLiPS command set you use to describe what you want to have happen at each State of each Task. In ECLiPS Statements are normal English words, phrases or sentences. An unlimited number of Statements can be used in any State.

Therefore, State Logic programs are a hierarchy of TASKS, subdivided by STATES, described by STATEMENTS.

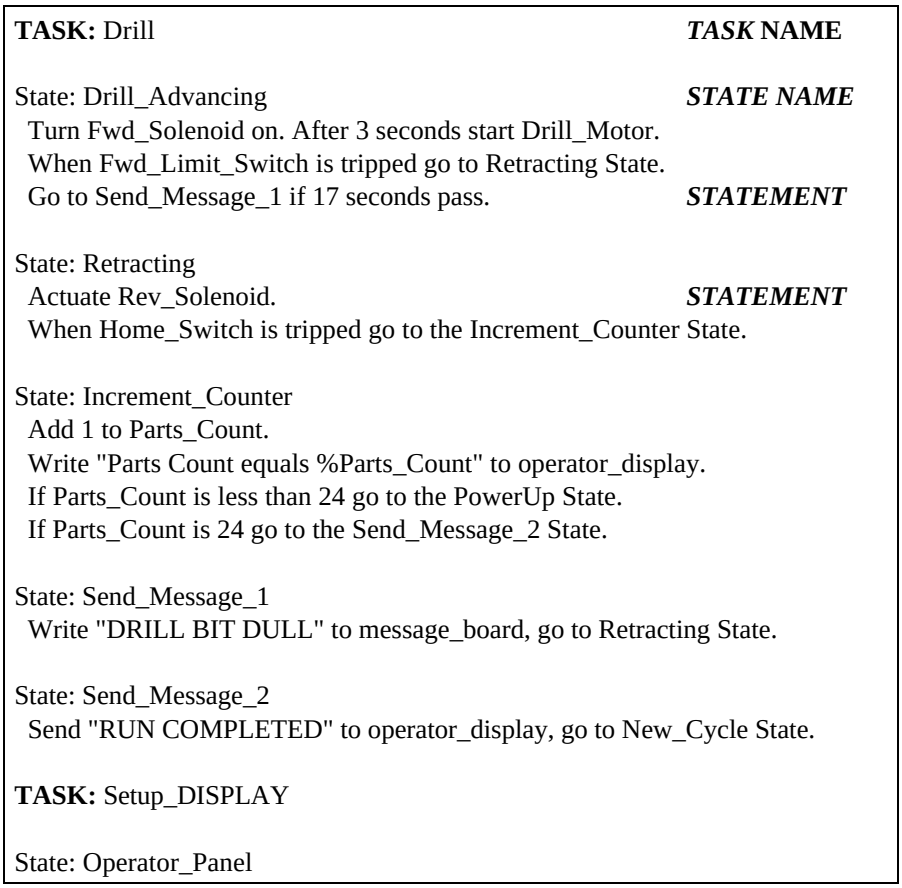


Figure 4- 3. Sample Task with Some Elements Labeled

Tasks - Sequences of States

By design a machine or process is a collection of Tasks that operate concurrently. A car engine has an electrical system, a fuel system, a mechanical motion system, cooling system, exhaust system and a starting system that, while independent in action, must be coordinated in time for the engine to work. Similarly all industrial processes, machines and systems will contain several Tasks that are mutually exclusive in activity yet joined in time.

While the Tasks are independent in action they are naturally related in time, since all Tasks come to life at power up and stop with shutdown. The control system designer can divide the overall process into individual Tasks to exactly mirror the system.

The types of Tasks that may be created are unlimited. Typical Task types include; motion control tasks, mode control tasks, filling tasks, measuring tasks, shutdown tasks, data recording tasks, diagnostic tasks, alarm tasks, operator interface tasks and so on.

States - The Building Blocks of a Task

```
Task: Mix_Station

State: PowerUp
If Can_At_Mix is on, write "Mixing Can" and go to Lower_Mixer.

State: Lower_Mixer
Run Mixer_Down_Motor.  When the Mixer_Down_Switch is tripped,
then go to Mix_Chemicals State.

State: Mix_Chemicals
Start the Mixer_Motor.
When 10 seconds have passed, go to Raise_Mixer.

State: Raise_Mixer
Run Mixer_Up_Motor until Mixer_Up_Switch is tripped,
then go to Mix_Complete.

State: Mix_Complete
When Can_At_Mix is off, go to PowerUp.
```

Figure 4- 4. Five State Task Example with a Single State Highlighted

In the automobile engine example we said an engine is viewed as a collection of Tasks; Fuel System Task, Electrical System Task, Starting System Task and so on. Each of those Tasks is further described as a precise set of States through which that Task will pass while the engine operates.

The automobile engine's Starting System Task has several possible States. For example we know there is a State in which the key is on, the engine is not running and the starter motor is not cranking the engine over. We know there must be another State in which the key is in another position, the engine is not yet running but the starter motor is cranking the engine over. There are also States in which the key is on, the engine is running and the starter motor is no longer cranking the engine.

Each Task is divided into States. The aggregate activity described by all of the States of a Task defines all the possible behavior of that Task under all conditions. A State defines the values for the outputs, sends messages, performs calculations, and assigns values to data variables. States also describe transitions to other States. Only one State is active and executed in a Task at any time. If two States need to be active at the same time then a concurrent Task is required.

Every Task must have at least one State. When the controller is powered up, the Task goes to this State, called the PowerUp State, which is the first State in the execution sequence of the Task. Thereafter, activity can move to any other State based on the Statements in the active State.

Statements - The Command Set for State Descriptions

State: Raise_Mixer
Write "Mixer Moving" to Operator_Panel.
Turn on the Mixer_Up_Motor. When the Mixer_Up_Switch is tripped, then go to Mix_Complete.

Figure 4- 5. ECLiPS State with One complete Statement Highlighted

In the automobile engine Starting System example, we would find that to make a complete description of the Starting System Task, activity would have to be described in greater detail. We could break down each State into a set of Statements that completely described the full and possible ranges of activity of that State.

Let's give the name "Starting" to the State in which we are actually trying to make the engine start up and run on its own. In the "Starting" State we could make a Statement like; "When the ignition key is in position three, go to the Crank_Starter_Motor State.", representing one of the Statements that form a part of the complete description of all possible actions of the Starting State in the Starting System Task. If the car was equipped with an automatic transmission the Statement might need to read; "If the transmission is in neutral or park and the ignition key is in position three then go to the Crank_Starter_Moter State.".

The actions of a State are described with a Statement or a collection of Statements. In ECLiPS a Statement is a collection of Terms describing the desired actions for that State. Statements end with a period (.) and can be thought of as sentences, although punctuation and proper grammar are not required.

There are two types of Terms used in a Statement;

- Functional Terms
- Conditional Terms

Functional Terms perform a specific action, including turning on digital outputs, setting analog outputs to values, performing calculations, setting variables to values, transferring to another State or communicating with other devices.

Conditional Terms perform some decision making test which enables or prevents execution of the functional Term in the Statement. The conditions that can be checked for include digital point status, analog values, a read from a serial port, or status of any system variable, including State activity from other Tasks.

Functional and Conditional Terms are listed below using typical ECLiPS terminology.

Functional Terms	Conditional Terms
Actuate, Start, Turn on	If
Go	when
Add, Subtract, Divide, etc.	Read
Make, Set	get
Write	
Start_PID, Stop_PID	
Suspend_Task, Resume_Task	
Perform	

Table 4- 1. Functional and Conditional Terms

Communications

State: Wait_For_Command
Read Start_Command from Operator_Panel, then go to the Start_Process State.
If 20 seconds pass go to the Operator_Prompt State.

State: Operator_Prompt
Write "PLEASE SELECT BATCH AND START PROCESS" to the Operator_Panel, then go to Wait_for_Command.

State: Problem_Report
Write "PROCESS SHUT DOWN BECAUSE MATERIAL IS TOO THICK".
Go to PowerUp State when Reset_Button is pushed.

Figure 4- 6. Highlighted Communication Functions

The Adatek controllers have two very powerful serial communication functions. These are a Read and a Write Term for the various serial ports.

The Write Term allows characters to be written to any of the serial ports in the controller. These can be connected to operator interface terminals or smart panels to present full screen displays or simple messages. These ports can also be connected to intelligent actuators or control devices, such as a robot controller, to provide set points and operating commands.

When a Read Term is encountered in the execution of a State, it is treated as a Conditional Term that isn't satisfied until characters are received from one of the serial ports. Once the complete message is received it places the characters in the designated variable for use by the rest of the program and then allows the active State to execute the next Statement.

The Read Term can be used to communicate to any serial input device. This would include operator interface devices such as terminals, smart panels, and personal computers. It would also include intelligent sensors such as weigh scales, and the various smart pressure and flow transmitters now sold by various manufactures.

Together the Read and Write Terms make communicating with the operator very powerful yet simple. It also makes it easy to communicate with intelligent sensors, controllers and other machines that populate the plant or factory.

State Logic controllers also provide access to internal data to remote devices through SNP and CCM protocols, using the Ethernet, PCM and CMM modules, and using Genius communications. The internal data is mapped to %R register locations for remote access. A mapping of all data is provided with the documentation features.

Scan Overview

The State Logic CPU which executes the control program continuously scans the inputs and the control program. Before each program scan, all of the inputs are scanned, so that each Statement of the program makes decisions based on the same input information.

During the scan of the program, the active State of each Task is scanned. Each Statement of a State is scanned in order from the first Statement to the last unless a GO Term is encountered. As soon as a Go is scanned, no more Terms in this State are scanned, and the scan moves to the active State of the next Task. Another State is scanned during the next scan sweep.

While scanning a Statement, the scan evaluates all conditional Terms before implementing the action described by the functional Terms. If any conditional Term is not satisfied or false, the scan of this Statement is stopped, the functional Terms are not implemented, and the scan resumes at the next Statement of the State.

The State Logic CPU keeps a table of all digital outputs and flags which are set ON during the program scan. Only the outputs set ON by one of the functional Terms in one of the active States during the scan are set ON, all others are OFF. The real world outputs and flags are set ON at the end of the scan. Therefore, an output does not go OFF during the transition from one active State to the next when that output is set ON in both States.

This scan discussion is a general overview of the program and I/O parts of the scan. The reference section of this manual has a more detailed discussion of the State Logic CPU scan procedure.

Interaction Between Tasks

Returning to the automobile engine State Logic model, we can see that we could describe the entire range of actions of an automobile engine as a collection of Tasks. Further, we can identify the different States through which each individual Task could pass during operation. Further, we should be able to see how we could use Statements to describe all of the actions possible for each State and the input conditions that would dictate which of the possibilities would actually happen.

But the engine wouldn't work unless we synchronized the timing of Task's execution with one another. The Starting System can work perfectly but if the Fuel System Task does not provide a squirt of gas into the cylinder during the time the Starting System Task is in the Starting State the

engine does not run. The same is true of the Electrical System Task, which needs to provide voltage to the spark plug at the right time in relation to the Fuel System Task, Mechanical System Task and Starting System Task if the engine is to start running.

All Tasks have two natural synchronization points, PowerUp and Shutdown. In between the Tasks will execute based on their own instructions and without regard to the other Tasks unless the program developer instructs the Task to do otherwise.

Joining the Tasks in time at various points in the operation of the process under control is not difficult. There are several techniques for accomplishing this coordination. A good working knowledge of how to implement Task interaction is important to the efficient development of State Logic control programs.

The following are a few examples of situations and techniques for controlling Task interaction.

1. Using a Variable to link Task Activities. One way to communicate between Tasks is by using a variable. Any Task has access to all variables even if the value for that variable is controlled by a different Task. An example of States in two Tasks using the same variable:

```
Task: Make_Parts
    State: Refill_Bin
        If the Parts_count is greater that 100 pieces go to the
        Refill_State. Otherwise go to the Grab_Base_Part State.
```

```
Task: Conveyor_Control
     State: Start_Station
        When Part_In_Place Switch is tripped, then Add 1 to Parts_Count
        and go to Start_Conveyor.
```

2. Changing the State of one Task from another Task. A common technique for implementing this type of Task interaction might be in connection with an Emergency Stop Button. Commonly, a designer may want every State of a Task or Tasks to take a specific action should an E-Stop Button be pushed. It would work perfectly well to specify the recognition of and reaction to the E-Stop button activation in every State, but this would be unnecessarily cumbersome.

A much more efficient method would be to create a separate E-Stop Task that forces a change of States in other Tasks when the E-Stop Button is activated. The State that the other Tasks would transition to by the by the E-Stop Task would contain a description of the desired response to the E-Stop button being pushed. This is how such a Task might look in ECLiPS.

```
Task: E-Stop

State: Emergency
        If the E-Stop-Button is pushed put the Generating Task into the
        Safe State and the Switching Task into the Controlled_Stop
        State then go to the Wait_for_Reset State.
```

3. Using the Current State of a Task State as an Conditional Term. The current active State Status of any Task is a variable in ECLiPS and can be treated as an input condition to make a Conditional Term within a State of any other Task. An ECLiPS example follows:

```
Task: Start_Motors

  State: Check_Conditions
       If the Conveyor Task is in the Running State then go to the
       Start_Main_Motors State. Otherwise go to the Send_Message State.
```

4. Using internal flags to signal another Task. Internal flags are set and tested just as are digital outputs. One or several Tasks may set a flag for another set of Tasks to test, for example:

```
Task: Smoke_Alarm_Monitor

   State: PowerUp
       Turn on the Smoke_Alarm_OK flag.
       If the Dock_Detector is on or the Boiler_Detector is on, or
       the Transfer_Detector is on go to the Alarm State.
```

The Smoke_Alarm_OK flag is true until one of the detectors is activated and Alarm becomes the active State. Any other State may test this flag to instantly see whether there is a smoke alarm activated.

Creating Process Diagnostics

One of the advantages of using the Adatek State Logic approach to control is the ease with which on-line process diagnostics can be added to the control program. Because the control program describes the process, any aberrations to the normal process can be detected and a response pre-programmed.

Creating Diagnostic Routines

The diagnostics can be added as Statements inside of States in Control Tasks, or whole new States within the Tasks, or as complete new Tasks.

If the desired response to an abnormal occurrence is simply a message or closing a digital output to turn on a light or sound an alarm, then that condition should be inserted as a Conditional Term in the appropriate Task.

```
If Tank_Pressure exceeds 90 psi, then write "OVER PRESSURE
CONDITION!" to the Operator and turn on the Alarm and go to the
Alarm_Light State.
```

If the occurrence of the condition needs to trigger a more elaborate response, or if it should alter the normal sequence of operation, then a Conditional Term should be inserted that is followed by a "go to" instruction that transfers the Task to a State where the diagnostic procedure takes over. If the occurrence can happen in multiple States, then a separate Task that checks for the occurrence and forces the control State into the diagnostic State may be the best way to perform the on line diagnostic.

Because the control program written in ECLiPS is self descriptive, and each State describes what should be happening and what should happen next, it is easy to insert diagnostics after the control program is finished.

In addition to the ability to add diagnostic logic with Statements, States and diagnostic Tasks, ECLiPS also contains several functions to help the user automate the process of adding them.

There are three primary techniques for adding diagnostic capability to a State Logic control program.

1. Each time a new State is added the user is given the opportunity to enter a maximum time for which that State can be active. The maximum time selected will be shown automatically in the program after the State name.

2. An "add diagnostics" choice is available from the menu. If selected the user will be given a choice of the type of diagnostic to add and a fill in the blanks screen. Once the screen is filled in, the Diagnostic State will be written automatically and inserted into the program.

3. Diagnostic logic can be created in the same fashion as control activities are accomplished, by using Task, States and Statements to describe the desired diagnostic activity.

```
Task: PINPOINT_FAILURE

State: Check_Pressures

     If main_tank_pressure is less than 100 psi write "Pressure too
     low, check tank door seal" to operator_panel and go to the
     PowerUp State. If main_tank pressure is more than 250 psi go
     to the Issue_Warning State.

State: Issue Warning

     Write "PRESSURE ABOVE NORMAL". Go to the Pinpoint_Problem
     State.

State: Pinpoint_Problem

     If Plant_Overview Task is in the Start-Up State and Pump_One
     is on, write "MAIN PUMP ON DURING STARTUP - SHUTDOWN WILL
BEGIN IN 30 SECONDS" to the Plant_Alert_Board and go to the
     ShutDown State. If the Normal_Run State of the
     Pressure_Control Task is active and the Relief_Valve is true
     write "Main Tank pressure relief valve is probably jammed" to
     Terminal_3.

State: ShutDown

     When 30 seconds have passed, go to the Begin_Shutdown State.
```

Figure 4- 7. Example Diagnostic Task

Chapter 5 Creating a State Logic Control Program

This section presents the fundamental concepts of how a control program can be built using ECLiPS. Every designer will develop his own style in using ECLiPS. ECLiPS is designed to support and even to encourage personal or corporate program development styles. Initially however, it is suggested that the following procedure be followed in creating your first control system program with ECLiPS. This procedure is split into two steps:

1. Outline the Application

2. Write the Program

Outline the Application

In this step the control problem is analyzed using a top down design strategy where the components of the main problem are identified at the top level and then each of these components is broken down into its separate parts. This decomposition of the problem continues until the application is completely described. The State Logic Control model invites top down design because of the hierarchy of its elements, Tasks, States, and Statements as described in the previous section. There are several different formats to aid in the top down design process including structure charts and structured flow charts, but we use a simple outline approach.

Identify the Tasks

The goal of this step is to identify the Tasks of the application. We start at the highest level, decomposing the problem into its general components. See the discussion on Tasks in the State Logic Control Theory section of this manual.

Think of the independent operations which must be accomplished to achieve goals of the application. The natural separations of activity often become Tasks.

The goal is to decompose the problem into parts that can be defined as sequences of I/O operation. Any cycles which repeat even with some variations are prime candidates to be Tasks. An important concept for identifying Tasks is that Tasks are a set of sequential operations. Events which occur in parallel or concurrently should be in separate Tasks.

These main sections of the outline should be general descriptive phrases such as:

Bore Cylinder
Load Boiler
Fill Vat
Retrieve Part

At this stage the goal is to just describe the application not force some solution. Some of the independent Tasks are quite obvious, others which require interaction with other Tasks are more difficult to identify at first. This is usually a repetitive process where original efforts must be adjusted as the outline progresses. As with most activities, proficiency increases with the number of efforts.

Identify The States

Once the Tasks are determined, then the States of each Task should be identified. The States describe the actual condition the outputs and responses to inputs at a certain point in the control process. The States form the control sequence and are really a picture of how this piece of the process (Task) should behave. See the discussion of States in the State Logic Control Theory section of this manual.

At this point in the design stage the goal is to determine that the correct action can be accomplished with the chosen Task architecture. Simply give each State a descriptive name fitting the major attribute of the activity that takes place when that State becomes active. Typical State names are:

```
Send Message
Add Water
Raise Drill
Start Motor
```

State names identify the general action of the State. The specific actions and the transitions are specified in the Statements.

Identify the Statements

This level is the most specific level of the outline. Statements specify detailed actions which are to occur while this State is active, and the transitions to other States. See the discussion of Statements in the State Logic Control section of this manual.

The actions specified by Statements include digital outputs that are to be ON, changes in analog output values, changes in variable values, and messages to be sent. Examples of actions as specified in Statements are:

```
Turn ON Pump 5.
  Start Mixer Motor.
  Write "Operation complete" to Operator.
  Add 1 to Parts Count.
  Turn ON Forward Solenoid.
```

Statements also specify the transitions of a State. Both the condition for transition and the target State are identified. The status of digital inputs, values of analog inputs and variables, and elapsed time are used to specify conditions for a transition. State names specify the target State that becomes active when the condition is true. Typical transitions as they would appear in an outline are:

```
If Forward Limit Switch is ON, go to the Drill State.
  If Vat Temperature is less than 45 degrees go to Raise Temperature State.
  When 10 Seconds have Passed, go to Raise Mixer State.
  If Part in Place Switch is ON and Manual Switch is OFF, go to Move State.
```

Statements are often complete English sentences, since very specific operations are specified at this level of the outline. In fact, feel free to specify Statements in any comfortable format. Some additional examples combine the State actions with the transitions:

```
Run Mixer Motor. When 5 seconds have elapsed, then go Raise Mixer State.
  Write "Drill Bit is Dull" to operator, then go to Retract Drill State.
  Read Command from Operator, then go to Report State.
```

Writing The Program

With this outline in place, the ECLiPS program is almost completely written. The finished program is very close to the outline.

There may be some changes to the outline because of some naming conventions for how Task, State, and some other names are entered into the program. ECLiPS can not provide for the full expressiveness of the English language so some of the sentence constructions may have to be changed, although many alternative structures and the ability to make custom changes to ECLiPS are provided. Also, the outline is in a general format with no specific reference to the actual I/O of the system so that the wording of the outline usually becomes more specific in the program.

To write the program the Tasks, States, and Statements of the outline are entered into the project using the ECLiPS editor which is active whenever ECLiPS is in Program Mode. Another part of creating the program is specifying I/O names and circuit configurations. Defining the I/O may be done before, after, or during the writing of the program.

Using English Names in the ECLiPS Program

When you start a new program, ECLiPS asks for the name of the first Task. After the name is entered, ECLiPS starts the program for you by writing the Task keyword followed by a colon and the Task name. ECLiPS also writes the first State name, "PowerUp" into the program. Tasks, States, I/O points and variables can all be assigned English names. Names can be as brief and code like or as descriptive as you wish.

Clever, descriptive names that fit well to the primary attribute of that State activity is strongly encouraged. This will pay dividends in future program modifying, clear documentation and easier troubleshooting.

Further, good descriptive names will enhance the quality of the automatic diagnostics that can be created by linking Task, State and I/O names together for automatic diagnostic output information.

Each name can have up to a twenty characters. These characters may be letters, numbers, or the underscore character (_). Names must begin with a letter. The name must be a continuous string of characters, i.e., no spaces are allowed.

Because ECLiPS uses the space character as a way to tell where one word ends and the next begins, as we normally do in written English, a name can not contain a space. To construct a multiple word name for descriptive purposes the designer should use the underscore character (_) to separate words or use uppercase to start every new word.

Table_Movement
TableMovement

Naming Tasks

Tasks can be given any name, although each must be unique. It is suggested that Task names be descriptive of the activity they represent. This descriptive use of names means clearer

documentation and the ability to create automatic diagnostic output messages by combining Task, State and I/O names to make complete messages.

A Task may be added to the program by using the "Add a New Task" option from the Add Menu or just typing in the Task keyword followed by a colon and the Task name. Each Task is assigned a name as it is built and each Task must have a unique name. This name appears at the beginning of every Task. Every time the designer wants to refer to the Task using ECLiPS such as in writing other Task sequences, during debugging or during diagnostics development, the English Task name should be used.

Naming States

Each Task contains one or more States. Similar to Tasks, a name is assigned to every State of the program either through the Add menu or directly into the program using the ECLiPS editor. Once assigned, these names are used when performing any functions associated with States while using ECLiPS.

Each Task always begins execution in the PowerUp State when the program starts. As a reminder as to which State will begin the sequence when power is applied to the controller, one State of every Task must be named PowerUp.

While every Task in a controller must have a unique name to differentiate it from the others, States in different Tasks may have the same name. All States within one common Task must have a unique name, but a State in one Task can be named the same as one in a different Task.

As with Tasks, names chosen for the States should be descriptive. By using combinations of words that describe something unique to the State, such as the action performed or function of the State, the program becomes self documenting. Using descriptive names makes it possible for people other than the original designer to use and modify a sequence at a later date with minimum learning time spent trying to understand the program.

Naming I/O Circuits, Variables, and Internal Flags

Each input and output from and to the field enters and leaves the controller through some particular hardware module. A name is given to each of these I/O points. All references to I/O circuits use the assigned name.

Names are also used for variables and internal flags. All names must be unique. A variable must not have the same name as a State or a flag must not use the name of a Task. The only exception to this rule is that States in different Tasks may use duplicate names.

The English name should be descriptive and can be made up of several words attached by the underscore character. ECLiPS allows the user to define I/O points or other name other elements of the program at any time during the programming process.

The "I/O Reference" form for naming analog I/O also has blanks for scaling that analog circuit. Analog values are scaled so that more meaningful engineering units are used in the program instead of raw data from the analog module.

For example an analog scale may use a 4 - 20ma signal which is converted to a 0 - 4095 digital value. See the discussion on analog scaling for more information on how to fill in these blanks.

Statement Structures

This section describes how to express the Statements in an ECLiPS program.

```
State: Drill_Advancing

Turn Fwd_Solenoid on.  After 3 seconds pass, start Drill_Motor.
When Fwd_Limit_Switch is tripped go to the Retracting State.
If 17 seconds pass, go to the Send_Message State.
```

Example ECLiPS State with Four Statements

Most States consist of several Statements that describe what action is to happen while the Task is in that State, what conditions will cause a transfer, and to what State the Task transfers to. With ECLiPS these Statements are written in descriptive English generally but not necessarily consistent with the rules of English grammar. Statements are short sentences or phrases that describe the desired actions in a way that anyone can read and understand. A Statement always ends with a period just as a sentence does in English.

Constructing Statements

There are two types of Terms in a Statement, functional indicating some action taken and conditional indicating some test for decision making.

After 3 seconds pass **start Drill_Motor.**
Turn Main_Heater on if Start_Switch is on.
Write "Parts Run Complete" to User_Panel.

Statement examples with Functional Terms highlighted.

```
After 3 seconds pass start Drill_Motor.
Turn Main_Heater on if Start_Switch is on.
Open Vent when temperature is greater than 100 degrees.
```

Statement example with Conditional Terms highlighted.

Functional Terms describe an action to perform when they are reached in the execution of a Task. **Conditional Terms** describe a condition that needs to be evaluated to decide whether the Functional Terms in the Statement should be executed at this time.

A Functional Term, such as "turn_on Motor_A" or "close the Red_Clamp", generally has a verb that describes the action such as, "turn_on", "close", plus a variable name or I/O name, such as "Motor_A", "Red_Clamp".

Terms are combined to form Statements. Most Statements will be a sentence or a phrase. A Statement may be entirely made up of a Functional Term such as "Turn_on the Automatic_Mode_Lite.". A Statement can also be a combination of Functional Terms such as "Turn_on the Automatic_Mode_Lite and the Main_Conveyor.". Often a Statement is a combination of a Conditional Term and a Functional Term such as "If Motor_A is on turn_on the Automatic_Mode_Lite and start Main_Conveyor.".

Every Statement must always have at least one Functional Term. A Statement can contain more than one Conditional Term, or a Conditional Term that is a combination of conditions, such as "If

Motor_A is on and the Red_Clamp is closed" or "If Motor_A and Main_Conveyor is on". There may be many Functional and Conditional Terms in a Statement.

Using Keywords, Synonyms and Filler Words

Keywords are the words in a Statement that ECLiPS recognizes as instructions to perform some function. Keyword can be words that cause an action, such as the word "actuate" when applied to a contact output. Or they can cause a conditional comparison such as the word "if", or be part of the comparison such as the symbol " > ".

ECLiPS comes with default keywords assigned. Some of these keywords also have synonyms defined. Using a synonym in the program is the same as using a keyword. A detailed list of all the keywords are given in the reference section and described in detail.

ECLiPS also comes with several filler words such as "the" or "a" defined. Filler words have no meaning to the control program. The sole purpose of filler words is to make program Statements more readable and understandable. The user can place filler words anywhere in the Statement. Commas and other punctuation may also be used for clarity without effecting program execution. The only punctuation which has meaning is the period (.) and the exclamation mark (!). The exclamation mark is used to document or add comments to the control program.

Keywords are the vocabulary of ECLiPS and together with filler words make it possible to easily write understandable descriptions that are the control program. This vocabulary may be changed to suit any desired convention. All of the keywords, synonyms, and filler words may be changed. ECLiPS can therefore be configured so that the program is written in a foreign language.

A menu based window function allows the user to make these assignments at any time during an ECLiPS programming session. In addition, another menu based window function allows the user to see a list of all synonyms previously assigned and to select one to enter into the program.

Use the flexibility to create a language that fits the terminology of the industry, or plant, etc. where ECLiPS is to be used. The written ECLiPS programs become even clearer to all involved with operating and maintaining the plant as they use the English names for the process points and the local terminology for the actions and descriptions.

Using Variables

Statements manipulate variables in addition to controlling the I/O. At times the application requires responses to values other than those represented by real field sensors. Examples of this might be the number of parts built during a shift, the flow through a pipe calculated based upon the pressure drop across the pipe, the style of part being built this production run, etc.

Items stored in variables are the results of calculations, totals that are being accumulated over time, something that must be remembered from one time period to the next, and constants that may be changed or tuned. Each variable is assigned a name during program development.

Once created each variable is available to be shared between Tasks and States within Tasks. One State may assign a value to the variable and another State (or the same State at a later time period) uses the value of the variable in making a decision. Once a variable is assigned a value, that variable maintains that value until a program Statement assigns a new value to that variable.

Variables may be configured to save their values when the program is halted or if there is a loss of power to the system. If not configured to save values, a variable is always set to zero when the program starts running. If the system loses power, the retentive data is maintained by the backup battery located on the power supply. When the CPU first powers up it checks to see if there has been any corruption of the retentive data. If some corruption exists, an error message is displayed, and all of the variables are initialized to zero.

When using ECLiPS to write programs or monitor running controllers, or generate diagnostics, the user needs only to refer to the variable by its English name. Remember that choosing descriptive names for variables helps to make the program self documenting.

The different variable types are listed below:

Integer Variables

This type of variable represents a whole number from +32767 to -32768. Integer variables have many uses including counts, menu choices, and item quantities.

Integer variables can also be used as logical variables, or variables that have only two possible values, either 1 or 0. Variables used in this manner can be thought of as true or false, on or off, etc. Using integer variables in this manner differs from using flags, since flags are ON only when a State turning them On is active. On the other hand a variable maintains its value independent of which State is active.

Internal Flag

Internal flags are variables that act like digital outputs, but do not produce any physical output from the controller. An internal flag can be set true by a State in one Task and then checked by a State in another Task. These flags can be used to coordinate the actions of different independent Tasks.

An internal flag is like a digital output in that if an active State is not setting it true the controller will automatically turn it OFF.

Floating Point Variable

This variable type is used to store numbers that are not whole numbers or numbers outside the range of integer variables. When variables are needed with a math function, generally it should be a real variable.

String Variable

This variable type stores a collection or "string" of characters. These characters can be any alpha numeric or control character represented by an ASCII code. This type of variable is a little more complicated and is used mainly in accepting inputs or creating outputs to serial communication devices.

Character Variable

Character variables store one single ASCII character. This type of variable is especially useful for operator interfaces when the operator must enter a single character.

Character Variables can not be used with either the ECLiPS or OnTOP programming software running. ECLiPS and OnTOP use the ECLiPS serial port to pass information between the State Logic CPU and the PC that is running ECLiPS or OnTOP. This data flow conflicts with the Character Read statement, causing the Read term to be satisfied by the incorrect data.

Character read statements can be used while working with the SCM module, or if some other device is communicating with the ECLiPS port.

Reserved System Variables

There are two categories of reserved system variables, the time variables and the fault variables.

The time variables are used to view or change the values of the system Clock/Calendar. The time variables are second, minute, hour, day, day of the week, and month. Use these variables to set the clock or to check the current time.

The fault variables are used to store any fault information about the system. Each slot is represented by a different fault variable. In addition there is a variable that indicates if there is any fault in the system and one for the Low Battery Fault.

These variables equal 0 when there is no fault. Faults may be cleared by setting the fault variable to 0 from the program or by using the Debug Mode CHANGE option or by some other device connected to the PLC using SNP or CCM protocol, or through Ethernet or Genius connnections.*See the appendix on the fault system for more information about the fault variables.*

Program Execution

The State Logic CPU operating system is a scanning system. The scan cycle starts at the start of the program, scanning the active State of every Task. During program execution there is always one and only one State active in each Task. The operating system completes a scan of the program hundreds of times every second.

During the scan of the active State of a Task, each Statement of the State is scanned in the order that it appears. Keep in mind that a Statement is a series of Terms terminated by a period (.).

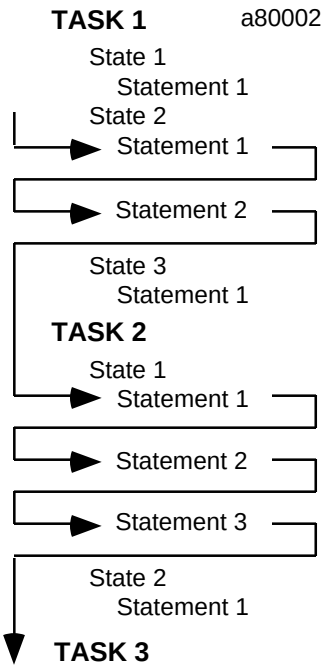


Figure 5- 1. Program Scan

The actions specified by Functional Terms are executed when the Functional Term is scanned. Each Statement must have at least one Functional Term, Conditional Terms are optional. If there are no Conditional Terms in a Statement, the Functional Terms are always executed during each scan. When Conditional Terms accompany Functional Terms in a Statement, the Functional Term is executed when all of the Conditional Terms are satisfied. There are four types of conditional Terms (see the reference section).

Conditional Terms are satisfied as follows:

1. **Read - When valid data is received at the appropriate channel.**
2. **If - When the conditional expression is TRUE.**

To understand how Statements are scanned, assume that the Statement Conditional Terms precede the Functional Terms and that the scan proceeds from left to right.

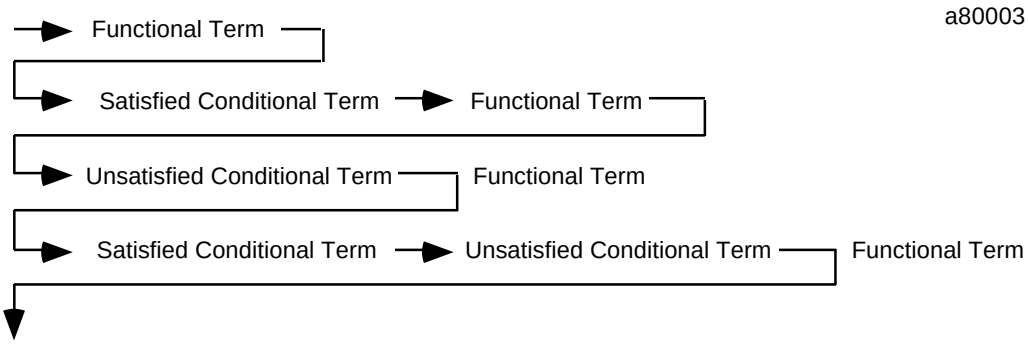


Figure 5- 2. Statement Scan

The Statements of a State are executed in the order that they are written into the program. Functional Terms of Statements with no Conditional Terms are always executed. Conditional Terms in Statements control whether or not the Functional Terms in those Statements are executed. If all of the Conditional Terms are satisfied, the Functional Terms are executed. If any of the Conditional Terms are not satisfied and Conditional Terms, the Functional Terms are not executed. For simplicity this rule assumes that the Conditional Terms are ANDed together. See the reference section about combining Conditional Terms using the AND and OR logical keywords.

The Statements are executed one at a time. In this manner every Statement of the active State is evaluated.

There are two types of Functional Terms that can prevent the execution of the rest of the Statements in a State. One is the Halt command which stops program execution. The other is the "go to ..." command, which immediately causes another State to become the active State. No Terms in a State are scanned after a GO is scanned in that State.

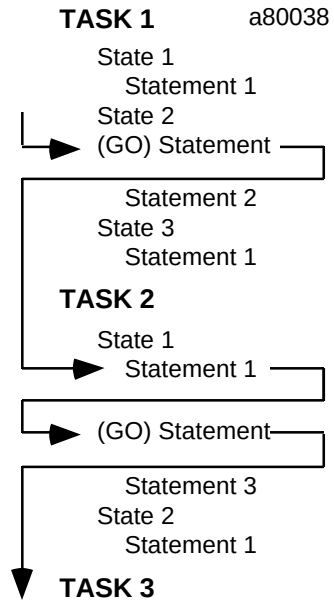


Figure 5- 3. Program Scan with GO Terms

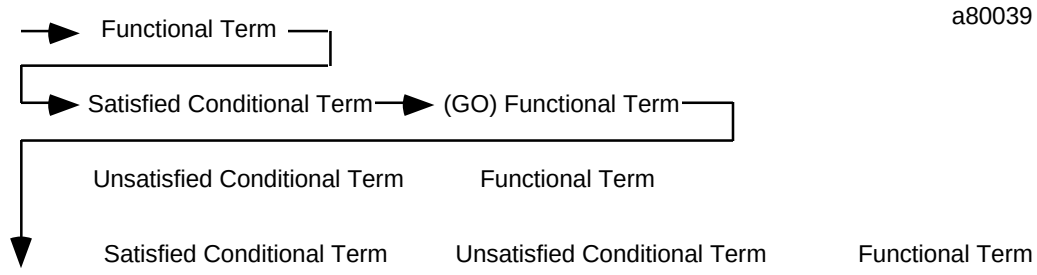


Figure 5- 4. Statement Scan with GO

```
If Start_Pushbutton is pushed, go to Start_Up State.
Write "Press Start Push Button Now!!".
If 30 seconds have passed, go to the Restart_Buzzer State.
```

This series of Statements causes the Start_Up State to become the active State when the input represented by Start_Pushbutton name is true. When GO Term is scanned, all Terms or Statements following this Term are not executed and at the next controller cycle, the scan of this Task starts at the first Statement of the Start_Up State.

During the program scan any changes to variables are made immediately. Therefore, a variable change in one Task is visible by the rest of the program during the same scan. On the other hand, digital I/O and Flag and analog values are made at the end of the scan. Therefore, if one Task makes a change to the condition of a digital output or Flag, the condition cannot be tested by another Task until the next scan through the program.

Hints for Creating ECLiPS Programs

Outputs are OFF by Default

One of the key features of the State Logic model is that the discrete outputs are OFF by default (their normal condition). Outputs are only ON if they are being turned ON by a program Statement in an active State.

This arrangement enables the State Logic model to have a one-to-one correspondence between the control program and the real world. States in the program exactly reflect the States of the machine being controlled. There are other direct benefits of the outputs being OFF by default.

While writing the State Logic program, the programmer need not be concerned with turning any outputs OFF. Having to be concerned with turning outputs OFF adds much complexity not to mention a great deal of logic to a control program.

Another feature of this arrangement appears when troubleshooting or debugging an executing program. To see which outputs are ON at any point in time, one only need check the State definitions of the active States of each Task. Any outputs set ON in these States are ON and all others are OFF.

Outputs are turned on by using the keyword Start (or any of the synonyms such as Turn_On, Energize, Actuate, etc.). Outputs are turned off when another State that does NOT turn on the output becomes active. An example is listed below:

```
Task:  Drill_Press_1
     State:  PowerUp
          If Clamp1 and part_in_place are true, go to Advancing.

     State:  Advancing
          Turn_On Forward_Motor.  When Drill1_Forward_LS is true then
          Go to Retracting.

     State:  Retracting
          Turn_On Reverse_Motor. When Drill1_Home_LS is true then
          Go to the Counter State.
```

In the Advancing State the Forward_Motor is turned on. When the Drill1_Forward_LS digital input is true the machine will go to the retracting State. The Forward_Motor output will then be turned off because it was NOT turned on during this State.

The operating system assumes that if an output is not actuated during an active State that the output is OFF. When actuating an output remember to continue to actuate (or turn_on or energize) that output in all successive States that also require that output to be ON.

NOTE: An output stays ON during the time when the operating system goes from one State definition to another, since State transitions do not take any time.

The following example demonstrates how to keep an output ON for successive State definitions:

```
Task:  Fill
          State:  PowerUp
                    Energize Fill_A.
                    Write "Filling tank with liquid A"
                    Go to the Weight State.

          State:  Weight
                    Energize Fill_A.
                    If Weight_Input > 20 lbs,
                    Go to  Fill_B State.
```

If there are several States where the same outputs are turned ON again and again, then probably the program should be rewritten and another Task added to control the outputs that are ON in several successive States. In the following example the output Run_Light would be on at all times unless the E_Stop button had been pressed.

```
Task: Output_Always_On
        State: PowerUp
                if Emergency_Stop is on go to E_Stop.
                Turn on Run_Light.
        State: E_Stop
                If Reset_Buttton is pressed go PowerUp.
                Put Other_Operations Task Into Emergency State.

Task: Other_Operations
        State: PowerUp.
                Turn on First_Operation.
                if  2 seconds have passed go Next_Step.
        State: Next_Step
                Turn on Second_Operation.
                if 2 seconds have passed go PowerUp.
        State: Emergency
```

Using multiple to Tasks to keep an Output ON.

Task Design

Tasks should be designed to control operations that are executed in parallel or at the same time as other operations. This might be a motion operation that happens in parallel with operator interface activity.

An emergency stop push button is an example of a function that should be placed in its own Task. The Task can be called ESTOP for example and its only function is to monitor the emergency stop button and coordinate the activity of the other Tasks.

```
Task:  ESTOP
        State:  PowerUp
                Let Reset = 0.
                If the emergency_stop button is pressed,
                Go to the Shut_down State.

        State:  Shut_down
                Make Fill_Can Task = Shut_Down State.
                Make Conveyor Task = Shut_Down State.
                Make Temp_Control Task = Shut_Down State.
                Go to the Wait_Reset State.

        State:  Wait_Reset
                If the Reset button is pressed Let Reset = 1.
                Go to the PowerUp State.
```

This example uses one Task to determine if an emergency stop button has been pressed and then forces the other Tasks to go to their own shut down States. Notice that States in different Tasks can have the same name. The integer variable Reset is used to communicate to the other Tasks when the Reset button has been pressed.

- An indication that another Task should be created, is that a program segment requires an output to be turned ON in several States or an input is repeatedly monitored in several States.

This type of structure indicates that more than one activity is being controlled by one Task, and a new Task should be used so that Tasks are used to control only one activity at a time.

- When Tasks are used for more than one activity, the complexity of the programming increases. Therefore, inordinate program complexity is another clue that there are too many activities being controlled by one Task.

Write Term Considerations

The write term is interpreted in a somewhat different manner from the rest of the terms. Most terms are executed every time they are scanned. Some terms may not be scanned if preceding conditional terms are not true, but the write term executes only once for the entire time that the state remains active. As soon as there is a state change, the write term again will write its message when scanned.

```
State: OverTempAlarm
  Write "Over Temperature Alarm! Press reset switch" to OperatorStation.
  If ResetSwitch is pressed, go to PowerUp State.
  Wait 10 seconds, then go to OverTempAlarm State.
```

The previous program segment writes the alarm message once every ten seconds, until the reset switch is pressed.

There is a limit of 32 write terms per state. If more than 32 write terms are possible, only the first 32 will execute. All additional write terms, 33 and up will not execute.

Calculations and a Scanning Operating System

Because the State Logic CPU operating system is a scanning system, each Statement of the active State of each Task is scanned many times every second. Because of this scanning design, care must be taken when using mathematical operations.

The make structure is executed each time it is scanned. If there is a mathematical operation in the Make, that operation is performed each scan. This may result in unexpected values generated by the Make.

```
State:  Check_For_Part
  Make Parts_Count = Parts_Count + 1.
  If Photo_sensor is ON, then go to the PowerUp State.
```

The State above will cause the variable Parts_Count to be incremented every scan that the State is active and the Photo_Sensor is not ON. Instead of incrementing the count by one, it may be incremented by several hundred.

The correct way to construct this counter is to put it after the conditional so that it is executed only once before the next State becomes active.

```
State: Check_For_Part
  If Photo_Sensor is ON,   then Make Parts_Count = Parts_Count + 1 and go to
PowerUp State.
```

Read Term Considerations

A Read Term is the READ keyword followed by a variable. This Term causes the processor to Read an input from the keyboard or from a SCM port and store the input into the variable that follows READ. The Read Term is a conditional Term and must be followed by a GO Term. The Read is satisfied when valid input to the variable is completed.

Input to the variable is completed when valid data for the variable is received at the specified communications port. Any of the variables may be used with the Read Term, and the type of data received must match the variable type to be valid. If the data type does not match the variable type, the data received is ignored and the Read Term continues to wait for valid input.

If a character variable is used with READ, the first character received at the port is stored in the variable and the READ conditional is immediately satisfied causing the following GO Term to be executed. All other variable types used with READ, wait for an End Of Message Character to be received before the input is satisfied. The default End Of Message Character is a Carriage Return, so that normally the READ is satisfied when the <Enter> key is pressed. The End OF Message Character can be changed by using the Set_CommPort keyword.

The 90-30 State Logic CPU does not allow the use of READ statements with character variables in the the ECLiPS or the OnTOP programming software. Character variable read statements can be used from other devices that are using the ECLiPS port or the Serial Communications Module. If ECLiPS is used with a Character variable read statement, the read will be satisfied by communication that is occurring between the State Logic CPU and ECLiPS. The information that satisfies the read will be incorrect.

It is possible to have two Read Terms for the same port to be active at the same time. This arrangement can only happen if the Read terms are in separate Tasks. When two Read terms are active at the same time, it cannot be predicted which one may receive the next input from the port.

It is a good practice to place all read terms for the same port in the same Task, so that only one Read is active at a time.

Timer Considerations

All timers in ECLiPS monitor the time that the current State has been active. This method of establishing a timer applies to the IF, FOR, and WAIT conditional terms.

Another way to think of the Wait Statement is as a conditional term that says "On the condition that this State has been active for __ seconds ...". If the State has been active for the amount of time specified the condition will be seen as a satisfied condition. An example follows:

```
Task:  Cook
        State:  PowerUp
                  Go to the Start_Cooking State.
        State:  Start_Cooking
                  Actuate Heater.
                  Wait 10 seconds, go to the mixing State.
```

A logical ERROR in using a timer is demonstrated below.

```
Task:  Drill
    State:  PowerUp
        Go to the Punch_Down State.

    State:  Punch_Down
        Energize Punch_Down_Output.
        When Punch_Down_LS is true and if 3 seconds have passed,
        Go to the Punch_Up State.
```

If it takes 3 or more seconds between the time the Punch_Down_Output is energized and when the punch down limit switch is met, the Wait timer is immediately satisfied. If the desired action is to wait 3 seconds after the Punch_Down_LS is true, then another State can be added as follows:

```
Task:  Drill
    State:  PowerUp
        Go to the Punch_Down State.

    State:  Punch_Down
        Energize Punch_Down_Output.
        When Punch_Down_LS is true  go to the Punch_Wait State.

    State:  Punch_Wait
        Energize Punch_Down_Output.
        Wait 3 seconds, go to the Punch_Up State.
```

Documentation Hints

This section offers advice on how to more effectively use the ECLiPS documentation features.

Descriptive Names

When naming Tasks, States, I/O and other data types it is extremely useful to use descriptive names. This will prove to be very helpful when debugging the program and when troubleshooting the production machine. The following program provides an example of descriptive name use.

```
Task:  Fill_Station
     State:  PowerUp
          If Can_At_Fill is on, go to the Pour_Chem_1 State.

     State:  Pour_Chem_1
          Turn on Chem_Valve_1.
          When Fill_Weight_Input is above 20, go to Pour_Chem_2.

     State:  Pour_Chem_2
          Open Chem_Valve_2 until Fill_Weight_Input is above 30 lbs,
          then go to Wait_for_Can_Removal.

     State:  Wait_for_Can_Removal
          When Can_At_Fill is OFF, go to PowerUp.
```

This program is easy to understand because of the names chosen for the Task, State and I/O channel names selected.

Underscores

The use of underscores is also helpful when naming Tasks, States, and I/O. The previous example also demonstrates the use of the underscore to help clarify the name of an I/O point. The underscore is necessary for the compiler's interpretation of the English text. A word is always considered complete when a space is found after text. Therefore to separate letters in one word (like an I/O point name) the underscore acts as a space without causing the compiler to see the descriptive name as two individual words.

Another popular way to separate words is to capitalize each word but use no spaces.

Power_Up	or	PowerUp
Can _At_Fill	or	CanAtFill
Fill_Weight_Input	or	FillWeightInput
Punch_Down	or	PunchDown

In either case it is best to be consistent throughout the project.

Programming Conventions

It is also suggested that the company using ECLiPS decide on a few programming conventions to increase the standardized look of all of the programs created in one company. Some suggested conventions include:

1) The default keywords should be selected and used throughout the program.

2) Task and State names are typed in all upper case.

3) Standardize abbreviations such as LS for limit switch, LT for light, etc.

Comments

Comments should be used in the program to clarify confusing or complex logic. To insert a comment in the program simple type an exclamation mark (!) followed by the comment. ECLiPS ignores any text following the exclamation mark on that line. Comments may be inserted after any program lines or take up an entire program line.

Scan Time

This section presents a few ways to optimize program scan time. It is easy to forget that this system is a scanning system and that the current State of each Task is scanned repeatedly. A common mistake is to remain is a State that continuously performs some redundant operations such as initializing a variable value. Such operations need only be performed once then proceed to another State. Remember each time the operation is performed, scan time is used.

Another technique that can save scan time is to locate variables and I/O points at the lowest number location available. The State Logic CPU accesses all memory locations up to the largest defined for that type in the program. For example, if the largest analog point is defined as %AI 900, all of the data from the start of the AIs up to 900 is read by the State Logic CPU. If there were only one AI in this system, designating it as %AI 900 waists the scan time necessary to read in all of the unused AIs.

Another scan time saving technique is to split up large States which repeatedly check many items. If the items do not need to be checked every scan, splitting a large number of checks into several successive scans can reduce the scan time.

Chapter 6 ECLiPS Programming Tools

This chapter describes the tools for creating and modifying State Logic programs. It also discusses the features used to translate and download a State Logic program to the State Logic CPU.

This chapter highlights the following features:

- **State Logic Program Editor**
- **Project Management Tools**
- **Variable Name Definition Procedures**
- **On - Line Program Changes**

State Logic Program Editor

State Logic programs are created using ECLiPS Program Mode. The State Logic program is written in English using the State Logic Program Editor.

ECLiPS Editor Screen

ECLiPS always displays the project name and the name of the working Task Group in the top banner of the screen. The top banner also may display over type mode with “OT” or block operations with “BLOCK” in the upper left-hand corner. The <Alt + F1> key toggles between insert and over type mode for the editor. Use the “Find” option from the PROGRAM MODE menu to initiate block operations.

The bottom bar of the screen shows what key functions are currently available depending on which menu or form is being displayed. Always check the bottom banner when unsure how to proceed or which key to press.

The main window is the State Logic editor that allows programs to be created and edited. The bottom border of this window displays the line number of the current cursor position.

Pressing the <F3> function key brings up the Program Mode Menu. There are several Hot Keys that can be used to short cut the menu steps. These keys are listed to the right of the menu option. The hot keys are also listed in the appendices of this manual and on the help screens. Press <F1> twice to see this listing.

The Help System is available in Program Mode by pressing <F1>. The Help system provides help that is in context with the operation currently being performed. Pressing <F1> twice brings up a list of all of the Hot Keys available. The General Help screen which is displayed when no menu is displayed, displays a screen of general information about ECLiPS and states that pressing <L> displays a list of help screens. To find help on specific operations select the operation from the list.

Creating Program Text - Overview

There are three ways that text can be entered into a State Logic program. The normal way that most program text is entered is by typing text from the keyboard as in any program editor. Options from the ADD menu can be used to enter new Task and State names and some specialty functions. The LIST option on the Program Mode menu enters names for I/O and variables into the program at the current cursor location.

Add Menu

ECLiPS uses the Add Menu options to assist the State Logic programmer. The add options use forms to help create program instructions. The forms accept information which is used to automatically build State Logic program instructions. Options on this menu create new States and

Tasks, and add diagnostic instructions. The most used option from this menu is the creation of the perform Functions.

List Menu

Choosing the list function displays a list of possible data types and other information that can be listed. Selecting a data type lists all the variables of that data type. When the <Enter> key is pressed, the currently selected name is entered automatically at the current cursor location.

For example if the name of a digital point is needed at the current cursor location, selecting "Digital Points" from the LIST menu displays a list of all of the digital points in alphabetic order.

To find the name in the list press the key for the first letter of the name. The first item starting with that letter is highlighted. When another letter is entered, the first name with the selected first two letters is highlighted. The arrow keys and <Page Up> and <Page Down> keys are also available. <Ctrl + Home> goes to the start of the list and <Ctrl + End> selects the last item.

New items are added to the list by pressing <Ins>, and the highlighted item deleted by pressing <Del>. To change the highlighted item press the <Right Arrow> key.

Lists are limited to 500 entries. After the end of the current list is reached, press <Tab> to work with the next 500 entries.

Text Menu- Block Operations

ECLiPS has several keys that can be used to manipulate blocks of text. These functions are accessed through menus, or through the use of hot keys.

Block functions are accessed by selecting the "TEXT" option from the PROGRAM MODE MENU or by using the Hot Key <F8>. The word "BLOCK" appears in the top banner and moving the cursor now highlights some of the text in the program. Press <Enter> when the desired text is highlighted.

A menu of block options is now displayed. The options are Copy, Move, Remove, or Perform multiple copies of the selected text. To copy, move, or perform multiple copies, select the option from the menu and move the cursor to the new text location and press <Enter>. For multiple copies continue moving the cursor and pressing <Enter>. Press <Esc> when no more copies are desired.

The text selected for any block operation is placed into a PASTE buffer. Text can be copied to another Task Group or another project by using the PASTE buffer. To add the contents of the PASTE buffer at the cursor press <Ctrl + U>.

Find Menu - Search and Replace

ECLiPS is equipped with common editing search and replace capabilities. These functions can be accessed through the Program Mode Menu under FIND or through a series of Hot Key sequences based on the <F5> key.

The FIND option on the Program Mode Menu is also used to go to the beginning of another Task and to list the current Translator Errors. From the list one of the errors can be selected by highlighting the error and pressing <Enter>. The cursor is then placed at the location of the

error and an error message is displayed in red at the bottom of the screen. For additional information about the error press <Alt + F1>.

<Alt + F5>	Search for a text string and replace it with specified text.
<F5>	Search for a text string
<Ctrl + F5>	Looks for each occurrence of a specified text string and gives the option to replace the string or to continue searching. <Esc> stops search.
<Shift + F5>	Finds every occurrence of a text string and replaces it with a different string. This command works on text in all of the task groups of a project.
<F7>	Looks for Another task, you must know the task name.
<Alt + F7>	Displays a list of project errors detected during the last translation operation.

Table 6- 1. The Hot Keys for FIND Functions

Moving Through the Program

ECLiPS uses a number of key sequences to facilitate the movement of the cursor through the document. The following keys can be used to control the cursor:

<Ctrl + Home>	Moves the cursor to the top of the program
<Ctrl + End>	Moves the cursor to the bottom of the program
<Home>	Moves the cursor to the start of the line
<End>	Moves the cursor to the end of the line
<Del>	Deletes single letter or highlighted block (see Text)
<Ctrl + Y>	Deletes the line the cursor is on. Text is lost.
<Ctrl + Left>	The Left arrow key with Ctrl moves cursor one word left
<Ctrl + Right>	The Right arrow with Ctrl moves cursor one word right
<Pg Down>	Moves cursor down one page
<Pg Up>	Moves cursor up one page
<Alt + F1>	Toggles between insert text and over type text

Table 6- 2. Cursor Control Keys

Project Menu

ECLiPS has a number of features that are designed to assist in the creation and maintenance of the projects. These features are accessed through the PROJECT selection of the Program Menu.

Project Management

The following features are used to manage projects.

"Retrieve"

The retrieve option displays a list of projects in the current working directory. Highlight the project of choice and press <Enter> to work with the selected project.

"Save"

This option saves the current project. See the BACK UP section for information about backing up projects and version numbers.

Note

Do not use the save option to make a copy of a project. The save option only save the current Task Group.

"New Path/Drive"

The current working directory is displayed and changes can be typed in directly to this form. To browse through the directories press <Insert>.

The second window shows the current directory and displays any of its subdirectories. Press the <Left Arrow> key to move to the parent directory and display its subdirectories.

To move to a subdirectory and display its subdirectories, highlight the directory and press the <Right Arrow> key. Pressing <Enter> changes the current directory to the directory displayed in the top of the second window

"Make a New Project"

Saves the current project and creates a new project in the current working directory.

"Copy"

This option copies the current project. A new drive and path can be specified. If no path is entered ECLiPS creates a new project in the current working directory.

Note

Do not use the save option to make a copy of a project. The save option only save the current Task Group.

Backup

This option controls the backup features. Options are to backup current project, restore current project, or enable/disable automatic backups.

Backup Project

Backups are saved in the BACKUP directory which is a sub directory of the current working directory.

Restore Project

Restore copies the project from the backup directory into the current working directory. The file in the current working directory is overwritten by the backup file.

Automatic Backup

With automatic backup enabled, a backup of the project is made whenever a project is downloaded. Backups are not made on "Save" or "Translate and Error Check". This procedure ensures that the backup copy of the project matches the project running in controller.

ECLiPS/Controller Version Mismatch

For safety reasons, ECLiPS does not allow communication with the controller if the ECLiPS version of the project does not match the version in the controller. If there is a mismatch in versions, ECLiPS offers an option to restore the back up version.

This option causes the backup version to replace the newer version. Any changes that have been made to the project since it was downloaded are erased.

If changes have been made to a Project and those changes should not be erased, first copy the project with the changes into a project with a new name using the "Copy" menu option. Then use the "Backup - Restore" menu option to restore the original project.

"Delete a Project from the Disk"

Deletes a project from the current working directory.

"Import"

The Import function combines all or parts of other projects with the current one. There are several options under the “Import” menu option.

The IMPORT "All Project Sections" option does a check for conflicts between the imported data and the existing project data. When conflicts are found, the options “Resolve the Conflict”,

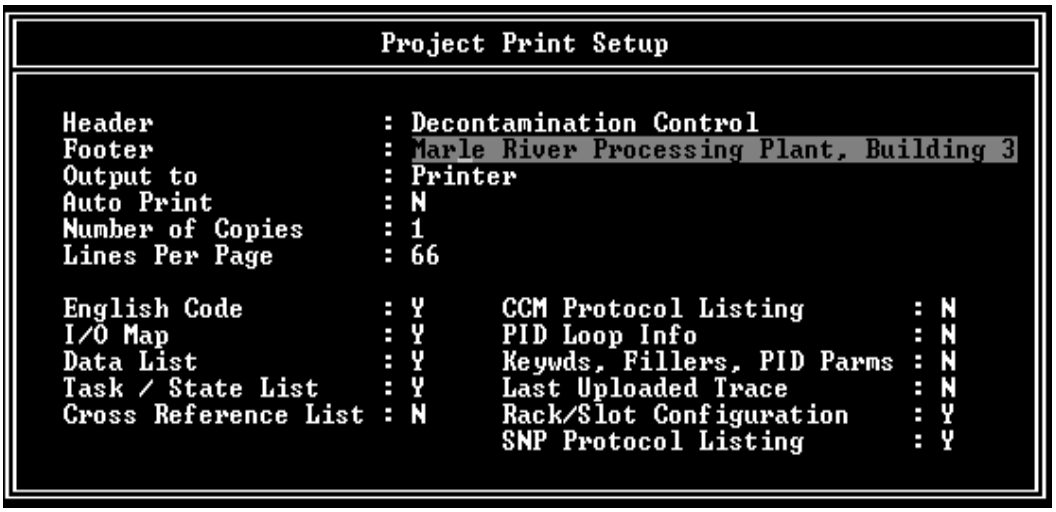
"Delete", and "Abort the Import" are presented. Changes to resolve the conflict change the imported data.

The "Import English Text" option does not do any conflict tests.. A new Task Group is added for the imported.

The IMPORT "I/O Configuration and Variable Names" imports the modules and racks that have been configured. Any Variable name that has been assigned to a hardware module are also imported. The imported data is checked for conflicts.

Documentation - "Print Function"

To print project documentation to a printer or file, select the "Print" option from the PROJECT menu. Selecting this option displays a form of documentation items.



Header -	Text Printed at the Top of Each Page
Footer -	Text Printed at the Bottom of Each Page
Output To	Any entry besides **Printer** directs documentation to a file with the specified path name.- **Printer** entry sends output to parallel port printer
Auto Print	When entry is 'Y' the print operation are executed whenever the program is downloaded to the State Logic CPU
Number of Copies	Number of copies to print
Lines per Page	Number of lines printed per page - some printers may need setting less than the 66 default
English Code	Prints the State Logic Program by Task Groups
I/O Map	Prints a list of all named I/O Points arranged in numeric order according to type
Data List	Prints a list of all variables and I/O points in alphabetical order according to type
Task/State List	Prints a list of Tasks followed by the States in each Task. The result is a useful outline of the entire project
Cross Reference List	Prints an alphabetical list of every named I/O point and variable and every Task and State where it appears in the program. This option may take a long time to print.
CCM Protocol Listing	Prints the CCM references providing access to the program values from a CCM device. These references are used for the programming port or SCM port only. Use the SNP references when using the CCM protocol with the CMM or PCM modules. - *See the chapter on Serial Communications for more information on using the CMM protocol.*
PID Loop Info	Prints all of the PID parameter initial value settings as specified in the initialization form
Keywords, Fillers, PID Parameters	Prints the current keyword and filler word definitions in addition to the PID parameter keywords
Last Uploaded Trace	Prints the Trace listing that was last viewed in ECLiPS or OnTOP
Rack/Slot Configuration	Prints a listing of the rack and slot configuration information
SNP Protocol Listing	Prints the SNP number and type for every named variable plus Tasks and States. *See the Serial Communications chapter for more information on using the SNP protocol.*

"Error Check, Translate, and Download Projects"

The process of converting the English program text into a format that is executable by the State Logic CPU is called translation. During the translation process the program is also checked for any errors. There are two options on the PROJECT menu to translate and error check the project: "Translate and Download Project to the Controller" and "Error Check and Translate the Current Project". The only difference between the two options is that the first also downloads the project to the State Logic CPU when the translation is complete. At the end of a successful translation a screen of statistics is displayed.

"Task Groups"

Task Groups are simply a collection of Tasks. Using Task Groups is a way of breaking a large project down into smaller and more manageable sizes. When more than one Task Group exists, the “Task Group” option is displayed on the MAIN menu. When only one Task Group exists, the option to add a new Task Group is displayed on the PROJECT menu.

The Task Group menu allows the Task Groups order to be changed. There are times although rare where the order of Tasks makes a difference in program execution. The Tasks at the beginning of the program are executed first. The Task Group menu allows the order of the Task Groups to be changed.

Instances where the Task order is important are when using variable or current State values in conditional terms. Variable assignments and State changes are visible to the rest of the program immediately. Discrete value changes are not apparent until the next program scan cycle.

Variable and I/O names and values are shared across the different Task Groups, and Tasks in one Task Group can control Tasks in another Task group.

Project Files

Project information is stored in files using the project name with different extensions for the different types of information stored. The following list shows filenames and the data they store for a project named Car_Wash:

Car_Wash.**PRJ**	Configuration and name information.
Car_Wash.**TG?**	The English text for Task Groups are stored in files with extensions TG0 -TG9 and TGA - TGF.
Car_Wash.**DBG**	Data about Debug operations, Force and monitor tables, etc.
Car_Wash.**PSM**	The translated program that is loaded to the State Logic CPU
Car_Wash.**TRC**	The information from the last uploaded Trace.

To move a project to another computer, the TG? and PRJ files are the only ones required. OnTOP requires access to these same files to connect to the controller.

The keywords and filler words are stored in the file, ECLIPS.KWD. When changes are made to the keywords and filler words, this file is the one that gets changed. The keywords are not stored in the project files. All projects automatically uses the same keyword definitions. When using a

project with another version of ECLiPS, transfer this file to make the new keywords available in this version of ECLiPS.

Naming Variables and I/O Points

State Logic programs refer to variables and I/O points by using names of up to twenty characters. There are several ways to set up and view names. Naming functions also appear on both the DEFINE and LIST menus.

A common method to name elements of the program is to create the program first, then have ECLiPS find all of the undefined names. To have ECLiPS find the undefined names select the "Translate and Download" or "Error Check and Translate" option from the PROJECT menu, or the "Undefined Words Throughout the Text" option from the DEFINE menu.

Name Data

The forms used to create names provide blanks for the different information required. Different information is required for different types of names.

Reference Number

All of the forms have a reference number blank. This value identifies the module and circuit location for I/O points. Variable reference numbers are used to identify data from a remote device such as an OIT or MMI connected through serial ports, Ethernet module, or the Genius Bus Controller module.

Save Over Halt

This YES/NO option is available for all data variable names and output names. Input forms display a blank which is not active. Inputs cannot be configured to "Save Over Halt".
When activated, this option causes this name to maintain its value over a power or halt/run cycle. Analog outputs and variables maintain their last value and digital outputs retain their last state. If this option is not activated, variables and analog outputs are initialized to zero and digital outputs are OFF when the program starts executing.

Naming Analog Channels

In addition to "Save Over Halt" and the "Reference Number" blanks, the analog name forms provide blanks for scaling the analog channel. Scaling allows analog data to be referenced in engineering units instead of raw D/A or A/D values.

Raw Analog Data

The actual raw values returned to the CPU by the Series 90-30 analog modules is in the range from -32,000 to 32,000 or from 0 - 32,000. These modules are 12-bit resolution modules which actually produce only 4096 values throughout this range.

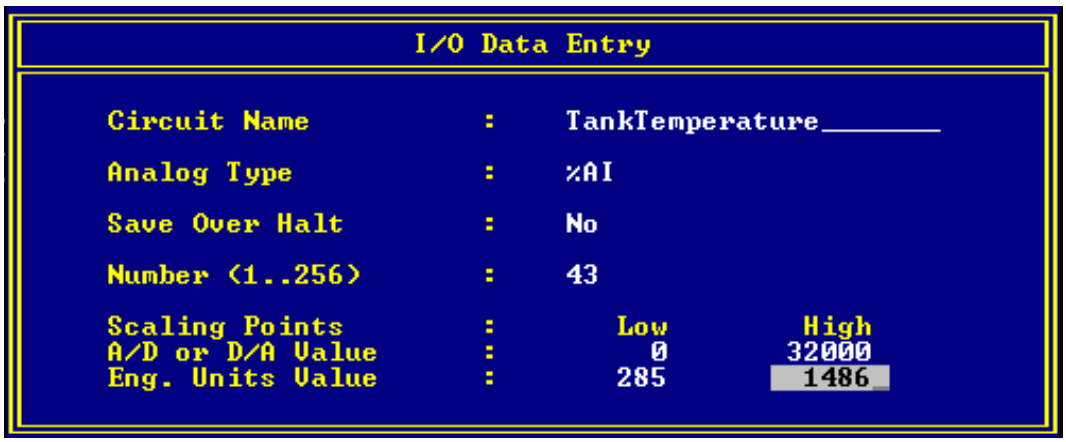
The reason for these ranges of values is that the three least significant bits of the 16-bit word are not used. The most significant bit is used to designate the sign of the value. The result of this method of representing analog data is that the values are represented in increments of 16. In other words the smallest change of analog data is a change of 16. When scaling analog channels, use values in these ranges for the A/D or D/A value.

If the scaling blanks on the analog naming form are left at all zeros, the analog channel is represented by raw unscaled data. Unscaled analog values are always integer data type in the State Logic control system.

Further information on the Stair Step Effect in Analog Modules can be found in the *Series 90-30 Programmable Controller Installation Manual, GFK-0356.*

Scaled Analog Data

To scale an analog channel fill in the four scaling blanks. Once an analog channel is scaled, all references in the State Logic program and in the Debug Mode screens display the scaled engineering units data. The raw data is still accessible from remote devices. *See the chapter on Serial Communications for more information on accessing analog data.*



Scaled analog values are always floating point data type. The scaling is always done linearly and any two data points can be used. It is not necessary to use the maximum and minimum values of the range of the module.

The scaling factors displayed in the form above create an engineering units scale covering a range from 286 to 1486 over the full range of the module. If the module were to return a value of 16000, the scaled value used in the program is 885.5.

Define Menu

The Define Menu provides four options for assigning, observing and changing variable and I/O names and definitions.

1. "**Define Current Word" -** This option returns the definition for the word at the current cursor location. If the word is an I/O name the reference number and rack/slot/circuit

information is also displayed. If the word is not defined, options are displayed for defining the word.

2. "**Search for Undefined Words"** This option scans the program for any words without definitions. When an undefined word is discovered, options are displayed for defining the word.

3. "**Define System Configuration"** This option displays lists of names for I/O points associated with each of the modules. Select the "I/O Names" option from the configuration menus for a slot to edit names associated with that module.

4. "**Variable Names"** This option displays a list of names for each of the variable types (Integers, Floating Point, Characters, and Strings, Internal Flags). The Variables are displayed in numerical order and separated by data type. To move to a reference number not listed on the screen ,press <Alt + F9>. Variable names are deleted by erasing a name from the list and pressing <F9> to save.

List Menu

The LIST option from the Program Mode menu is used to manage program and system names. Functions available include adding, deleting, and editing these names. In addition names are entered directly into the text from these lists.

Variable definitions are modified by positioning the cursor on top of the name in the list and pressing the right arrow key. The names are listed in alphabetical order, except for the list of keywords, which are listed in groups for each default and associated synonyms.

Pressing the <Insert> key when listing a data type, allows the addition of another variable name to that type of data. To delete a name from the list use the <Delete> key. The <Enter> key causes the selected name to be entered into the program at the current cursor location.

Pressing the <Left Arrow> key when the cursor is positioned on a I/O point will give the hardware location (Rack Number, Slot Number, and Circuit Number) for that point. The hardware information can not be changed from this screen. Hardware Configuration must be done under the System Configuration Option.

On-Line Program Changes

The State Logic program can be changed without halting the program running in the State Logic CPU. Changes to the State Logic program are made one Task at a time. To make State Logic program changes while the program is running, select the **"On-line Modify"** option from the MAIN MENU, and then select the Task to edit from the list of Task names displayed.

The Task modifications are sent to the controller by selecting the "Translate and Download", from the PROJECT menu. Before the changes take effect, ECLiPS asks which State to start the Task in, and provides a list of the available States in the modified Task. The process then uses the selected State in the modified Task as the current state in the next scan of the program.

Runtime Editing Restrictions

Programming in the On-Line Modify mode is very similar to programming in the Program Mode with a few restrictions.

- Digital I/O or Internal Flags cannot be added or changed
- Variables cannot be changed to “Save Over Halt”.
- The Project Management features cannot be used (copy, import, etc.)
- The system configuration cannot be changed
- New states can be added to the program. Up to five States may be added to any Task.
- Each time an on-line change is made, some of the CPU program memory is consumed. Continuous on-line program changes cause the CPU to eventually run out of program memory space. Lost memory is recaptured when the program is downloaded.

How to Use the ECLiPS Menus

There is one starting menu for each mode of ECLiPS. Press <F3> to view the main menu when in Program Mode or Debug Mode.

Options are selected from all ECLiPS menus in the same manner. Use the up and down arrow keys to highlight the option and press the <Enter> key. A quicker method of making a selection is to press the highlighted letter of the desired option. Each option has a letter (usually the first letter) that is displayed in a different color or highlighted (for monochrome monitors). The first level menu option can also be selected by pressing <ALT> plus the highlighted letter of the option when no menus are displayed.

Using ECLiPS Hot Keys

The most popular menu options may be selected without using any menus, by pressing a Hot Key. Any Hot Keys are listed to the right of the option on the menu where the option is displayed. A full listing of all key functions is available at any time by pressing the help key, <F1>, twice.

Appendix A has a table of Hot Keys. Notice that the Hot Keys are assigned in a manner to make it easy to remember their functions. All of the Hot Keys which refer to options from the Program Mode Project Menu use the <F2> key. The <Ctrl> key plus letters refer to options from the List Menu or for Debug Mode selections, for example <Ctrl + D> lists the Digital Points defined in the program. The <F4> key is used to work with the current word definition in both Program and Debug Modes; the <F5> key is used for the Find Menu options, <F6> for Add Menu options, and <F8> for the Text or Block options.

ECLiPS Editor Functions

The “Text” option offered on the Program Mode Menu is very useful for manipulating blocks of program text. The Text functions are used to copy, move, and delete blocks. These functions are very useful for manipulating blocks of the program.

When any of the Text functions are used, the block defined is saved in memory. This memory location is called the Paste Buffer. The Paste Buffer always contains the last block highlighted for any of the Text operations. The contents of the Paste Buffer are 'pasted' into the program at the cursor location by pressing the <Ctrl + U> keys. The PASTE function may be used to move or copy blocks of text. The PASTE function is also useful for copying blocks from one program to another.

Use the following steps to copy a block from one program to another program.

Press <F8>, the Hot Key for the Text functions
Highlight the desired block by using the cursor movement keys
Press the <Enter> key and choose the COPY option from the menu
Press the <Esc> key to cancel the operation, the paste buffer now has a copy of the block
Load another program or Task Group into ECLiPS
Press <Ctrl + U> to enter a copy of the block at the cursor location

How to Use ECLiPS Lists

ECLiPS uses lists to display the elements of the control program including I/O points, keywords, filler words, and variable names. The arrow keys and the <Page Up> and <Page Down> keys are used to move through the list. Also use <Ctrl + End> and <Ctrl + Home> to move to the end or the start of the list.

The elements of the list are displayed in alphabetical order. One method to quickly find an element is to enter the letters of the element. When a letter key is pressed, the first element starting with that letter is highlighted. Pressing another letter key causes the first element starting with those two letters to be highlighted. It is possible to enter all of the letters of the element to highlight it, but usually a few letters brings the highlight close to the target element.

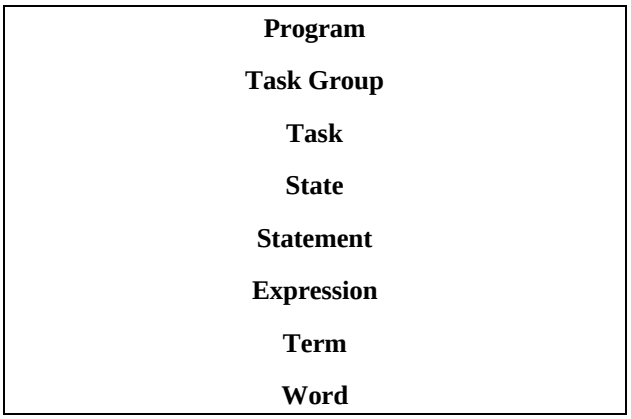
Generally the <Ins> key is used to add, the <Del> key to delete, and the <Esc> key to exit the list. Lists displayed using the "List" option from the Program Mode main menu, also allow the highlighted element to be entered directly into the program at the current cursor location when the <Enter> key is pressed.

Chapter 7 Program Instructions

This chapter describes how to write the State Logic program instructions. The hierarchy structure of the program is described, and there is a quick reference list of all of the types of both Functional and Conditional Terms. These lists are followed by more complex examples and detailed explanations of Conditional and Functional Terms. Also included are sections on mathematical calculations, variables, data types, a list of grammatical rules and an explanation of filler words.

Program Structure

There is a hierarchy in the structure of the State Logic program. Each program is divided into one or more Task Groups, each Task Groups may have one or more Tasks, each Task is divided into one or more States, etc. The figure below lists each element of the hierarchy in descending order of significance.



State Logic Program Hierarchy

The hierarchy of Tasks, States, and Statements are explained in chapters 3 and 4 of this manual. A program is a collection of Tasks Groups, a Task Group is merely a collection of Tasks and is used only as an organizational convenience. Tasks are a collection of States which describe a sequence of actions.

There may be many Tasks all executing simultaneously. Each State is described by one or more Statements and each Statement consists of Expressions. Expressions are constructed from Terms which are composed of words.

Task Groups

Task groups are simply a collection of Tasks. Each Task Group can have up to 60K bytes of the State Logic program. There may be up to sixteen Task Groups in a program.

Each Task Group is stored in a separate file. The file name uses the project name with the extension TG followed by 0-F. The first Task Group for the project PRESS is stored in the file PRESS.TG0.

Tasks

In the program each Task begins with the keyword, **Task**, followed by a colon then a Task Name.

Task: Assembly

Each Task includes all of the States from the start of the Task to the beginning of the next Task. The Task name may be followed by the keyword, **Start_In_Last_State**, which indicates that this Task is to restart with the State that was active when the program stopped running. This feature works over either a power cycle or a run/halt cycle.

Task: VatCooling Start_In_Last_State

State

Similarly, each State begins with the keyword, **State**, followed by a colon then a State Name.

State: Attach_Arm

Each State includes all of the Statements from the start of the State to the beginning the next State.

After the State name the **Max_Time** keyword followed by a number may be used. This function generates a diagnostic message if the State is active too long. The message is displayed in either the ECLiPS or OnTOP online Terminal Log screens. The value following the Max_Time keyword indicates the maximum number of seconds that the State should be active.

Statements

Statements, like English sentences, are terminated by a period. The first Statement of a State begins right after the State name and includes all of the expressions appearing before the period.

During program execution each Statement of the active State is executed in order starting at the first Statement in the State. The only exception occurs when a **GO** term is executed. Nothing else in a State is executed after a **GO** is executed.

Expressions

There are two types of expressions, Conditional and Functional.

Functional Expressions describe some action that the controller executes. A Functional Expression may be the only expression in a Statement or may be accompanied by a Conditional Expression.

Turn on HeaterCoil_1.
If TankTemperature is greater than 45.6, **go to the Cooling State**.

Functional Expressions in Bold Type

Conditional Expressions describe a requirement (permissive or test) which must be satisfied for the functional expression in the same statement to be executed. A Conditional Expression must always be used together with a Functional Expression.

If TankTemperature is greater than 45.6, go to the Cooling State.
Go to the Collection State, **when 5.5 seconds have passed**.

Conditional Expressions in Bold Type

Conditional Expressions can be a combination of several conditional instructions. Conditional instructions must be linked with either the AND or OR keywords.

If 35 seconds have passed and Parts_Count > 5 then go to Restart.
If ForwardProx is on OR ForwardMotor is on AND Torque > 95, go to ShutDown.
Go to ClosePress, **if 3.5 seconds passed and (autoPB is on or startPB is ON)**.

Examples of Multiple Conditional Instruction Expressions

Turn on DrainValve, DrainPump, and FlowLight.
If InComingSwitch is on, **add 1 to PartsCount and go to Clamp**.
Start_Pid TankTemperature, write "Tank Temp Auto", and go to Testing, if AutoPB is on.

Examples of Multiple Functional Instruction Expressions

Words

All words are classified into one of three categories: Keywords, Names, and Filler words. Words are always separated by spaces. Multiple words can be simulated in a single word using the underscore character, as in **Forward_Solenoid,** or mixing upper and lower case letters, as in **ForwardSolenoid.**

ECLiPS comes with a set of keywords already defined. The programmer can define synonyms for the keywords and even change the default keywords. Because of this feature the control program can actually be written in another language.

All filler words are ignored when the program is translated into a control program. The keywords and names are the only words which affect the execution of the program. Filler words are used only to make the program more readable.

Words are up to 20 characters in length, start with a letter and consists of letters, numbers and the underscore character.

Numeric Data Types

There are two State Logic numerical data types, integer and floating point. These values may be represented in the program as variables or constants.

Integer values are whole numbers in the range -32,768 to 32,767. Integer constants can be specified in hexadecimal format by preceding the number with '#'. For example, #77FF is a hexadecimal representation of the decimal number 30,719.

Floating point values are numbers using decimal points, numbers outside of the integer range of values, or numbers using scientific notation. Floating point numbers represent values in the range of -3.4E+38 to -1.2E-38 and +1.2E-38 to +3.4E+38. Floating point numbers may use scientific notation, but are not required to do so. Floating point constants must start with a number not the decimal point. For example .0345 is an **invalid** number which should be represented as 0.0345.

The numerical data types may be mixed freely within expressions. If any of the operations in an expression require floating point notation, all of the data elements are converted to floating point values for that calculation. If a floating point value is assigned to a variable of integer type, the floating point value is converted to an integer value observing the following rules:

1. All values are rounded to the nearest number.
2. Values outside the integer range are clipped to -32768 or + 32767 and a **Math Overflow Error** is generated.

Floating point operations require more time to perform than integer operations. Therefore, refrain from floating point operations as much as possible, if response time is critical to your application.

Variables

Variables are memory storage areas for different types of data. The different variable data types are:

1. Numeric

A. Integer Variables

B. Floating Point Variables

C. %AI Analog Inputs

D. %AQ Analog Outputs

E. Reserved System Variables

2. Digital

A. %I Digital Inputs

B. %Q Digital Outputs

C. %G Internal Globals

D. Internal Flags

3. ASCII

A. String Variables

B. Character Variables

All variables except inputs and reserved system variables can be configured to either retain values over a power and run cycle or to be initialized when the program starts running. To make a variable retentive, change the “Save Over Halt” option to ‘Y’ on the form used to define the variable.

Numeric Variables

The numeric variable types are Integer, Floating Point, Time, Analog, and Fault. The numeric data types are described later in this section.

Integer Variables

Integer variables store 2 byte integer values. The range of values represented by these variables are from -32,768 to 32,767. Attempting to assign an integer variable a value outside of this range, causes a math overflow, non-critical, runtime error, and the value is clipped to the minimum or maximum integer value.

A floating point value assigned to an integer variable is rounded to the nearest whole number.

Floating Point Variables

Floating Point variables store 4 byte IEEE format floating point values. Floating point values are numbers using decimal points, numbers outside of the integer range of values, or numbers using scientific notation.

Floating point numbers represent values in the range of -3.4E+38 to -1.7E-38 and +1.7E-38 to +3.4E+38. Floating point numbers may use scientific notation, but are not required to do so. Floating point constants must start with a number not the decimal point. For example .0345 is an **invalid** number which should be represented as 0.0345.

This format provides seven digits of precision. The display of a floating point number in a WRITE instruction or in an ECLiPS display may or may not be in scientific notation depending on the value being displayed.

Note

Whole numbers from -33,554,432 to 33,554,431 are represented precisely by this floating point format. Some whole numbers beyond this range cannot be represented precisely.

Analog Variables

Analog variables store the values of the analog I/O connected to the system. Analog input variables are referenced as %AI type and the analog outputs are %AQ.

Analog values are always floating point data type if the channel is scaled and integer if unscaled. Analog channels used directly in PID loop operations must be scaled values.

Analog outputs can be made retentive over a power cycle, but the analog inputs cannot.

Both analog inputs and outputs can be forced using the ECLiPS Debug mode or OnTOP. When forced, an analog input maintains the forced value, ignoring its real world value changes. Forced analog outputs ignore the program changes and just maintain the forced value.

Caution

Forced %AI data is not available to remote devices using the SNP or CCM protocols or through Ethernet and Genius communications. The actual value is transmitted to these devices, not the forced values.

Remote devices can access either scaled or unscaled analog values. References to scaled values use the %R reference while %AI and %AQ references are used for scaled analog values. See the *Serial Communications chapter in this manual* for more information on accessing analog data from remote devices.

Note

The remote devices cannot be used to directly change a %AQ reference. Use the %R reference to change a %AQ value.

Reserved System Variables

The system automatically has several variables predefined. The names of these variables cannot be changed. There are two types of Reserved System Variables, Clock/Calendar Variables and Fault Variables.

Clock/Calendar Variables

Clock/Calendar Variables store the current time information from the system clock. These variables are all integer values.

The reserved Clock/Calendar variables are:

- **Second**
- **Minute**
- **Hour**
- **Day**
- **Day_of_week**
- **Month**

The 331/340 platforms use a hardware real time clock that uses the battery power to maintain correct time and date values over a power cycle. The 311/313/323 clock/calendar variables get reset to zero after a power cycle, although they do retain their values over a run cycle.

The time and date can be set using any one of the following methods:

- Changing the Clock/Calendar Variables in the Program
- Using the “Set Clock in the Controller” Option from the Debug Mode PROJECT Menu
- Using the "Reserved System Variable" Option from the CHANGE Menu

Caution

Changing the clock while the program is running may cause some timing functions in progress to work erratically.

Fault Variables

There are reserved system variables that store fault information for use in the program. There is one variable for each slot in the system plus one for a Low Battery fault and one which indicates if there is any fault in the system.

The fault variables are listed under the Reserved System Variables category. These variables are integer variables that display a number code for a fault that is logged. Any non-zero number indicates a fault condition.

The program can test for fault events and individual faults using these variables. The fault variables are read-only variables which cannot be changed from the program. The Fault Variable Names are in the following table:

311/313 CPU	323 CPU	331/340 CPU
Battery_Low	Battery_Low	Battery_Low
Any_IO_In_Fault	Any_IO_In_Fault	Any_IO_In_Fault
Slot_1_Fault	Slot_1_Fault	Rack_0_Slot_1_Fault
Slot_2_Fault	Slot_2_Fault	Rack_0_Slot_2_Fault
. . .	. . .	. . .
Slot_5_Fault	Slot_10_Fault	Rack_4_Slot_10_Fault

Table 7- 1. PLC Fault Variables

See the online chapter in this manual for more information on the 90-30 State Logic fault system.

Digital Variables

Digital variables represent one bit of data. There are four types of digital variables:

- %I Digital Inputs
- %Q Digital Outputs
- %G Global Digitals
- Internal Flags

%I Digital Inputs

These variables represent input circuits which are assigned to modules during the configuration of the system. Digital inputs can be forced from the ECLiPS Debug Mode or from OnTOP.

Remote devices can change the %I reference variables, but if the point is configured to a module, the normal I/O scan overwrites the value every scan. If unconfigured %Is are changed from a remote device, the value is retained until changed remotely again.

%Q Digital Outputs

Digital outputs are always OFF by default. The output is ON only if there is an instruction in the active State of one of the Tasks that is directing that output to be ON or if that point is currently forced ON.

Remote devices (those using a serial protocol, PCM programming, Ethernet, and Genius communications) should treat %Q outputs as read only values, since the controller has control of these points. Digital outputs cannot be turned off by a remote device and remain on for only one scan when turned on by a remote device.

%G Digital Globals

The %G global points are internal digital points designed to be used as global data. These points have several uses and may be accessed by any external device. Other data types besides %G points can be used as global data.

The %G global points can be either inputs or outputs. The I/O status of %G points is determined by the configuration of Genius global data setup. Any %G points setup to be broadcast by either a GCM+ or GBC module become outputs. All other %G points are inputs.

The %G points that are inputs exhibit the same characteristics as %I points in that it is expected that some remote device controls their value, not the State Logic program.

The %G points that are outputs exhibit the same characteristics as %Q points in that they are OFF by default. The program only turns them ON when some instruction in the active State of a Task is turning them ON. The State Logic program is expected to control these points, therefore no remote device can change their values.

All %G points can be forced using the Debug Mode or OnTOP. Points that are forced cannot be controlled by an external device.

Note

There can be only two contiguous ranges of %G points configured as outputs. If there are more than two devices configured to broadcast %G points, configure the points to be contiguous, so that only two separate ranges are created.

Internal Flags

Internal flags are digital points that like %G points are not connected to any I/O device. All of the internal flags behave as %Q outputs since their value is controlled by the State Logic program. They are OFF by default and cannot be controlled by an external device.

Internal flags can be forced using the Debug Mode or OnTOP. These flags are most often used to communicate from one Task to another when an event has occurred. Many programs use no flags, since one Task can test the current State of another Task. This interaction capability between Tasks makes it unnecessary to use flags in most instances. Most programs test the State of Tasks, since programs using this method are easier to understand. One reason to use flags instead of testing the State of a Task is that setting and inspecting flags is faster.

ASCII Variables

The ASCII variable types are Character, and String. Character Variables store one character and use one byte of memory. String Variables store up to 80 characters and use 82 bytes of memory, since every string is terminated by a null character, (00), and memory is used in two byte increments.

String Variables store any ASCII characters. Control characters are used in the string variable by using the % followed by the # and then two digits that are the hexadecimal number for the ASCII character, for example:

```
Make Test_String equal "abc%#1Bxyz".
```

This example is an assignment to a String Variable to store the characters abc the escape character and then xyz.

Functional Instructions

This section explains the functional instructions. The first table shows a quick reference list of all the functional instructions with an example of each instruction. The rest of the section describes the syntax and details of using each of these instructions in the program.

INSTRUCTION	EXAMPLES
Turn On Digital I/O	**Actuate** DropGateSolenoid.
Assign Analog/Variable Values	**Make** ExtrusionLength = 23.45 **Make** VatHeater = 75.5. **Make** WarningString = "Reservoir LOW".
Perform Calculations	**Add** 1 to PartsCount. **Make** SlabArea = SlabHeight * SlabWidth. **Make** RLevel = SIN(2*RadianAngle) + 75.3.
Change Active States	**Go** to the Estop State.
Change State of Another Task	**Put** FlowControl Task into NormalFlow State. **Suspend_Task** the CutterControl Task. **Resume_Task** the Cutter_Control Task.
Send Serial Data Port	**Write** "Tank Full" to OperatorDisplay.
PID Control	**Start_PID** TorchTemperature. **Stop_PID** WaterFlow.
Execute a Perform Function	**Perform** Time_Counter with Action='A' Integer_Variable=NumberOfMinutes TimeInterval='M'.

Table 7- 2. Functional Instruction Quick Reference List

Turning ON Digital Points

The Actuate Instruction is used to turn on %Q Digital Circuits and %G Global Data, and Internal Flags. All of the digital points are OFF by default. Transition to a State that is not turning an output ON has the effect of turning the output OFF. There is no keyword that turns an output OFF.

DEFINITION: The Statement to turn ON a digital point starts with the keyword **Actuate** or one of its synonyms, followed by one or more discrete names.

```
Actuate the Ready_Light.
Start Pump_1 and Pump2.
Energize Clamp_1, Clamp_2, Clamp_3 and Clamp_Flag.
```

Assigning Analog Output and Variable Values

To assign values use the Make Instruction, Math-Assignment Instructions, Set_Bit/Clear_Bit Instructions. These Instructions assign values to variables and analog I/O points.

Make Instruction Definition:

The Make Instruction is used to assign a value to a variable or analog I/O point. The Instruction starts with the keyword **Make** and is followed by a variable or analog name, the keyword **equal**, then a number or a calculated value.

```
Make Flow_Setpoint equal 25.
Make Valve_Control = 67.89.
Make Total_Defects equal Temperature_Failures + Stress_Failures.

Make Output_String equal "Enter setpoint now".
Make Test_Character = '$'.

Make Tank_Level_PID Bias equal 34.456.
```

See the section "Calculated Values" for a description of mathematical calculations.

Math-Assignment Instruction

For mathematical calculations where a simple operation is performed to a numeric variable use the Math-Assignment Instructions. These simple four function math instructions are **ADD, SUBTRACT, MULTIPLY, and DIVIDE.**

The Add Instruction is the keyword **Add** followed by a number or variable name then a variable name.

```
Add 1 to Parts_Count
Add Second_Shift_Parts_Count to Total_Parts_Count
```

The Subtract Instruction is the keyword **Subtract** followed by a number or numeric variable name then a variable name.

```
Subtract 2.78 from Starting_Value
Subtract Tare_Weight from Test_Weight.
```

The Multiply Instruction is the keyword **Multiply** followed by a variable name then a number or variable name.

```
Multiply Parts_Lost by 2
Multiply Machine_Strokes by Strokes_Per_Cycle
```

The Divide Instruction is the keyword **Divide** followed by a variable name then a number or variable name.

```
Divide Right_Side_Length by 4.5
Divide Box_Volumn by Volumn_Adjustment
```

Set_Bit/Clear_Bit Instruction

The Set_Bit/Clear_Bit Instruction is used to set an individual bit to either 1 or 0 in an integer variable. First the **Set_Bit or Clear_Bit** keyword is used then the variable name followed by the zero based bit number.

```
Set_Bit Transfer_Status 2
Clear_Bit Tac_Register 0
Set_Bit PID_Loop1 Track_Mode
```

State Transitions

A Task can change its own State and can also change the current State of another Task. Changing the State of another Task is one way that Tasks coordinate their activities.

Changing the Current State within a Task

The **GO** instruction is used by a Task to transition to another State. Only the Go and the State name are mandatory. All other words are optional. The Go may appear in any Statement but there may only be one Go per Statement.

```
Go to the Forward_Motion State.
Go EmergencyStop.
```

As soon as the GO instruction is executed, the new State becomes the active State for that Task. The new State information is available immediately. Other Tasks testing the State of the Task see the changed State value in the same scan.

A very **important** characteristic of the GO instruction is that once a GO is executed, no more statements are executed in that State, although the rest of the instructions in that Statement are executed. This property of the GO instruction can be a very useful programming tool and also means that the order of Statements within a State can impact the execution of the program.

The following examples demonstrate how using this feature of the GO instruction can be used to optimize program response time.

Example 1

```
State: RunningConveyor
If Prox_1 is ON go to the Conveyor_OFF State.
Turn ON the Conveyor_Motor.
```

Example 2

```
State: RunningConveyor
Turn ON the Conveyor_Motor.
If Prox_1 is ON go to the Conveyor_OFF State.
```

Example 1above produces the fastest response for turning the conveyor OFF, after the proximity switch comes ON. The reason that example 1 is faster is that, the first scan of this State after the proximity switch comes ON, the Statement to turn ON the conveyor motor does not get executed. Therefore the conveyor motor goes OFF, one scan faster than the order used in example 2.

Example 3

```
State: CheckPartInPlace
   If  PartInPlace is ON, turn on the Diverter and go to the DiverterON State.
State: DiverterON
   Turn on Diverter.
```

Example 4

```
State: CheckPartInPlace
   If PartInPlace is ON, go to  the DiverterON State.
State: DiverterON
   Turn on Diverter.
```

Example 3 above produces a quicker response for the diverter coming ON from the time that the PartInPlace switch comes ON. Example 3 turns on the diverter one scan faster than example 4, since example 4 requires the program to change State before the output comes ON.

Example 4 is more correct programming practice, since generally it is good practice to follow the rule that the action for a conditional expression is to always GO to another State. Use the example 3 structure only when response time is critical.

Example 5

```
State: TestStartConditions
        If TemperatureAlarm is OFF and PressureStatus is ON and
PartPositioned is ON, GO to the StartOperations State.
```

Example 6

```
State: TestStartConditions
        If TemperatureAlarm is ON, go to the TestStartConditions State.
        If PressureStatus  is OFF, go to the TestStartConditions State.
        If PartPositioned is ON, go to the StartOperations State.
```

Example 5 shows a typical status checking State where several conditions must be met before the process continues to the next step of a process. A State structured like example 5 can add a lot to the scan time, especially if there are many conditions to check. Each condition must be checked every scan.

Example 6 shows an alternate way to program this status check. This example uses the State Logic ability to go to the same State and also the fact that the Statements following an executed GO instruction are not executed. This programming structure limits the number of Statements being scanned and therefore the scan time of the system. For example if TemperatureAlarm is ON, that is the only term that is scanned, since the program immediately Goes again to the TestStartConditions State.

The operation of example 6 is not intuitively understood, therefore it is recommended that this structure not be used unless it is necessary to reduce the system scan time.

Changing the State of Another Task

Controlling the current State of one Task from another Task, is a main method of coordinating Task activities. There are two types of instructions that allow one Task to control the State of another Task:

1. Setting the Task to a new State value
2. Halting and restarting Task operations

The following example shows how a Task can change the currest State of another Task.

```
Put the Assembly_Control Task into the Emergency_Stop State.
```

The structure of this Statement is the same as assigning a variable some value. Put is a synonym for make, and into is a synonym for equal.

For halting and restarting Tasks use the Suspend_Task and Resume_Task keywords. The Suspend_Task keyword saves the current State for the Task and puts the Task into the Inactive State.

Note

Every Task has an Inactive State by default, even though the Inactive State does not appear in the program. A Task in the Inactive State does no operations and cannot by itself change to another State.

The Resume_Task keyword causes the named Task to go to the State that was active before being suspended.

```
State: CheckHighLevel
        If Water_Level is above 45.5 feet then Suspend_Task Fill_Tank and
go to the CheckRestartLevel State.

State: CheckRestartLevel
        If Water_Level is below 43.8 feet Resume_Task Fill_Tank and go to
the CheckHighLevel State.
```

This example puts the Fill_Tank Task into the Inactive State, when the level is above 45.5. When the level drops to below 43.8, the Fill_Tank Task is placed back into the State that was active when it was suspended.

Note

If the Task has not previously been suspended, the Resume_Task function causes the Task to become inactive.

Serial Output (Write Instruction)

The Statement to send data out a serial port is the keyword Write followed by data to send inside double quotes. Optionally a communications port name may be specified following the data to be sent. If no port name is specified the data is sent to the programming port to be displayed by OnTOP or ECLiPS.

```
Write "Push Start Button" to Operator_Control.
```

The serial data can be a mixture of typed text, variable values, ASCII control characters, and formatting characters.

The typed text are any characters entered directly from the keyboard. The text may include carriage returns so that several lines can be entered in one Write Instruction. Multiple line messages can be formatted in the program exactly as they appear on an instructioninal screen.

```
Write "
Opening Operator MENU

  1. Change Today's Date
  2. Change the Current Time
  3. Engage Startup Procedure
  4. Restart the Process"
     to Operator_Panel.
```

The menu created by the Write Instruction above appears on the operator screen just as it does in the program. The limit of the number of characters between the quotes is 512, which is about 6 full (80 character) lines of text.

Variable values are sent out the port by preceding a variable name with a "%". Any variable type, including analog I/O values but excluding discrete values, can be sent out the serial port.

```
Write "Current parts count is %Part_Count." to Operator_Terminal.
```

If the variable, Part_Count, has a value of 10 at the time the Write Instruction above is executed, the following line is displayed on the screen connected to the port named Operator_Instructioninal.

Current parts count is 10.

Formatting Characters that are used with the Write Instruction follow:

%NOCRLF - Write Instructions always send a carriage return line feed pair following each message. Use this formatting feature to suppresses these instructioninating characters.
%CRLF - sends a carriage return line feed character pair
%CRLF(X) - X number of carriage return, line feeds
%CLS - Clear the Screen, sends 25 carriage return, line feeds
%SPACE(X) - X number of spaces

Control or Escape Characters can be embedded in the string of characters. Embedded control characters have many uses, controlling screen displays for dumb instructioninals and PCs using ANSI.SYS device driver, some serial devices also use control characters for special function control.

%CHR(XX) - The ASCII character for the decimal value in the "()" is sent.
%#XX - The ASCII character for the hexadecimal value XX is sent.

Each of the following examples sends a carriage return to the port named OperatorPanel.

```
Write "%CHR(13)" to OperatorPanel.
```

```
Write "%#0D" to OperatorPanel.
```

Specialty Characters - The " | " character is used to place two words together without any spaces in between them. For example, "%Pressurepsi" looks like one long variable name to ECLiPS and would yield an error message. But "%Pressure|psi" yields the desired result of the value of the variable called Pressure directly followed by the character string, "psi".

To send a double quote sign use "%#22" or "%CHR(34)". To send per cent sign use "%%". All other keyboard characters are sent by simply typing them between the quotes.

Note

Each state is limited to 32 write instructions. If more than 32 write instructions need to be used, a state that writes error codes for example, then split the write instructions into two states.

PID Loop Control (Start_PID, Stop_PID)

PID Loop control Statements start with the keywords Start_PID or Stop_PID, followed by the PID Loop Name. If stopping a PID loop, a value which sets the value of the control variable can follow the PID loop name.

```
Start_PID Oven_1.
Stop_PID TankLevel.
Stop_PID KILN5 with 456.29.
```

When the PID loop begins execution, it uses parameters specified in the PID initial values form accessed through the DEFINE or LIST menus. See the chapter on PID loops for more information.

Note

Make sure that the Start_PID Instruction is not executed repeatedly every scan. Always change State after starting the loop operating. The PID loop does not execute as expected when it is constantly restarted every scan. For more information on PID Loops see the PID loop chapter.

Change Serial Port Configuration

The Instruction to change the configuration of a serial port is the Set_Commport keyword followed by a port name and then a list of parameters and their values. *See the chapter on Serial Communications for more information about serial communications.*

This Functional Instruction is automatically entered into the program by using the "Communication Ports" option on the LIST menu. Select the port and press the <right arrow> key to change the configuration options. When configuration is complete, press <F9> to save. A list of Comm Port names is displayed.

Highlight the desired port name and press <Enter>. Two options are presented, enter port name or enter reconfiguration data. Select the "Insert Reconfiguration Data for the Port" to enter the entire instruction into the program at the current cursor location.

Perform Functions

A perform function is the Perform keyword followed by the function name, the keyword with and then a list of parameters and values. Usually this Instruction is entered into the program automatically by ECLiPS after the form accessed through the Add menu is completed. The Perform instruction is entered at the end of the State where the cursor is located.

First select the "Add" option from the PROGRAM MODE MENU, then the "Add a Perform Function" option. Fill in the blanks that are displayed after selecting the function desired. *See the chapter on perform functions for more information.*

Conditional Instructions

Conditional Instructions are used to test for conditions of discrete, numeric, analog, time and character values and also to test when data has been received in a serial port. Conditional Expressions must be true for the Functional Expression in the same Statement to be executed.

TYPE	EXAMPLES
Digital	If PressDownSwitch is ON . . .
Time	If 3.5 seconds . . .
Relational	If TankLevel is greater than 45.67 . . .
Current State	If the CleanInPlace Task is in the WaterRinse State . . .
Serial Input	Read OperatorInput from OperatorDisplay . . .

Table 7- 3. Conditional Instruction Quick Reference List

Digital Conditional

The Digital Conditional Instruction tests the status of digital memory types %I, %Q, %G, internal flags, and also specific bits of integer variables.

Testing Digital Types

To program this type of test of the digital types use the IF keyword followed by the name of the discrete point then followed by the keyword ON or OFF.

If Forward_Limit_Switch is **on** . . .
If Part_Ready_Flag is **off** . . .

Several digitals can be specified in the same expression joined by AND or OR keywords as follows:

If Top_Limit_Switch or Bottom_Limit_Switch and Counter_Weight_Switch are **OFF** . . .

The ANDs are executed first when both ANDs and ORs are in the same expression. This order can be changed by using parenthesis to surround expressions which should be evaluated first.

Testing Integer Variable Bits

Individual bits of a variable are tested using the **Test_Bit** keyword. The syntax of this conditional is the keyword **If** followed by the **Test_Bit** keyword then the integer variable name and zero based bit number in parenthesis, followed by the **ON** or **OFF** keyword.

The following example tests the fourth least significant bit of the integer variabl named Status_Variable. This conditional is true for several values of Status_Variable, among these are 8, 10, 24, 40.

```
If Test_Bit (Status_Variable 3) is ON . . .
```

A common use of this test is checking the results of shift register operations. The bits of the last variable of the shift register can be tested using this function. The following example shows a test if the most significant bit of the variable Shift_Variable_7 is zero.

```
If Test_Bit (Shift_Variable_7 15) is False . . .
```

Timer Conditional

The Timer Conditional is a number or variable followed by the keyword SECONDS (must be plural). The timer has a resolution of 1/100 of a second and the value used to indicate the number of seconds can be a floating point number.

```
If 3.76 seconds have passed, then . . .
```

An integer variable can also be used to specify the number of seconds. The value of the variable indicates the number of hundredths of a second, so that a variable value of 100 would indicate a time delay of 1 second.

```
If Wait_Time seconds, then . . .
```

Timers always refer to the amount of time that the State has been active. A common mistake is to assume that the timer starts when the instruction before it becomes true.

```
If Track_Monitor is ON and 5.3 seconds have passed . . .
```

The timer above always refers to the time that the State that it is in has been active and is not influenced by the condition of Track_Monitor.

The timer number must be in the range of 0.01 to 600.00 seconds which is a maximum of 10 minutes. When using an integer variable, the variable value must be a positive value.

There are several ways to make a timer that uses a period of time greater than 10 minutes. The common methods use State transitions to reset a State timer.

```
State: Heater_On_One_Hour
   Actuate Vat_Heater.
   If Ten_Minute_Counter is >= 6, go to Start_Process State.
   If 600 seconds have passed  then add 1 to Ten_Minute_Counter
         and go to Heater_On_One_Hour State.
```

Also see the Time_Counter Perform function for a description of other ways to handle timing situations in State Logic programs.

Relational Conditional

Relational Instructions test variable and analog values. The Instruction is a value, followed by a relational operator, then another value. The values tested can be numbers, calculations, variable names, and analog names.

If Parts_Count = 500 . . .
If Flow_Meter_Input is **above** Flow_High_Limit . . .
If Canister_Pressure - Atmosphere <= Pressure_Limit - Safety_Margin . . .

If String_Entry **equal** "Formula 1" . . .
If Test_Char is **not_equal** to '@' . . .

See the section, Mathematical Calculations, for a discussion on how to use calculations with Conditional Instructions.

State Conditional

The Current State Conditional is a Task Name followed by the keywords EQUAL or Not_Equal and then a State Name. The keyword is is a synonym for EQUAL. This conditional is used to test the current State of another Task.

If Pump_Monitor Task **is** in the Backwash State . . .

A Task can test the current State of any other Task. This instruction is one of the main ways that Tasks can coordinate their activities.

Complex Conditionals

The Conditional Instructions can be combined to form a Conditional Expression using the keywords AND and OR keywords. The AND Instructions have lower precedence and are therefore executed first. The order of execution can be changed by use of parenthesis and parenthesis can be nested. The keyword NOT can also be used preceding the conditional instruction.

If Hydro_Pump_Cont Task **is** in Over_Pressure State **or** Hydraulic_Prssure is **above** 23.56 . . .

If 1 **seconds** and **not** Temperature_Setpoint **greater** than 4.67 / Settling_Value . . .

If Spin_Drive is **ON** and (Pour_Ladle **is** in the Pouring State or **not** Mold_No is **above** 67) . . .

Serial Input Conditional

The syntax for this conditional is the keyword READ followed by a variable name. This conditional is true when a character input message is completed. The character input is stored in the variable listed.

Optionally this conditional can specify the port from which the input is received. If this option is used the keyword FROM follows the variable name and then a communications port name is listed.

```
Read Menu_Choice from Operator_Station, then go . . .
```

The above example reads data into the variable named, Menu_Choice, from the communications port named Operator_Station.

A GO Functional Instruction must always follow the character input conditional and there cannot be any other Instructions in the Statement besides the READ instruction and the GO instruction.

If two READ instructions for the same port are both currently active (both in an active State) it is unknown which conditional receives the message from the port. The program should be written such that all READs for a port occur in the same Task, assuring that two READs for the same port are not both active at the same time.

The types of variables used with the Read are:

Integer Variables
Floating Point Variables
String Variables
Character Variables

If the type of data received does not match the variable type, the input is ignored and the conditional is not satisfied. An example of invalid data is entering string of characters to a numeric variable.

The input is completed and the READ Conditional Instruction is true, when an end of message character is received at the port. If the READ Instruction is not true, the program scan continues on to execute the next Statement. When the READ is true, the variable is assigned the data received in the port and the GO instruction following the READ is executed.

The default end of message character is the carriage return, so that when receiving input from a instructional the READ is completed when the <Enter> key is pressed. The end of message character may be changed using the serial port setup forms. See the Serial Communications chapter for more information.

Input to a character variable is complete as soon as one character is received, so that a character is stored and the GO executed as soon as any character is received. End of message character is not used.

Note

Character variables cannot be used to receive input through the programming port when connected to ECLiPS or OnTOP.

Mathematical Calculations

Mathematical calculations are used in Functional Instructions as in this assignment Instruction:

```
Make Pointer_Position = Last_Position * (Forward_Pressure + 345.8)
```

and in Conditional Instructions as in this comparison instruction:

```
If Advanced_Magnitude < SIN(Current_Angle) / 45.6 go to Reposition State.
```

Numerical expressions may be much more complicated using any of the operators in any order and nested in parenthesis to change order of evaluation or make the expression more readable. Up to 18 levels of parenthesis may be used.

Operator Precedence

Operators are executed according to their precedence. The operators with the lowest precedence number are executed first. Operators with the same precedence are executed left to right. Use parenthesis to change the order of execution. *See the keyword appendix for the precedence of each operator.*

System Clock/Calendar

This section describes how the State Logic Controller clock/calendar is set and accessed. Also described are the differences between the 311/313/323 CPU and the 331/340 CPU implementation of the System Clock.

The clock/calendar data is accessed through the Reserved System Variables listed below:

- **Month**
- **Day**
- **Day_Of_Week**
- **Hour**
- **Minute**
- **Second**

These variables are integer data type and can be inspected and changed from the State Logic program. The time and date can also be set by using the Debug Mode or OnTOP "Set PLC Clock" option from the PROJECT menu. The Debug Mode or OnTOP "Display" option and monitor tables are used to inspect the values of the time and date variables.

Differences Between the 311/313/323 and 331/340 Clock/Calendar

The clock/calendar values for the 311, 313 and 323 CPUs are not saved over a power cycle. Each time power is cycled the time variables are reset to zero. These variables must be set to the current time and date each time power is applied to the 311,313 and 323 State Logic CPUs.

The 331 and 340 CPU time and date values are saved and updated over a power cycle. When power is lost to a 331 or 340 CPU system, the time and date are continuously updated as long as the system battery is functioning properly. When power is reapplied to the 331 or 340 system, the time and date variables are at their correct values and do not need to be reset.

Grammatical Rules

- Every Task must begin with the word "Task:" followed by the Task name.
- Every State must begin with the word "State:" followed by the State name.
- Every Statement must end with a period.
- Every Statement must have a functional expression.
- Only one "Go" is allowed per Statement.
- Only one "Read" is allowed per State.
- If a "Read" Instruction is used in a Statement, it must be accompanied by a "Go" in the same Statement. There may be no other Instructions in the Statement.

Filler Words

Filler words have no functionality, i.e. they do not change the meaning of any of the statements in which they appear. Filler words are only be used to increase the clarity of the English text. For example, "go to the Motion State" looks better and sounds better than "go Motion". To some programmers, however, typing fewer words is better.

Chapter 8 Perform Functions

The perform functions implement operations which are more complicated than the common State Language Terms. Perform functions provide a form to facilitate using the function. To use the perform function, fill in the form that ECLiPS provides. Once the form is completed press <F9> to save the data, ECLiPS then enters the text for the perform function at the current cursor location. To access the forms select the "perform functions" option from the ADD menu.

General Information

To enter a perform function to the program, use the forms provided through the Add menu. Except for the specialized perform functions, there is a common form to help create perform functions. An example form for Time Counter shows the headings for each of the columns on the form.

The meaning of each of these columns is described below:

Parameter Name	Identifies the parameter describe in this row of the table.
Type	Specifies the parameter data type and may be integer, floating point, character, string, digital, or any type.
Use	Specifies whether a variable, constant, or both may be used for this parameter
Required	Specifies whether this parameter is required in the function call. The parameters that are not required are at the end of the parameter list. No parameters may be entered following one that has not been used.
Actual	This is where the variable or constant for the parameter is entered for this particular function call.

Table 8- 1. Typical Perform Function Parameter Form

Use the Help system for assistance in using the forms. While a perform form is displayed, Help for that form is available using the <F1> key.

Note

The State where a perform function appears should usually be structured so that the perform is executed for only one scan. A common error in using performs, is that the program is structured such that the perform is executed several times, i.e. each time the State is scanned.

Table functions

The set of table functions allows several functions to be performed using data stored in a table or array type fashion.

All of these functions work on the same tables or arrays of data. A Table is a two dimensional array of values that can be either floating point numbers (float), integer numbers (integer), sequence of characters (string) or binary numbers (digital I/O status). The elements of the Table are accessed by specifying a ROW and COLUMN number for the cell containing the data. Every element of that table must be of the same type and when the Table is defined the type of variable is established.

There can be a maximum number of 100 Tables, each assigned a unique number from 1 to 100. These tables can be of any size determined by the number of rows and columns assigned to them until the maximum amount of memory allocated for Table use is consumed.

A Table must first be defined before it can be used. This is done using the four Table Define functions. There are four Swap_Table_Value functions named Swap_Table_Value_Int, Swap_Table_Value_Float, Swap_Table_Value_Dig, and Swap_Table_Str. The swap functions allow values to be exchanged between a variable and a table of the same data TYPE.

There are three Init_Table functions named; Init_Table_Integer, Init_Table_Float, and Init_Table_Digital. There is no initialization function for a string table. The init functions will initialize a table with a series of values that will be written from a series of variables into a table of the same data TYPE. The Copy_Table_To_Table function will place the values in one table into another table of the same data TYPE. The table to be copied into must be the same size or larger.

All Table functions check to make certain that when a Table is selected for use, it fits the definition that has been previously created. Thus a Swap_Value to a Table defined as integer must have an integer variable, or both Tables selected in a Copy_Table_To_Table must be of the same type, or a Swap_Value cannot refer to a row number for a table that is greater than the total number of rows defined for that Table.

In the event of misuse of the functions being detected either ECLiPS will produce an error at download time or (and this is most likely) an error message will be generated at run time.

It should be noted for all the Table functions the row number comes first followed by the column number.

Define_Table

Every Table that is used must be defined. Defining the table assigns a data TYPE to the table and gives the table a size. When this function is selected from the perform menu, a menu displays the following options:

Number_of_table	the Table number from 1 to 100
Type_of_table	the type of variables (I for integer, D for digital or F for float, S for String) stored in the Table elements
Number_of_rows	the number of rows of the table
Number_of_columns	the number of columns
Save_value_over_halt	indicates whether the Table should be saved through a halt-run cycle. Enter Y or N.

Table 8- 2. Define Table Parameter Form

If the table number is defined more than once, the original table definition is kept. The only other non-critical run time error that can occur is if the table number specified is greater than 100.

Entering and Retrieving Table Values

There are four Swap_Table_Value functions that are exactly the same except they work with different data types.

- Swap_Table_Value_Int
- Swap_Table_Value_Flt
- Swap_Table_Value_Dig
- Swap_Table_Value_Str

These functions exchange values between a variable and a table. These functions Write a value from a variable into a Table element or to Read a value from a Table element into a variable. There are four distinct functions, one for each different data TYPE. Selecting one of these functions from the perform menu will bring up a form requesting the following information:

Number_of_table	the Table number from 1 to 100
Type_of_operation	the type of operation (R for read or W for write) to be performed with R meaning to take a value from the table and place it in the variable, and W to enter the value of the variable into the table.
Row_number	the row number of the element to be read from or written into
Column_number	the column number of the element to be read from or written into
Variable	the name of the variable to be used to store the Table value during a read or get the value during a write

Table 8- 3. Swap Table Parameter Form

The State Logic CPU generates a run time critical error if the Table selected does not match the type of Swap being used or if the row and column numbers are out of range for the selected Table.

Initializing Tables

The three Init_Table functions set values for multiple elements in a Table at one time. There are three distinct functions, one for each of the different data TYPEs available with this function. There is no initialization function for String Tables.

- Init_Table_Int
- Init_Table_Flt
- Init_Table_Dig

Selecting one of these functions from the perform menu will display a for requesting the following information:

Number_of_table	the Table number from 1 to 100
Row_number	the row number of the first element where the values listed are to be stored
Column_number	the column number of the first element where the values listed are to be stored
Number_of_values	the number of values that will be stored in the following consecutive Table elements
Value_1	a constant or the name of a variable (of the same type as the Table) to be stored in the first Table element identified by the Row_number and Column_number
Value_2	a constant or the name of a variable (of the same type as the Table) to be stored in the first Table element after the Table element identified by the Row_number and Column_number
..	
Value_28	a constant or the name of a variable (of the same type as the Table) to be stored in the last Table element identified by the Row_number and Column_number

Table 8- 4. Initialize Table Parameter Form

There can be up to 28 values initialized with each individual function and they can begin at any Table element location. Each column element of a row is filled in before the next value is placed in the next row.

The State Logic CPU will generate run time critical errors if the Table selected does not match the type of Swap being used or if the row and column numbers are out of range for the selected Table or if the number of values added to the starting element position would go beyond the last element defined for the Table.

Copy_Table_To_Table

The Copy_Table_To_Table function allows the User to copy one Table's values into another Table. The Tables must be of the same type and the Table that the values are copied from must have a less or equal number of rows and columns than the other Table.

The Table's elements will be copied into the corresponding Table's elements. If the first has less rows or columns than the second, then the elements that do not exist in the first Table will be left unchanged in the second Table.

When the User selects one of these functions from the perform menu the following information will be requested:

Table_To_Copy_From	the number of the Table from which the values will be copied. Its row number and column number must be less than the other Table identified
Table_To_Copy_Into	the number of the Table the values will be written into.

Table 8- 5. Copy Table Parameter Form

The State Logic CPU will generate run time critical errors if the two Tables selected are not the same type or if the Table to copy from is larger than the other Table.

Table Uses

There are many uses for the Table functions. As an example, the Table functions are valuable in applications where the set up of parameters varies depending on the product under manufacture on the process line. Batch process recipes or flexible manufacturing assembly lines are examples.

The ECLiPS program can be written using English name variables for parameters throughout with statements such as:

If Oven_temp_1 is greater than Melting_point ...

used throughout the program description of the process. Then in a State, lets call it the Select_Product State, by using the Swap_Table_Value_Flt function, the variable Melting_Point can be made equal to one of the elements of Table 1, where Table 1 contains the parameters for this particular product run.

Using the Init_Table_Float, Tables containing parameters for each style of product that can be made on the line can be initialized. When the Operator selects a style of product in the Select_Product_Style State, the Copy_Table_To_Table function can be used to move those parameters into Table 1, the Table in which Melting_Point finds its values for this style and product run.

Data Type Conversion

ECLiPS stores information in different Data Types, such as integer, or string. Normally, operations can not be performed across data types, i.e. strings cannot be compared with integers, floating point cannot be assigned to a discrete data type, etc. The one exception to this is that floating point, analog, and integer variables can all be compared and assigned across data types.

In some cases, there is a need to switch information between data types. The "Copy_var_to_integer" function can be used to convert any data type, discrete, analog, floating point, string, or character into an integer or block of integers. The inverse is also possible, the values in a block of integers can be converted into variables of different data types.

Another data conversion function is the Convert_Double_Int perform function which converts between double integer and floating point data types. The double integer is stored in two successive integer variables.

Copy_Var_To_Integer

The Copy_Var_to_Integer function is particularly useful for the construction and decoding of global data blocks. Global data blocks are used to transmit packages of mixed data types over a Genius Network

Information that is transmitted over the Genius Network is transmitted and received in contiguous blocks. Up to 128 bytes of contiguous information can be transmitted or received at each bus address. If the information that is being received or transmitted contains more than one data type, a Global Data block must be used.

The Copy_Var_To_Integer command can be used to construct or decode Global Data Blocks. Variables of different data types can all be copied into contiguous integer data registers using the Copy_Var_to_Integer function. The contiguous integer data block can then be transmitted over the Genius Network.

EXAMPLE Two 9030 State Logic CPUs are communicating using a Genius Network and the Enhanced Genius Communications Module (GCM+). PLC 'A' is transmitting information from a High Speed Counter to PLC 'B'. The information includes all of the data types used by the High Speed Counter (16 %I, 16 %Q and 16 %AI).

The program in PLC 'A" would use the Copy_Var_to_integer function to build a data block that could be sent as Integer over the Genius Network. PLC 'B' would receive the information as 18 Integers and use the Copy_Var_to_Integer function to translate the Integers back into 16 %I, 16 %Q, and 16 %AI variables.

Note

All analog variables are considered to be two words long when using the Copy_Var_to_Int function. Scaled analog variables are 32-bit floating point Unscaled analog variables are integer values stored in the first word of the 2 words reserved for the value.

Step by Step Instructions

The previous example can be used to help illustrate the step by step process of converting data. Selecting the “Add a perform function” option from the ADD menu brings up a list of choices. Selecting Copy_Var_to_Integer brings up a form that asks for the following parameters:

Direction	The direction of the copy; I to copy into the Integer, O to copy out from the Integer
Number_of_Words	The number of consecutive words to copy. Discrete points always translate a minimum of one word of information, starting at the byte boundary. String Variables translate the specified number of words or until a null character is seen.
Variable_Type	There are four discrete data types and five other Data types. The discrete data type codes are: I for %I, Q for %Q, G for % G, and F for Internal Flags. Non Discrete Codes are: A for Analog input, O for Analog Output, P for Floating Point, S for String, and C for Character
First_Integer_Var	The name of the first integer Variable to copy into or out of
First_Var_to_Copy	The name of the first variable to be copied to or from. The data type of this variable must be the same as the data type specified in the Variable_Type parameter above.

Table 8- 6. Copy Variable to Integer Parameter Form

Using the previous example, PLC would need to use this function three times to build the datagram. One function for the discrete inputs %I, one function for the discrete outputs %Q, and one function for the Analog Inputs %AI.

For the sake of this example there are 18 consecutive integers that are not used for any other function. The name of the first integer is Trans_HSC_Inputs, the second integer is named Trans_HSC_outputs, and the third of the 18 integers is called Trans_HSC_Analog1. The first perform function translates the discrete inputs, the second translates the discrete outputs, the third translates the Analogs.

The form for the discrete inputs looks like this:

Direction	I for Into the Integer
Number_of_Words	1 16 bits of data or one Word
Variable_Type	I For Discrete Inputs
First_Integer_Var	Trans_HSC_Inputs
First_Var_to_Copy	Count1StrobeStatus (The name of the first input of the 16)

The code generated by ECLiPS looks like this:

```
perform Copy_Var_to_Integer with
  Direction='I',
  Number_of_words=1,
  Variable_type='I',
  First_Integer=Trans_HSC_Inputs,
  First_var_to_copy=Count1StrobeStatus.
```

A state that would build the entire Data Block looks like this:

```
perform Copy_Var_to_Integer with
  Direction='I',
  Number_of_words=1,
  Variable_type='I',
  First_Integer=Trans_HSC_Inputs,
  First_var_to_copy=Count1StrobeStatus.

perform Copy_Var_to_Integer with
  Direction='I',
  Number_of_words=1,
  Variable_type='Q',
  First_Integer=Trans_HSC_Outputs,
  First_var_to_copy=HSCStrobe1Reset.

perform Copy_Var_to_Integer with
  Direction='I',
  Number_of_words=16 ,
  Variable_type='A',
  First_Integer= Trans_HSC_Analog1,
  First_var_to_copy=HSCModuleStatus.
```

The information would then be configured to transmit across the Genius Network as Integer Data. The receiving PLC 'B' would have the same number of perform functions with the direction parameter changed to O for writing out of the integers, into the correct data types.

Convert Double Integer and Floating Point Data Types

The numeric data types in the current State Logic implementation are integer and floating point. Some external devices such as operator interfaces use double integer values. To accommodate these devices the function Convert_Double_Int converts between double integer data and floating point data. The double integer data is stored in two consecutive integer variables.

This function can be used to convert Floating Point values to Double Integer or Double Integer values to a Floating Point Variable.

The State Logic controller does not display or perform math operations with double integers. Any double integer data must be converted to floating point data to be used in a State Logic program.

The form for this perform function has blanks for the three parameters of the function:

Action	'D' or 'd' to convert from Floating Point to Double Integer 'F' or 'f' to convert from Double Integer to Floating Point
Float_Variable	Name of the Floating Point Variable
Integer_Variable	Name of the first of two consecutive integers used to store Double Integer Value.

Figure 8- 1. Specialized perform functions Parmeter Form

String Manipulation

The String_Manipulation function allows the User to perform various functions upon a string variable. String variables can be up to 80 characters long and often are used for inputting data from an ASCII oriented device such as a bar code reader, or outputting to a similar device such as a scale or robot.

General

In many cases the 80 characters is not one piece of data but a series of sub-strings each containing unique data. Thus the ability to extract the sub-strings, convert the data to integer or float and store in a variable and store a sub-string into a larger string is sometimes needed.

The string manipulation function is in reality several functions in one. The User defines the name of the string to be manipulated and the number of the starting and ending character, if a sub-string within that string is of interest. The start and end must be less than 80 , the first character is number 1, and the start must be smaller than the end or a run time error is generated.

The User also defines the type of manipulation (see following sections) and a reference variable that is used by the operation. The parameters defined by the User for the function are:

String_name	the name of the string to manipulate
Start_character_num	the number of the starting character used in the string. The first character is 1
End_character_num	the number of the ending character used in the string. The last character is 80.
Operation	a character code that defines the operation to be performed (see following section for definition)
Reference_variable	the name of the reference variable to be used in this manipulation. The variable type required depends on the Operation and the State Logic CPU generates a critical error if a mismatch occurs.
Search_Character	the name of a Character_Variable or the actual character to be matched by this operation. This is an optional parameter which only needs to be entered when the match (M) operation is chosen.

Table 8- 7. String Manipulation Parameter Form

Operations

E for Extract

If the Operation character is an E, the function will extract the sub-string defined by the starting and ending characters and copy those ASCII character numbers to the string variable named in the Reference_variable.

s For store in sub-string

If the Operation character is an 's', the function will use the string variable named in the Reference_variable as a sub-string, and store those ASCII characters in the sub-string defined by the starting and ending character numbers. The characters in the string named as the Reference_value will be stored until the end of that string is reached or until the last character number in the main string is reached.

I for extract and convert to Integer

If the Operation character is an I, the function will extract the sub-string defined by the starting and ending character numbers and convert those ASCII characters into an integer value and store that value at the integer variable named in the Reference_variable.

i For Convert int eger To ASCII And Store In String

If the Operation character is an ‘I’, the function will convert the value stored at the integer variable named in the Reference_variable to ASCII and store that value into the sub-string location defined by the starting and ending character numbers.

F for extract and convert to Float

If the Operation character is a F, the function will extract the sub-string defined by the starting and ending character numbers and convert those ASCII characters into a float value and store that value at the float variable named in the Reference_variable.

f For Convert float To ASCII And Store In String

If the Operation character is a f, the function converts the value stored at the float variable named in the Reference_variable to ASCII and stores that value into the sub-string location defined by the starting and ending character numbers.

C for Concatenate

If the Operation character is a C, the function concatenates or adds the string of characters named in Reference_variable to the main string. The resulting main string can not exceed 80 characters, the addition of the Reference_variable characters will be truncated at 80 characters.

L for string Length

If the Operation character is an L, the function calculates the number of characters in the string, and puts that number into the integer variable named by Reference_variable.

M for Match the Given Character with a Character in the String

If the operation character is an M, the function matches the character in the Search_Character parameter to the first character in the sub-string designated by starting character and ending character values. The position of the first match is returned in the Reference_Variable.

Errors in General

The functions check to make sure when conversions to or from ASCII are performed that legal values result and that errors are produced if they do not.

Communications Request

The Comm_Request perform function is used by the State Logic program to send commands to the option modules installed in the system. There are several types of communication request functions that can be executed for most of the option modules. *See the user's manual about the specific module for more information about using these communication request commands.*

There are Comm_Request program examples in the Option Module chapter of this manual.

The parameters for the function are described in the following table:

Rack	The variable or constant specifying the rack number for the module receiving the command.
Slot	The variable or constant specifying the slot number for the module receiving the command.
Task_ID	The variable or constant specifying the task number of the operation called for in the command. *See the module manual for more information.*
Data_Block	The variable name of the first of several consecutive integer variables that contain the data and command information for this function call.

Note

There are some limitations in using the Comm_Request perform function. Only the NOWAIT option is supported. The WAIT option is not supported. The only data type supported for Comm_Request data stored in the CPU is register, type 8. This data is stored in integer variable locations. This type refers to status information, data to be transmitted to a module from the CPU, and data received by the CPU from the module.

Time Counter

This time function is designed to keep track of the elapsed time of an event. The time is stored in named integer variables that can be examined the same way that any other integer variable is examined. There are four type of time counters, each one being incremented at a different time period (tenth of a second, second, minute, and hour). There may be up to 30 each of tenth of a second, second, and minute counters and 10 hour counters active at any time during program execution.

The Time_Counter form that ECLiPS displays when this perform function is selected, has three parameters that define the function call:

Action Specifies how the function is applied to the counter. The data type of this parameter is character, and it may be specified by a variable or as a constant. A description of the possible choices for this parameter follows:

'A' - Assigns the specified variable to be a counter that is incremented as specified in the Time_Interval parameter. The variable is incremented only when it is enabled.

'E' - Enables a defined counter to begin counting the time interval specified when the counter is 'Assigned'.

Note

When the program starts automatically after a power cycle, the time counters that were enabled are no longer enabled. Make sure to create the program so that all time counters get re-enabled after a power cycle.

'H' - Halts the counter from being incremented. The variable storing the time count maintains its value. To start counting again this counter must be 'Enabled'.

'D' - Deallocates the counter. The variable is no longer a counter freeing up space for one more counter of that type to be 'Assigned'.

Integer_Name Specifies the integer variable name to be used as a time counter. The named must be defined as an integer variable separately.

Time_Interval Specifies what time period must pass before incrementing the counter. The data type of this parameter is character and is specified by either a variable or a constant. This parameter is required only if the action parameter is an 'A'(Assign). For other actions this parameter may be left blank. A description of the possible choices for this parameter follows:

'T' - Specifies the counter to be incremented every tenth of a second.

'S' - Specifies the counter to be incremented every second.

'M' - Specifies the counter to be incremented every minute.

'H' - Specifies the counter to be incremented every hour.

The procedure to use a time counter is a two step process:

First, the time counter must be assigned to an Integer Variable. By using action = A and giving an appropriate Time_Interval. The Integer Variable is now prepared to function as a time Counter.

The second step is to start the time counter using the Enable action (Action in the perform function is E). The Time_Interval parameter is ignored when an enable time counter perform function is performed.

Note

The maximum value for each of these counters is 32767. After reaching the maximum, the counter continues counting, first becoming a negative value then returning to 0 and back positive again. If your counter goes over the maximum value, use the next higher counter type, for example use a minute counter instead of a second counter.

Another option is to monitor the counter variable in the program. The program can reset the variable to 0 when the maximum value is approached.

High Speed Counter Data Transfer function

The Send_HSC_Data function transfers data to the High Speed Counter from the State Logic controller. The High Speed Counter data that can be changed are the Accumulator Value, Hi Limit, Low Limit, Timebase, Preset, Preload, and Oscillator Frequency Divisor. This function provides the same functionality as a similar COMREQ in ladder logic systems.

The parameters for this function are:

- **Rack_Number** - Specifies the Rack Number where the HSC is Installed (0 - 4)
- **Slot_Number** - Specifies the Slot Number of the HSC (1 - 10)
- **Remote_Rack** - Indicates Whether this Rack is a Remote Rack (Y or N)
- **First_Variable** - Specifies the Name of the First of Three Consecutive Variables that Store the Command Data Sent to the HSC Module

The first variable of the three sent to the HSC specifies the operation (Least Significant Byte) and the ID number of the counter (Most Significant Byte) affected by this command. The other two variables contain the actual data used by the HSC. The following table specifies the definition of the data stored in the three variables:

	MSB	**LSB**
Command Word	0x	cc
Data Word (LSW)	dd	dd
Data Word (MSW)	dd	dd

Table 8-8. HSC Command and Data Format (x = Counter Number, cc = Command Code, dd = data, always 0000 for Type A Counters)

The Command Word can specify many different operations. The following table lists all of the operations codes available with this function.

DECIMAL VALUE	HEXADECIMAL VALUE	OPERATION	COUNTER TYPE
0x 01	0x 01	Load Accumulator	All
0x 02	0x 02	Load High Limit	All
0x 03	0x 03	Load Low Limit	All
0x 04	0x 04	Load Accumulator Increment	All
0x 05	0x 05	Set Counter Direction	A
0x 06	0x 06	Load Timebase	All
0x 08	0x 08	Load Home Position	C
0x 11	0x 0B	Load ON Preset	All
0x 12	0x 0C	Load Second ON Preset	B,C
0x 13	0x 0D	Load Third ON Preset	C
0x 14	0x 0E	Load Fourth ON Preset	C
0x 21	0x 15	Load OFF Preset	All
0x 22	0x 16	Load Second OFF Preset	B,C
0x 23	0x 17	Load Third OFF Preset	C
0x 24	0x 18	Load Fourth OFF Preset	C
0x 31	0x 1F	Load Preload	ALL
0x 32	0x 20	Load Second Preload	C
00 50	00 32	Load Oscillator Divider Ratio	All

Table 8-9. Operation Codes, where x = Counter Number

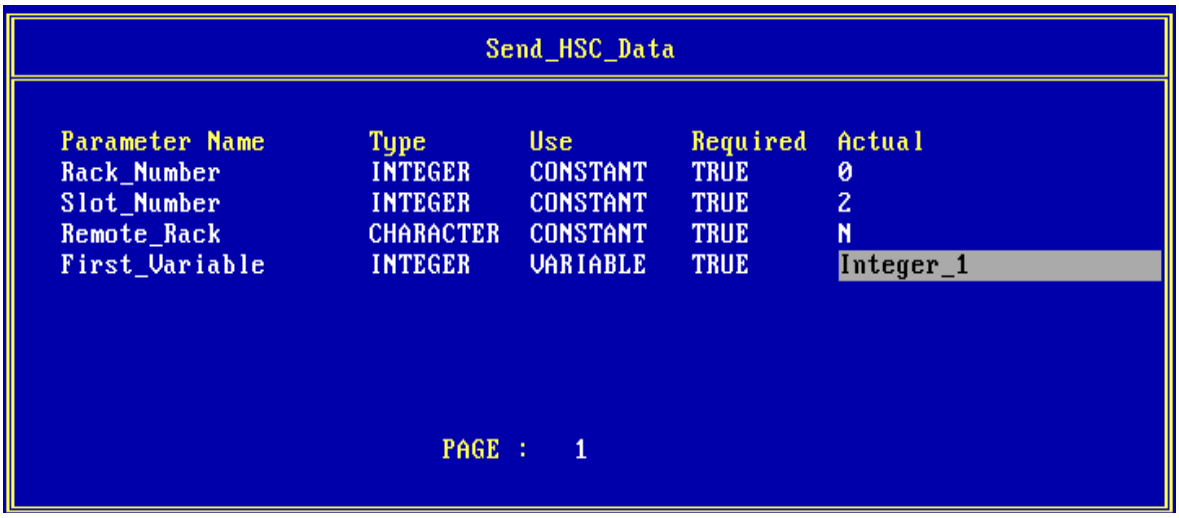
EXAMPLE 1: The HSC module is in local rack 0, slot 2. To set counter 3 output to turn on at 5000 (#1388) counts, set the first variable equal to #030B and the second variable equal to #1388 executing the following statements in the State Logic program.

Make Integer_1 = #030B.

Make Integer_2 = 5000. !5000 could also be expressed as #1388

Make Integer_3 = 0.

Next use the Send_HSC_Data perform function to send the information to the HSC Module. Create the function in the program by filling in the Send_HSC_Data form as shown below.



When completed, this form produces the following text in the State Logic program.

```
perform Send_HSC_Data with
  Rack_Number=0,
  Slot_Number=2,
  Remote_Rack='N',
  First_Variable=Integer_1.
```

Example 2: The HSC Module is in the remote rack 3, slot 8. To make the Type C counter start at 2500000 (#2615A0) at its preload 2 signal, use the following statements.

```
Make Integer_1 = #0120.
Make Integer_2 = #15A0.   !Could also use the 9532 decimal value
Make Integer_3 = #0026.   !Could also use the 38 decimal value
```

Next, execute the following perform function:

```
perform Send_HSC_Data with
  Rack_Number=3,
  Slot_Number=8,
  Remote_Rack='Y',
  First_Variable=Integer_1.
```

For values requiring more than one word, consider using the Convert_Double_Int perform function which converts floating point variable value into a double integer stored in two consecutive integer variables.

For more details on the High Speed Counter module see the *Series 90-30 High Speed Counter User's Manual, GFK-0293.*

Shift Register

The Shift_Register function allows the User to shift values from one integer variable to another by a user selected number of bits.

The User can define up to 28 integer variables and connect them together to form a shift register. The User then can shift the contents of each variable by a selected number of bits (up to 64) to the right or left.

The User can choose to have the shift behave in a circular fashion where the bits from the variable farthest to the right shift to the farthest variable to the left. Or the shift can be a fill type shift where the bits fall off the end and do not circle back to the first integer. In the case of a fill type shift, the User can choose the value (0 or 1) that is placed in the locations left empty by the shift.

One method of retrieving information from the shift register variables is to use the **Test_Bit** keyword.

The User defined function parameters are:

Number_of_bits	The number of bits to shift the registers, 0 to 63
Shift_direction	Either R right or L for left. The direction of the shift.
Type_of_shift	Either C for circular or F for fill, The way the shift will act when the last variable is reached.
Fill_value	Either 1 or 0. The value that will be placed in the bits of the variables that have been shifter in a F or fill type shift. No meaning in a circular shift.
Variable_1	the name of the first integer variable and therefore the variable farthest to the left in the shift register.
Variable_2	the name of the second integer variable in the shift register.
. . .	
. . .	
Variable_28	the name of the 28th or last integer variable and therefore the variable farthest to the right in the shift register.

Table 8- 10. Shift Register Parameter Form

BCD I/O Representation

At times input and output devices are used for data entry or display that use BCD representation. Thumb wheel switches and LCD displays are possible examples.

The devices are connected either to digital inputs or digital outputs where 4 hardware inputs or outputs represent 1 digit of the display. The display or switch then uses a binary code representation from 0 to 9. There are 16 total possibilities (4 outputs or inputs represent 2 to the 4th or 16 possible combinations), and the remaining 6 are used for such things as minus sign, decimal point and null or space character on a display.

The two functions, BCD_In_Conversion and Output_BCD_Conversion allow the User to designate a series of consecutive digital inputs or outputs to be treated as if they are groups of 4 BCD digits. The functions then translate between the I/O and either State Logic CPU integer or float variables.

BCD_In_Convert

The BCD_In_Convert translates between a series of digital inputs and an integer or float variable. The User specifies the name of the first digital input in the series and the number of BCD digits (4 inputs per digit) that represent the variable. The User also specifies the type of variable and the name of the variable to store the converted value.

The inputs are taken in hardware consecutive order, and the number of digits can be on more than one block or card as long as the cards have consecutive addresses. The User only need define with an English name the first digital input and initialize the blocks or cards involved.

BCD uses the standard binary representation for the numerical digits 0 to 9. There is no true standard for the minus sign or decimal point character. Therefore the function has the provision for the User to optionally specify the hexadecimal number, #A, #B, #C, #D, #E, or #F, (where # means hexadecimal number to ECLiPS), that is the pattern for these two characters.

The function parameters are:

Starting_input	the English name of the first digital input in the string of consecutive inputs that form the BCD digits
Number_of_BCD_digits	the number of BCD digits. The number of digital inputs in the string will be 4 times the number of digits.
Variable_name	the name of the variable to store the translated value
Variable_type	the type of variable either integer or float
Minus_sign_pattern	a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the minus sign.
Decimal_point_pat	a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the decimal point. Note the minus sign and decimal point are optional. A minus sign can be specified without a decimal point, but if a decimal point is specified the minus sign must also be specified.

Table 8- 11. BCD Input Conversion Parameters

Output_BCD_Convert

The Output_BCD_Convert translates between a an integer or float variable and a series of digital outputs. The User specifies the name of the first digital output in the series and the number of BCD digits (4 outputs per digit) that represent the variable. The User also specifies the type of variable and the name of the variable where the value to be converted is stored.

The outputs are in hardware consecutive order, and the number of digits can be on more than one block or card as long as the cards have consecutive addresses. The User only need define with an English name the first digital output and initialize the blocks or cards involved.

BCD uses the standard binary representation for the numerical digits 0 to 9. There is no true standard for the minus sign or decimal point or null (space) character. Therefore the function has the provision for the User to optionally specify the hexadecimal number, #A, #B, #C, #D, #E, or #F, (where # means hexadecimal number to ECLiPS), that is the pattern for these three characters.

The function parameters are:

Starting_output	the English name of the first digital output in the string of consecutive outputs that form the BCD digits - all of the other outputs used must be defined
Number_of_BCD_digits	the number of BCD digits. The number of digital outputs in the string will be 4 times the number of digits.
Variable_name	the name of the variable that is to be translated and output
Variable_type	the type of variable either I for integer or F for float
Minus_sign_pattern	a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the minus sign.
Null_character_pat	a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the null or space character.
Decimal_point_pat	a hexadecimal number from A to F that gives the pattern for a digit that should be recognized as the decimal point.
Number_decimal_dig	the number of decimal digits; the digits to the right of the decimal point, that should be output. Note: If the Number_decimal_dig parameter is used then the Decimal_point_pat is not optional and must also be used.

Table 8- 12. Output BCD Conversion Parameter Form

Note: if the value is too large to display in the Number_of_BCD_digits specified, but there is enough room for all the digits to the left of the decimal point, those digits will be displayed and no error will be generated. If there is not enough digits for all the numbers to the left of the decimal point, the display will not be output and a non-critical error will be generated.

Specialized perform functions

All off the above functions have specific parameters which are passed to the function. These parameters are all chosen by filling in similar forms which specify parameter type and whether or not it is required. The following performs each have unique ways that the operations are specified. All of the Specialty perform functions must be the only Statement in the State.

Display Date and Time

This function displays the current date and time in the desired format. After choosing this option a form is displayed to enter the Name of the State that is created, the format of the display, the

communications port to send the information and the State to branch to after the operation is completed.

Get User Input

This function enters program text used to retrieve information through a communications port. A form is displayed for entering the following options:

Current State	Name of the State that is created.
Clear Screen	Option of clearing the screen before the prompt is displayed.
Screen Message	Prompt telling operator to enter some information.
Input Variable	Variable that stores the input
Comm Port	Which port is used.
Branch To State	State that becomes active when this process is completed.

Table 8- 13. Get User Input Parameter Form

User Menu

This function displays a menu of up to 10 items and then waits for the user to enter a selection. If the selection is valid, it will branch to desired State for that selection. This is very useful when creating a user interface for the control program.

Chapter 9 PID Loops

The Series 90-30 State Logic CPU controller provides the capabilities of modulating control through the use of the PID algorithm. The Series 90-30 State Logic CPU provides twenty PID algorithms that are continuously executed at user selected time intervals of 1 to 65535 seconds. These PID algorithms can be connected to field inputs and outputs or internal floating point variables, or interconnected in cascaded and other fashions to implement the desired control strategy.

The memory for all twenty PID loops is reserved for PID loops only. Therefore the number of PID loops defined in the system has no effect on available memory for program or data storage.

A PID loop is created by completing the PID configuration form. This form is accessed by selecting the “PID Loops and Initial Values” option from the LIST menu, or the “PID Loop Initial Values” option from the DEFINE menu.

This chapter describes the PID algorithm and its implementation in this product. Seting up the initial PID loop tuning constants and starting the loop running are explained. There are also sections describing how to tune the loop from both the Debug Mode or OnTOP tuning screen and by program statements.

PID Algorithm and Philosophy

The Series 90-30 State Logic CPU employs a traditional PID algorithm that compares a setpoint with a process variable to generate an error signal. The error signal is acted upon by any or all of three parts; proportional, integral, or derivative, to generate an output value, the control variable.

Each of the three parts has a tuning constant associated with it that can be adjusted to affect how the control action occurs. The Proportional term uses the Gain tuning constant. The proportional result is simply the product of the error, the difference between the process input and the setpoint, multiplied by the Gain. It is an instantaneous value that changes as the error changes.

The Integral term uses the Reset tuning constant. Its result is an accumulation of the error signal, times the Gain, divided by the Reset. Even though an error signal is currently zero value, the integral portion may provide a result because previous error signals have accumulated.

The Derivative term uses the Rate tuning constant. The derivative term is is the Rate times the rate of error change. The amount of the Derivative term output for a given error is affected by the value of the Rate tuning constant.

The total output of the PID is the sum of the results of the three terms plus the Bias value. Figure 1 shows a simplified diagram of the algorithm. Typically the Proportional and Integral terms are used more often alone without Derivative because this provides a more stable control performance. The Derivative term allows more anticipation and quick response, but at a penalty of possible over response and undesirable process disturbances.

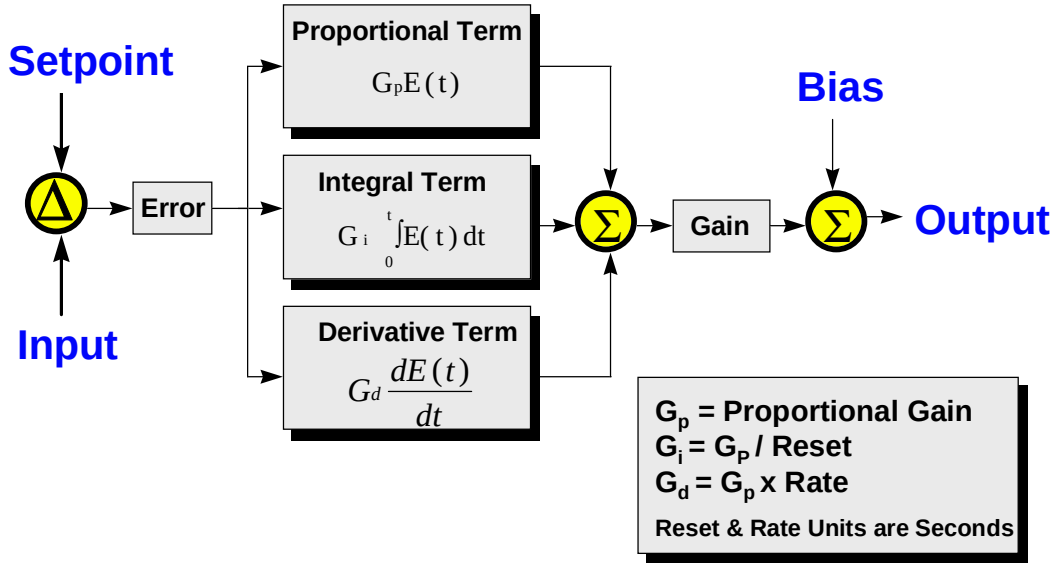


Figure 9- 1. PID Algorithm

When the process variable differs from the setpoint, such as at the time of a step change in the setpoint, the proportional term immediately causes the output of the controller to change. As time passes the integral term integrates the controller in the same direction. The action of the controller hopefully brings the process variable closer to the setpoint. This causes the error to become

smaller and decreases the proportional term, but the integral term continues to increase as it adds on the error signal over time.

Ultimately the process variable may equal the setpoint and the error is zero causing the proportional term's value to be zero. In addition, the integral portion is no longer changing because the error is zero, therefore the controller output remains constant equal to the value the integral term accumulated. Any changes in process variable or setpoint cause an error and the controller will integrate to adjust the output to bring the system to equilibrium.

The addition of the derivative term, makes the controller react more extremely when the error is first detected. Then as a function of the Rate tuning constant, this reaction decays out allowing the integral term to bring the system into balance and remove the error.

Complex PID Control

In some cases a complex strategy using PID algorithms may be desired. Rather than the output of the PID going directly to an analog output, a cascaded PID strategy can be used in which the output of one PID goes to the input of another PID.

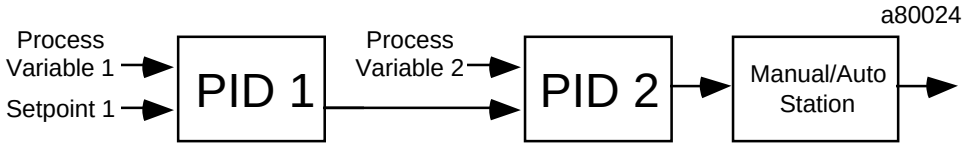


Figure 9- 2. Cascaded PID Loops

In this example PID 1 compares process variable PV1 with the desired setpoint. The output of that controller is then fed into PID 2 as a setpoint and is compared with process variable PV2. Generally the second PID (PID 2) called the downstream PID, will be tuned to have a faster response time. It will act first to move the controller quickly in the right direction, and then the slower acting upstream PID will act to integrate out the error between the control variable and its desired setpoint.

The PID algorithm can use an analog input as its input, as in the case of PID 1 or any floating point variable. In the case of PID 2, the input to PID 2 can be defined as the output of PID 1. Likewise the setpoint can be a constant, or a named variable or as in this case, an analog input. To set this strategy just use the "PID Define" menu and name the input and setpoint as described in Section .

Bumpless Transfer

The drawing in Figure9-2 also shows a Manual/Auto station between the PID output and the actual analog output card. This station allows the User to place the station in Manual where the output is changed directly. In the Auto mode, the output is controlled by the PID loop.

When the M/A station is in Manual, the station value can and usually will be forced to a value other than one that will make the setpoint equal the process variable input. If the PID followed its normal operation, the integral term would continue to integrate because of the error signal between the process variable and setpoint. This would leave a difference between the PID output,

which is the Auto input to the M/A station, and the actual M/A output. Then when Auto mode is selected, this difference would cause a jump or bump in the M/A output. This would upset the process and is desirable to avoid.

To prevent this bump from happening, the PID needs to have another mode of operation besides its automatic mode. In this mode, called tracking, the PID output will be maintained at what ever value it is set to, such as the M/A output in this example. The PID will not perform its normal arithmetic, but will instead set itself so that when tracking is removed, the PID output gradually goes to the proper value and avoids the bump. This is called bumpless transfer.

Each of the twenty PID algorithms have logical signals associated with them that use the name Track_Mode and the actual PID name. As an example, if the User named the first PID algorithm Tank_Level then the logical signal would be called Tank_Level Track_Mode. When Track_Mode is made true, then the PID automatically discontinues its normal algorithm, and begins tracking its output and preparing for bumpless transfer.

The User can set the Track_Mode variable from any active State. Using this variable, the User can create any tracking strategy he desires.

Anti-Reset Windup

Using a cascaded PID strategy can cause the User some subtle problems. In the example shown in Figure 2, if PID 2 has reached its maximum and PID 1 still has an error signal because the setpoint does not equal PV1, then PID 1 would continue to integrate. The output of PID 1 would continue to increase, but it would have no affect on PID 2 since it is already at its maximum. However when the error signal of PID 1 reversed direction and caused PID 1 to begin integrating in the opposite direction, PID 1 would have to integrate below the threshold value it was when PID 2 reached its maximum before it would have any affect on PID 2. This excess amount of output PID 1 has accumulated is commonly referred to as reset windup.

The upstream controller, PID 1, needs to be prevented from winding up. An input called Block_Up will transfer the PID algorithm into a mode such that it will not integrate in the Up, towards 100%, direction. This is called anti-reset windup.

The PID will still be able to integrate down if the error signal reverses direction. That is, if the process variable is less than the setpoint the PID will not integrate up. If the process variable becomes greater than the setpoint the PID will integrate down. The Block_Down input works exactly opposite.

In the case of Figure 9-2, if the setpoint for PID 1 is greater than PV1, PID 1 output will continually increase. This is the setpoint for PID 2 and assume it is already greater than PV2 and PID 2 has reached its high limit. There is no profit in PID 1 output getting larger since PID 2 can not respond to its demand, PID 2 is already at its limit. Therefore the User should set the PID_1 Block_Up input true and stop the PID from winding up further.

Then when either PV2 increases or the high limit on PID 2 is changed which will allow further action by PID 2, the PID_1 Block_Up input can be set false and PID 1 can resume integrating. Or if PV1 rises above the setpoint in response to the control action, PID 1 will begin integrating the output of PID 1 in the lower direction which is permissible. When PID 1 output falls below the input PV2 into the downstream PID 2, PID 2 will integrate down below its high limit and remove PID_1 Block_Up and return the complete loop to normal.

Using the Block_Up and Block_Down inputs, any amount of cascading of PID's can be accomplished without reset windup occurring and with bumpless transfers from one mode to another.

The anti-reset windup features are also invoked at the limits. When the integral term reaches one of the limits, the integral term stops winding up. The anti-reset features are not invoked when the control variable reaches the limit, only when the integral term itself reaches the limit.

Initializing and Starting PID Loops

PID loop is created by filling in the PID Loop Initialization Form. The actual starting and stopping of the loop is controlled by statements in the State Logic program.

Starting PID Loop Execution

The PID Loop is started in the program when the Start_PID keyword is executed during normal program sweep. For example the statement:

```
Start_PID Tank_Level.
```

starts the PID Loop named Tank_Level.

Note

If the a Start_PID statement is executed for a loop that has already been started, the loop is reinitialized and calculations restarted. Usually this result is not desired. Since State Logic control is a scanning system, the program should be constructed to immediately **GO** to another State once the Start_PID instruction is executed.

To stop a PID Loop use the Stop_PID keyword. For example the Statement:

```
Stop_PID Tank_Level with 10.
```

stops the PID Loop named Tank_Level and sets the contorl variable to 10. The ending **WITH** followed by a value is optional. If the **WITH** is omitted, the value of the control variable is not changed when the loop stops executing.

Once the loop is executing, it continues its operation until stopped by the program. The loop operates in the background, independent of which is the current active State.

PID Initialization Form

Each PID is created by filling in the PID Loop Configuration form accessed through the LIST or DEFINE menus. When the form is completed, press <F9> to save the entries or <Page Up> or <Page Down> to work with other loop settings.

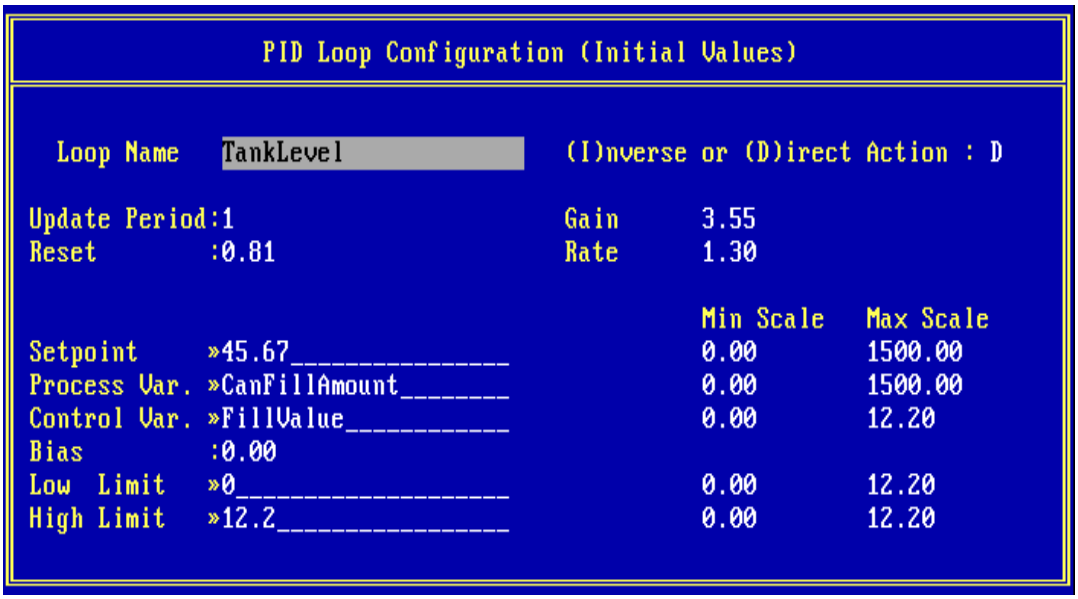


Figure 9- 3. PID Loop Initialization Form

Loop Name - Any legal State Logic name

Inverse or Direct Action - Press any key to toggle between 'I' and 'D'. Direct Action means that when the Setpoint is greater than the Process Variable the PID calculation usually increases the control variable to correct this error. For the Inverse Action, the same conditions result in the opposite response.

Update Period - The number of seconds between loop calculations. Allowable values are from 1 - 65535.

Gain - Parameter controlling the proportion error response.

Reset - Parameter controlling the integral error response. Notice from the algorithm description that this parameter is in the denominator of the integral component of the error response. Therefore, a larger **Reset** value results in a smaller integral response.

Rate - Parameter controlling the derivative error response.

Setpoint - Constant, floating point variable name, or scaled analog channel name which specifies the value that the loop attempts to achieve for the process variable.

Process Variable - Floating point variable or scaled analog channel name. The loop attempts to make this value equal to the setpoint.

Control Variable - Floating point variable or scaled analog channel name. The loop attempts to move the process variable value closer to the control setpoint by changing the value of the control variable. The loop uses the error (difference between process variable and setpoint) to decide how to change the control variable.

Bias - The constant added to the loop calculation for the control variable. Uses the control variable scaling factors.

Low Limit / High Limit - The minimum and maximum values allowed for the control variable. The loop does not force the control variable beyond these limits. Anti-reset windup features are invoked when the integral term not the control variable reaches the limit.

Min Scale / Max Scale - Within the PID algorithm, input, output, and limit values are converted to a percentage for the calculations. The **Min Scale** and **Max Scale** scaling constants for each value are used for the conversion to a percentage. Values at the **Min Scale** value are 0% for the calculation and values at the **Max Scale** value are 100% for the calculation. Other values are converted to the proper percentage using these scaling values.

The Bias parameter uses the same scaling factors as the Control Variable. For example if the Control Variable uses -200 to +200 for scaling factors, a Bias value of 0 would actually add 50 per cent to the Control Variable.

For example, if Min Scale equals 20 and Max Scale equals 30 for the setpoint and for the process variable, Min Scale equals 0 and Max Scale equals 200. If the setpoint is 25 and the process variable is 100, both setpoint and process variable values are at 50 per cent and therefore there is no current error for this loop.

The default scaling is set to 0 to 100, which treats the values as already being converted to percentages.

Note

The '»' character next to the blank in the form indicates that pressing the <Insert> key causes a list of floating point variable and analog channel names to be displayed. The desired name for the option can be selected from this list.

Loop Tuning and Monitoring

The PID algorithm has several adjustable tuning parameters and status variables. These values include those used to initialize the loop plus several control and status bits.

The PID loop data can be directly monitored and adjusted (tuned) while in operation by using any of the following:

- PID Tuning Screen available in Debug Mode or OnTOP
- State Logic Program Instructions
- External Devices using SNP or CCM Serial Ports or the Ethernet Module

Each of these methods are examined after the parameters and command and status bits are explained. The PID parameters available for tuning are listed in the following table. The keyword used in the State Logic program and a description of the parameter are also listed in the table.

Parameter	Register Number	Keyword	Description
Action Direct or Inverse	N/A	Loop_Action	Use this keyword to change or monitor whether the loop is **Direct** (increases control variable when process variable is less than the setpoint) or **Inverse** (decreases control variable when process variable is less than the setpoint) acting. A value of 0 indicates direct and 1 indicates inverse action.
Status Word	1	Status	Word of PID status information: Bit 0 - Blocked Up Bit 1 - Blocked Down Bit2 - Track Mode Bit 3 - Loop Action Status - 0 for Direct, 1 for Inverse Bit 8 - Low Limit Status (Read Only) Bit 9 - High Limit Status (Read Only)
Update Time	2	Update	Time interval in seconds between updates for this loop
Gain	3	Gain	Proportional parameter for the loop algorithm
Reset	4	Reset	Integral parameter for this loop algorithm (1/sec)
Rate	5	Rate	Derivative parameter for this loop algorithm
Setpoint	6	Setpoint	Floating point constant, variable name, or scaled analog channel name of the target value for the process variable
Setpoint Min. Scale	7	SP_Min	Value for 0% scale
Setpoint Max. Scale	8	SP_Max	Value for 100% scale
Process Variable	9	Process_Var	Floating point variable or scaled analog channel name of the value that is tested against the setpoint for error conditions
Process Var. Min. Scale	10	PV_Min	Value for 0% scale
Process Var. Max. Scale	11	PV_Max	Value for 100% scale
Control Variable	12	Control_Var	The output of the PID loop
Control Var. Min. Scale	13	CV_Min	Value for 0% scale
Control Var. Max. Scale	14	CV_Max	Value for 100% scale
Bias	15	Bias	Amount to be added to the Output. Uses the Control Variable scaling
Low Limit	16	Low_Limit	Minimum allowable value for the Output.
Low Limit Min. Scale	17	LL_Min	Value for 0% scale
Low Limit Max. Scale	18	LL_Max	Value for 100% scale
High Limit	19	High_Limit	Maximum allowable value for the Output
High Limit Max. Scale	20	HL_Max	Value for 100% scale
High Limit Min. Scale	21	HL_Min	Value for 0% scale

Table 9- 1. PID Loop Parameters

All of the parameters in this table are floating point data except the Status Word which is integer data. There are also digital keywords providing control and monitoring PID loops. The following table lists the status and command bits:

Status or Command Bit	Keyword	Description
Block Up Control	Block_Up	When this bit is set, the PID does increase the control variable. Anti_Reset Windup features in effect.
Block Down Control	Block_Down	When this bit is set, the PID does not decrease the control variable. Anti_Reset Windup features in effect.
Track Mode Control	Track_Mode	When this bit is set, the PID tracks the output so that no error is calculated. Used for a bumpless transfer to automatic mode.
High Limit	High_Limit_Status	When this bit is set, the PID output has reached the high limit parameter or Block_Up for this PID is true. This is a read-only bit.
Low Limit	Low_Limit_Status	When this bit is set, the PID output has reached the low limit parameter value or Block_Down is true for this PID loop. This is a read-only bit.

Table 9- 2. PID Command and Status Bits

Loop Tuning Form

The loop tuning form available in both Debug Mode and OnTOP is accessed from the main menu displayed when <F3> is pressed. This menu option is displayed only if there are loops defined in the program.

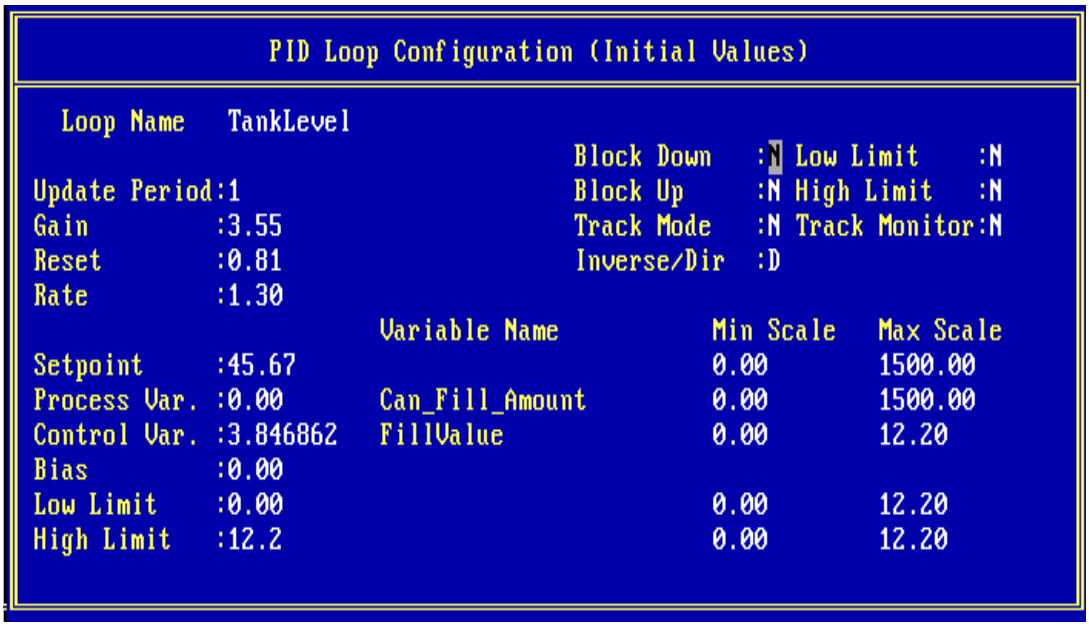


Figure 9- 4. PID Tuning Screen

This form displays the values of several parameters, scaling values, and status bits. The most common parameter values to be changed using this form are gain, reset, and rate. These parameters are adjusted to obtain the desired PID performance, such as speed of response and over shoot.

In addition to the gain, reset, and rate, each PID has a high limit and low limit that can be set to limit the output to less than its full range. These limits automatically employ anti-reset windup so that the loop does not continue to accumulate error when the loop is limited. In addition the setpoint and process variable values can be adjusted from the tuning form.

Once the loop has been tuned, pressing <F9> allows the changes to be made permanent for that loop. If this option is selected, the loop's initial values are changed in the program so that the next time the program is downloaded the loop starts executing with the newly tuned values.

Note

The saved tuning changes do not have any affect on the loop initial values until the program is reloaded. If the PID loop is halted and then restarted, the new initial parameters are not used when the PID loop is restarted. To make these tuned values permanent, redownload the program.

The Block Up, Block Down, and Track Mode status bits are also changed using this form. The Block Up and Block Down settings stop the loop from raising or lowering the control variable when set to 'Y'. The Track Mode control prevents the loop from changing the control variable and also from accumulating error over time. The Track Mode is most often used when manually controlling the loop output.

This form also displays whether the loop is at the high or low limits. Also displayed is whether the loop is currently in Track Mode (Track Monitor).

Program Control of PID Loops

As previously described, the PID loop is started in operation by the execution of State Logic program statements. All of the parameters listed that can be adjusted from the tuning form can also be changed from program statements. There are also control and status bits used in the program to either sense a condition of the loop or control its operation.

Using the PID Parameters in a State Logic Program

The program refers to the loop parameters by specifying the name of the PID loop followed by the parameter keyword. For example, the State Logic program statement:

```
Make Tank_Level Setpoint = 45.
```

sets the setpoint of PID loop named Tank_Level to 45.

Any of the parameters including Gain, Reset, and Rate can be adjusted by the program. This capability allows for flexible control to tune the loop automatically with program instructions.

Using the Command and Status Bits in a State Logic Program

The PID status and command bits associated with each PID loop are accessed using the keywords for those bits. These bits indicate information about the status of the PID loop or may be used to control the PID loop. To use these bits in a State Logic program, the PID name is used followed by the keyword for the bit.

The State Logic statement:

```
If Tank_Level High_Limit_Status is true, go to Manual_Mode State.
```

makes Manual_Mode the active State of the Task if the output of Tank_Level PID loop is at the high limit value.

The State Logic statement:

```
Set_Bit Tank_Level Track_Mode.
```

makes the Tank_Level PID loop track the output so that there is no error signal.

When in track mode, the PID loop no longer controls the output. This condition is often used for manual control of the loop output. To place the output back under control of the PID loop (automatic control), clear the Track_Mode bit.

Because the loop does not accumulate any error while in track mode, there is no surge in the output when the PID loop regains control. The transfer is said to be “bumpless”. This control bit is used with a Manual/Automatic station to obtain Manual/Auto bumpless transfer.

The other two inputs, PID_Block_Up and PID_Block_Down limit the PID loop to integrating in only one direction and tracking in the other direction. For example, when Block_Up is true, the PID is allowed to decrease the control variable, but the loop cannot increase the control variable. Anti-reset windup features are in effect so that no error is accumulated when the loop is blocked. Block_Down works exactly opposite.

These two inputs can be used to provide Anti-Reset Windup when one PID is cascaded into another PID. If during the course of operation the down stream PID reaches the point it can no longer integrate in response to the upstream PID's output, as when it reaches full scale, then any further integrating of the upstream PID would be counter productive. These inputs can stop that excess integration in the counter productive direction while allowing immediate response in the opposite direction which is the direction that will have an affect on the down stream PID.

Accessing PID Parameters from a Remote Device

The PID data is available to remote devices communicating through the Ethernet module or an SNP or CCM serial port. The PID data is referenced as specific register locations for this remote access.

Use the “SNP Protocol Listing” or “CCM Protocol Listing” options on the PROJECT PRINT form to get the base register number for each PID loop. Each printout uses the same register locations for PID data. The register location for the data desired is located in the 42 consecutive registers starting with the base register listed. See the PID Parameter table for the parameter register locations.

The first parameter, Status, is the only parameter that is not a floating point value even though two registers are reserved for this parameter. This parameter contains bit information about the

PID loop operation. The status information is contained in the bits as described in the table. The rest of the parameters are all 2 register floating point data. Make sure the receiving device supports IEEE 32-bit format to access this data.

See the chapter on serial communications for more information about using SNP and CCM register numbers to access PID parameter data.

Chapter 10 Online - Debug Mode and OnTOP

This chapter describes the features that are used to interact with the controller running the State Logic program -- the "online" features. The online features are available in the ECLiPS Debug Mode and in OnTOP.

The Debug Mode and OnTOP are used mainly as debugging and troubleshooting tools. Messages from the program are also displayed in the main screen providing a handy operator interface to the controller.

Online Functions Include:

1. **View or Change the Current State of a Task**
2. **Display a History of State Changes**
3. **View or Force I/O Points**
4. **Display or Download Control Program**
5. **Inspect and Clear I/O Hardware Faults Log**
6. **Log Data to a Printer or to a Disk File**
7. **Initiate CCM Communications**
8. **Change Simulation Mode Status**
9. **Display and Change Clock/Calendar Settings**

Online Display Screen

The online display screen is similar to the ECLiPS Program Mode screen. The top bar displays the project name and current Task Group. The bottom bar of the screen displays directions for use and control keys that are currently active.

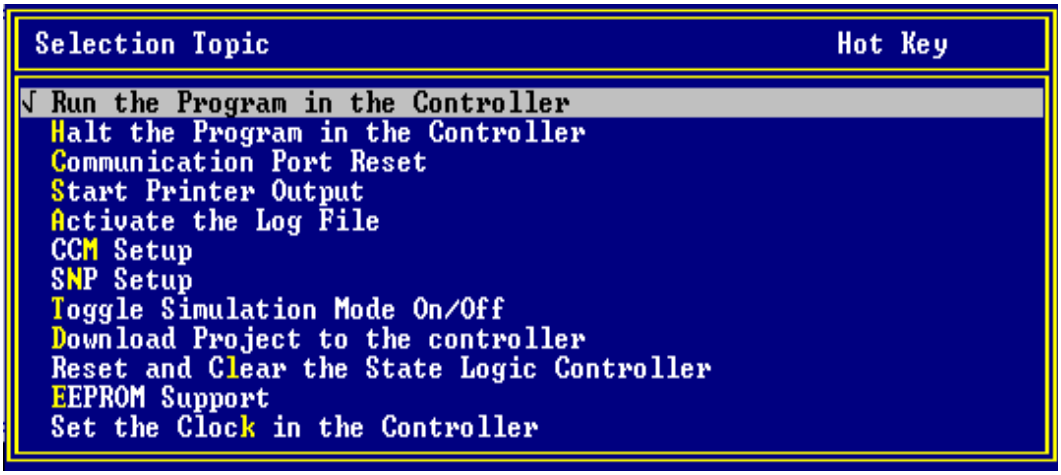
The main window, the Terminal Log displays any communication to or from the controller. Communication history can be accessed by pressing the up arrow key to scroll through the log of the communications.

The lower right corner of the screen displays the Controller status, Running, Halt, or No Comm Connect. A force indication is also displayed in this window when any points are forced. The lower left corner of the screen displays any run time errors generated by the State Logic CPU.

As with Program Mode, the online features are accessed through the menu system and the menu is accessed by pressing the <F3> key. There are a number of Hot Keys that can be used to initiate frequently used commands directly, and the context sensitive help system is accessed by pressing the <F1> key at any time.

Project Menu Options

The options on the PROJECT menu are used to select the online set up options and control the project in the controller.



Run/Halt

The Run the Program and Halt the Program options are used to control the status of the State Logic Controller. The Run/Halt condition is displayed in the Controller Status window.

Logging Data

All of the text displayed in the Terminal Log may also be logged to the hard drive of the ECLiPS or OnTOP computer. A hard copy can also be generated by sending the data to a printer connected to the parallel port, LPT1.

To log data from the controller, use WRITE program instructions to send data to the ECLiPS Port. This data is displayed in the Terminal Log and therefore can be logged to disk or printer.

When logging to a hard drive, the user is prompted to enter a file name. If the file does not exist, it is created. If the file already exists, the user has the option to either overwrite or append the existing file. Therefore several logging sessions can be stored in the same file.

CCM or SNP Setup

These options are used to change the programming port to CCM or SNP protocol. ECLiPS and OnTOP use simple ASCII communications to interact with the controller.

When the CCM or SNP protocol is enabled for this port, ECLiPS returns to the Program Mode. ECLiPS and OnTOP do not communicate with the controller while the CCM or SNP protocol is enabled. When ECLiPS or OnTOP next attempts to communicate with the controller, the programming port automatically changes from CCM or SNP protocol to immediately go online with either OnTOP or ECLiPS Debug Mode.

Note

The default State Logic serial port parameters for release 3 and beyond are as follows:

- **Baud Rate - 19,200 Baud**
- **Data Bits - 8**
- **Parity - Odd**
- **Stop Bits - 1**

CCM Set Up Procedure

To change to programming port to CCM protocol, follow these steps:

- **Set the CCM Station Address for the Controller**
- **Enable CCM Protocol**
- **Disconnect ECLiPS or OnTOP Serial Cable from the Controller**
- **Connect the Serial Cable for the CCM Device to the Controller**

To reconnect ECLiPS or OnTOP to the controller follow these steps:

- **Disconnect the Cable from the CCM Device to the Controller**
- **Connect the Cable for ECLiPS or OnTOP**
- **Connect to the Controller Normally - The Port is Reset from CCM Protocol Automatically**

SNP Set Up Procedure

To change to programming port to SNP protocol, follow these steps:

- **Set the SNP ID Number for the Controller**
- **Set the Modem TT and Idle Time Settings (Usually Set to Default)**
- **Enable SNP Protocol**
- **Disconnect ECLiPS or OnTOP Serial Cable from the Controller**
- **Connect the Serial Cable for the SNP Device to the Controller**

To reconnect ECLiPS or OnTOP to the controller follow these steps:

- **Disconnect the Cable from the SNP Device to the Controller**
- **Connect the Cable for ECLiPS or OnTOP**
- **Connect to the Controller Normally - The Port is Reset from SNP Protocol Automatically**

Process Simulation

Simulation mode can be toggled on or off from the PROJECT Menu. When simulation mode is on, the State Logic CPU does not interact with the I/O. Outputs are not sent to the output modules and inputs from the field are not seen by the program. The controller RUN LED does not come ON, even though ECLiPS reports that the program is running.

This function is used to test programs without connecting to real world I/O. Inputs may be controlled with the force function. To simulate the process create Tasks that control the inputs to exercise the program just as the real process does.

Note

While in simulation mode, the controller does not interact with any of the modules. Therefore all communications through these modules (Ethernet, PCM, CMM, GBC, GCM+, SCM) halts when the controller is in simulation mode.

Download a Project

The download option can be used to download a new project to the State Logic CPU without leaving Debug Mode or OnTOP. The current project in the controller must be halted before a new project can be loaded with this operation.

The project must have been translated and all the project files must be accessible for the download function to be successful. *See the section on file types for more information on the required project files.*

Reset and Clear State Logic CPU

The "Reset and Clear the State Logic CPU" option on the PROJECT menu is used to clear the State Logic CPU Program Memory Area and to set all configuration parameters to the default condition. A program must be downloaded before the controller is started running again.

EEPROM Support

All of the Series 90-30 State Logic Controllers provide the capability to store the control program in EEPROM. The 311, 313 and 323 models support either EEPROM or a UV EPROM while the model 331 and 340 provides a FLASH EEPROM. The 311, 313 and 323 models provide sockets for EEPROMs while the 331 and 340 EEPROM is soldered in place.

To use the EEPROM select the "EEPROM Support" option from the Debug Mode PROJECT menu. Select one of the displayed options:

- **Write EEPROM**
- **Read EEPROM**
- **Verify EEPROM**
- **Enable/Disable EEPROM**

The Write and Read options either write the program stored in controller RAM to EEPROM or Read the program from EEPROM into controller RAM. The control program is always executed from controller RAM memory. The PLC must be in halt mode to Read or Write to the EEPROM.

The Verify option compares the program in RAM with the one stored in EEPROM.

The last option on the menu is either Enable or Disable EEPROM. When this option is Enabled, the control program is automatically copied from EEPROM to RAM as power is first applied to the controller.

Information stored in the EEPROM are the control program, system configuration including all of the CPU configuration options, and the EEPROM Enabled status. To set up the system so that the program loads from EEPROM to RAM and runs automatically on power up, follow these steps:

1. Download the project with the completed program and configuration, including CPU configuration to automatically run the program on power up.
2. Select the “Write” option from the EEPROM Support Menu
3. Select the “Verify” option from the EEPROM Support Menu
4. Select the “Enable” option from the EEPROM Support Menu.

Note

If the CPU is configured to Auto Run on powerup and the EEPROM is also Enabled to load at power up, the program loads and automatically begins operation. If the battery is disconnected or gives a low power signal, the EEPROM loads the program into memory, but the program does not automatically run. Replacing the battery and cycling power, causes the program to load from the EEPROM, and if configured to Auto Run, the controller starts running the program.

PLC Clock Functions

The 311, 313, and 323 State Logic CPU use a software based clock. The clock does not hold its values over a power cycle. The date and time must be reset after every power cycle. The 331 and 340 platforms use a real time clock that does hold the correct value over power cycles, as long as the battery is functioning.

The PLC clock is set from the PROJECT menu by selecting the "Set the Clock in the Controller" menu option. The clock can be set to be equal to the PC clock, or the values can be entered manually. The values of the reserved time variables are displayed using the dislpay option or in a Monitor Tables.

Monitoring Controller Values

There are four menu options facilitating the observation of State Logic CPU data from the Debug Mode or OnTOP: Monitor Tables, View, Trace, and Display Data.

- **Monitor** tables provide a real-time view of variables, I/O, and the current State of a Task.
- The **View** option displays the program, the current State of each Task, System Status information, and the Event Queue.
- The **Trace** is used to track the Task transitions from State to State.
- **Display** data is a quick way to get a snap shot of the data of a single data point.

Monitor Tables

The monitor table options are used to create, display, modify, and delete monitor tables. A monitor table is an ongoing display of current values of selected elements. The conditions of these elements are displayed below the Terminal Log. Six elements from the following list can be combined to make a monitor table:

- String and Character Variables
- Digital Points
- Internal Flags
- Analog Channels
- Numeric (Integer and Floating Point) Variables
- Reserved System Variables (Time, Date, Fault Variables)
- Current State of a Task

Each Monitor table can show information for up to 6 data elements. There can be a total of 10 different monitor tables, one is shown at a time. To view a different Monitor Table, the "Select Monitor Table" option should be used. The <Tab> hot key quickly displays the next table without going through the menu system.

Monitor tables can be added, modified, or deleted through the use of the commands in the Monitor Menu. The bottom of the screen provides instructions on which keys to use to accomplish these tasks.

The “Clear All Monitor Tables” option can be used to clear all of the monitor tables. “Remove the Current Monitor Table” eliminates only one monitor table.

View

Use the VIEW option from the menu to display:

- Program Text
- Current States of every Task
- Event Queue
- System Status

When using the View command, the Terminal Log is replaced with the View Screen. To return to the Terminal Log press <Esc> to end the VIEW option.

View Program Text

The View English Text option displays the State Logic program text. There are special functions available while the program is displayed. As the bottom of the screen indicates, the <Alt + F> keys are used to access these functions. The functions are:

- Search for Text Strings
- Go to the Beginning of Another Task
- Display the Current Value of the Word at the Cursor

The search function locates test strings in the program. The function to find the start of a Task takes the cursor to the stated Task.

The function to display information of the word at the cursor location, displays information about the word. The type of the word (keyword, name or filler word)is displayed. If the word is a variable or I/O name the address or circuit number is displayed together with the current value. From this display the value can also be changed. I/O points are forced and variable values changed. If the cursor is located on a task name, the current state of that task is displayed and can be changed. The hot key for this function is <F4>.

Current State of Every Task

This option provides a real-time view of the current State of every Task in the program. The Task name is displayed in yellow with the constantly updated State name following in green.

Because State Logic describes the control process from the systems level instead of the I/O level, viewing the current States of the process is the most important information for debugging and troubleshooting the program. For this reason State Logic online functions are emphasize viewing current State information above view I/O data.

This option is one of the most important tools in the online functions of the system. In one screen the current status of the entire system is displayed.

Event Queue

The View, Event Queue display provides a history of important events in the controller, such as: Run, halt, power up, power down, run time errors, and system faults.

When this option is selected, a form is displayed to enter the number of entries to display. The entries are all time and date stamped. The most recent entries are at the top of the list.

POWER OFF	POWER ON	RUN PROGRAM
HALT PROGRAM	TORN EDITOR DATA	TORN DATA
MATH ERROR	SERIAL ERROR PORT 1	SERIAL ERROR PORT 2
RUNTIME ERROR	PRESCAN ERROR	LOW BATTERY
WATCHDOG TIMEOUT	BEGIN ONLINE EDIT	END ONLINE EDIT
BAD POINTER DATA	BAD PROGRAM DATA	MEM ALLOCATION ERROR
BAD DNCB REFERENCE	UNKNOWN MEMORY ERROR	LOSS OF MODULE [1]
ERR GETTING MAILBOX	BAD MAILBOX RECEIVED	SYSTEM HEAP ERROR
BAD CHECKSUM IN IO TABLE	MODULE ID MISMATCH [1]	GENUIS BUS FAULT [2]
NO MBX, MODULE BOOT MODE [1]	MBX COMM ERR IN GET/SEND [1]	SMIO, BAD CONFIG REC[1]
SMIO OR IOM RESET [1]	UNSUPPORTED FEATURE [1]	

List of State Logic Events Appearing in the Event Queue

[1] This event is followed by two numbers indicating the rack and slot numbers of the module where the event occurred

[2] This event is followed by two numbers indicating the block and circuit number of the fault

The event queue size is 64 events. Once the event queue fills, the oldest events are overwritten by the new events.

The event queue is used to augment the fault system:

1. The fault system maintains one fault per module so the event queue can be used to view multiple faults for the same module.
2. Specific block and circuit information is displayed for Genius Bus Controller faults.

System Status

System Status information lists the following information:

- Controller Firmware Version
- Current Time and Date Settings
- Memory Usage
- RUN/HALT Status
- Scan Rate

The RUN/HALT status and scan times are listed only when the program is running. The scan rate information specifies the average scan rate, the maximum, and minimum scan times over the last 1000 scans. If the scan time is displayed as 0ms, then the State Logic CPU has not had enough time to complete a thousand scans. Wait awhile and try again

Trace

The Trace is a history of the most recent program State transitions. This option allows the display of the history of State transitions on the screen for all tasks or selected tasks.

The transitions are displayed sequentially, most recent items at the top. The display shows that Task name, the starting and destination State and the time that the transition occurred. The cursor can be moved down through the trace or use the <ALT + F> key for functions that search for text strings.

The Trace is a key tool for debugging the control program. It is one method of determining the current State of the process when the process has stopped. The Trace also records past process activities to solve program logic flow problems. The Trace is also invaluable is finding intermittent problems.

The Trace is also used to provide a benchmark for future process operations. Because each State transition is date and time stamped, a Trace is a record of the process operations. A Trace of a machine when new and working properly can be stored for future comparisons. Any difference in future Traces are the result of changes in the machine or operator functions. Such comparisons reveal worn or damaged parts that affect the process timing and productivity.

By default the Trace stores State transitions for all Tasks. Usually the Trace function is customized to trace only the transitions of pertinent Tasks. Often, Tasks that are not currently relevant dominate the Trace display. By limiting the Tasks traced, transitions in irrelevant Tasks can be prevented from filling up the Trace display. To limit the Tasks displayed in the Trace, select the "Set Up Trace List" option from the Trace menu.

ECLiPS and OnTOP store the Trace display in a disk file. Each time the Trace is uploaded from the controller, this file is overwritten. The last uploaded Trace may be printed by using the Program Mode "Print Project Data" option from the PROJECT menu. The previous Trace can be displayed by selecting the "Display Previously Uploaded Trace" option from the TRACE menu.

Use the arrow keys to cursor through the Trace listing. There are also text search options available by pressing the <Alt + F> key.

Display Data

This option is used to show snapshot I/O, variable, an current State values in the controller. The value is displayed in the Terminal Log together with a time stamp.

The information is a onetime view of that value. The function is useful to get an instantaneous value of data names that can be selected from a list of names.

The Display function is also a way to enter values into the Terminal Log. A permanent record of this information is made when the functions to log data to disk or to a printer are used.

Fault System

The fault system handles module faults, alarms and a CPU low battery fault. When an I/O module fault or alarm occurs, a message is displayed on the Terminal Log in ECLiPS Debug Mode or

OnTOP and in the run-time error window. The fault or alarm is also logged and time stamped in the event queue and in the fault table. The time displayed in the Terminal Log is the time of the computer clock, while the time in the fault table and event queue is the time of the CPU clock. The fault variable values also change, when an alarm or fault occurs.

The difference between a fault and an alarm is that, once an I/O module fault occurs, backplane communications with the module are terminated. Backplane communications proceed normally after an alarm condition is detected.

Note

If an input module is in a faulted condition when the program first starts running, the input values that existed when the program halted are maintained. The control program uses these maintained values when executing its logic. These values are held over both run and power cycles. To clear these values, correct the fault condition and clear the fault from the fault table or download a program to the controller.

Fault Table

When a fault or alarm occurs, the fault is logged into the fault table. Each entry in the fault table displays the rack and slot number of the problem module, a description of the fault, and the time and date that the fault occurred. The total number of faults in the table appears at the top of the display.

To work with the fault table in ECLiPS or OnTOP, select the “Fault” option from the Debug Mode or OnTOP menu. The table can be either inspected or cleared using this option.

There are no faults or alarms generated for slots not configured, therefore, an extra module added to the system but not configured does not generate any fault or alarm condition.

One entry is logged for each module configured in the system. Only the most recent fault or alarm is displayed for each slot. To see more than one fault or alarm entry for a slot, use the event queue.

Module faults and alarms can either be cleared using ECLiPS or OnTOP. If the module fault is cleared, the CPU attempts to establish communications with the module. If the module is still in a fault condition after a fault is cleared, a new fault is written to the fault table and event queue. Backplane communications with that module are again terminated. If an alarm or fault is cleared, the fault table entry and fault variable are cleared.

Event Queue

The event queue logs hardware faults and alarms together with a time stamp. The event queue serves two fault system functions not provided by the fault table:

1. **More than one fault or alarm per slot is displayed**

   Since the event queue is just a list of system events, not only multiple faults or alarms but actually a complete fault history is recorded.

2. **Additional fault information for Genius Bus Controller faults is logged**

   The fault entry in the event queue for the Genius Buss Controller is followed by two numbers which indicate the block and circuit number of the block causing the fault. The block number is the first number following the time stamp and the second or last number in the listing is the circuit number causing the fault.

Some of the other alarms and faults listed in the event queue are also followed by two numbers. These numbers indicate the rack and slot number of the module generating the fault or alarm.

Power and Run Cycle Behavior

Faults and alarms logged during program execution continue to be displayed in the fault table and event queue when the program is halted. These entries can be cleared when the program is halted.

When the program starts executing, the fault table is cleared. If a fault condition exists, a new fault is immediately logged. The event queue is not cleared when the program begins running but is cleared when a program is downloaded to the CPU.

If the program is configured to automatically run the program on power up, the fault table is cleared when the program starts running. A faulted system can be cleared of faults without connecting ECLiPS by cycling power to the system. Configure the system to automatically run on power up, correct the fault or alarm condition, and then cycle power to the system.

Fault Variables

There are reserved system variables that store fault information for use in the program. There is one variable for each slot in the system plus one for a Low Battery fault and one which indicates if any of the slots has a fault.

The fault variables are listed under the Reserved System Variables category. These variables are integer variables that display a number code for a fault that is logged. Any non-zero number indicates a fault condition.

These variables are available in the State Logic program to effect programmed responses to fault events. The program can test for any of the fault events. There are 7 fault variables for the 311/313/323 CPU and 52 fault variables for the 331/341 CPU. The Fault Variable Names are in the following table:

311/313/323 CPU	331/340 CPU
Battery_Low	Battery_Low
Any_IO_In_Fault	Any_IO_In_Fault
Slot_1_Fault	Rack_0_Slot_1_Fault
Slot_2_Fault	Rack_0_Slot_2_Fault
- thru -	- thru -
Slot_5_Fault	Rack_4_Slot_10_Fault

An example of how these variables can be used to test to see if there is a fault in a particular slot follows:

If Rack_1_Slot_3_Fault <> 0, then go to the ShutdownTankMonitor State.

This Statement checks for any fault for the third slot in rack 1. If there is a fault, the Task transitions to the ShutdownTankMonitor State.

Note

The fault variables are read-only variables. The fault values cannot be changed from the program.

CPU Non - Critical Run-time Error Configuration

Faults are classified as non - critical run-time errors. Configuring the CPU to halt the program when a non - critical run-time error occurs, causes the program to halt when any fault occurs.

To configure the CPU to halt on non - critical errors, select the CPU slot from the System Configuration rack display. The entry for non - critical run-time error response can be 'D' or 'H'. 'D' stands for diagnostic meaning that a message is displayed but the program continues to execute. An 'H' means that the program halts in addition to a message being displayed.

System faults are displayed in the Fault Table. To view the fault table select "Fault" from the Debug Mode Menu. This option allows the fault table to be displayed or cleared.

Changing State Logic CPU Data

There are two options from the Debug Mode or OnTOP menu used to change data values in the State Logic Controller:

- Change
- Force

Change Variable Data

The Change Data option is used to change the value of string, character, and numeric variables. This function is a one time change that can be overwritten by the program.

Change Current State

The Change command is used to change the current State of any Task. Changing the current State allows the process to be controlled at the system level. Use this feature put a Task in a State to do some different operation or test how the State of the program executes.

Changing the current State of a Task is often used during system start up procedures. For example several Tasks can be placed in the INACTIVE State so that the other Tasks can be exercised independently. Also Tasks can be tested State by State with this feature.

Set Clock/Calendar

The CHANGE option is also used to set the clock/calendar values. To set the clock, choose the "Reserved System Variables" option from the CHANGE menu. The time variables are displayed:

- Month
- Day
- Day_of_Week
- Hour
- Minute
- Second

Force Inputs and Outputs

The FORCE option is used to change and hold digital I/O, Internal Flags, and Analog I/O values. When forced, inputs ignore real world sensor signals and outputs ignore program instructions.

To change a forced condition, use the force option to change the data or to remove the forces. After the force condition is removed, the input values returns to the real world status and the outputs are again controlled by the State Logic program.

There may be up to 32 discrete points and 32 analog points forced at a time.

Establishing an Online Session with the Controller

When either OnTOP or the ECLiPS Debug Mode is started, there is a series of initialization communications with the controller. This section describes the initialization procedure.

Establish Communications

First the communications are established. If communications cannot be initiated, a message is displayed with options for trying the connection again, changing the Com Port settings, or abandoning the connection. The Com port changes are to select a different Baud Rate or change to a different serial port. Either Com1 or Com2 can be used.

Check for a Controller Project

The next check is to see if a project exists in the controller. If there is not a project loaded in the controller, an option is displayed to download a project or abandon the operation. If the download option is selected, an option is displayed to select the directory where the project files are stored. From this display pressing <Ins> provides a way to browse through the directory structure as provided in ECLiPS Program Mode. Once the directory is selected a list of projects in that directory is displayed. Select the desired project from the list.

Check Project Versions

If a project does exist in the controller, a search is made of the current directory for the project files matching the project in the controller. If no project of the same name is found, an option to change directories is displayed.

When the project files are found, a safety check is made to make sure that the version of the controller project is the same as the version of the files stored on disk. This check is an important safety feature that assures that the names and program displayed in OnTOP and ECLiPS match the controller project.

If the versions do not match, an option is displayed to use the copy of the project stored in the Backup directory of the currently logged directory. If the option to change to the backup copy is selected any changes to the project, since the last download, are discarded.

When using the Automatic Backup feature, a copy of the program that has the same version number as the program in the controller should be located in the Backup subdirectory of the current directory. The version in the Backup Directory can be used to communicate with the controller, while the version in the standard working directory is used for modifications. When a new version is downloaded to the controller, the version in the Backup directory is updated to match the new version in the controller.

It is a good practice to have two copies of a project in different directories. Use one directory to store the version being used in the controller and the other directory as a working directory where new program development is performed. Access the different projects by changing the current working directory when connecting to the controller.

Online Hints

The structure of State Logic programs makes the key to trouble shooting and debugging programs knowing the current States of the Tasks. By knowing the current States, you know what part of the program is executing and therefore which outputs are ON, and what conditions must be satisfied to move to the next stage of the process. More importantly, by knowing the current State of the Tasks, you know the status of each process of the project.

When the program is running, a powerful tool for watching the process is to have the current State of every Task displayed on the screen. This display is available from the VIEW menu by choosing the "View Current State of Each Task". This screen provides a view of the status of the entire process.

To troubleshoot dynamic problems use the Trace function . The Trace shows a history of State changes including a time stamp for each change. This function shows timing comparisons between Tasks and actual State changes even though the changes happen too fast to physically see the change.

When the system has stopped for some unknown reason, check the current State of the Tasks to zoom into the section of the program that is responsible for current activities.

Use the Terminal Log as a history of process activities. Debug functions such as CHANGEs and DISPLAYs are recorded and time stamped here. Also any run-time errors and Event Queue and System Status displays are recorded in the Log. The Log also displays messages sent from the controller.

The Terminal Log is stored in a memory buffer so that items that have scrolled off the screen can be inspected by using the up arrow so that the screen does a reverse scroll redisplaying old data.. Eventually the Terminal Log can run out of memory and some of the data gets overwritten. To save information permanently, log the data to a disk file or to a printer.

Chapter 11 Serial Communications

This chapter describes 90-30 State Logic CPU Serial Communications. Serial communications are provided through the programming port located on the power supply and through the two ports of the Serial Communications Module (SCM) available in the State Logic 331 and 340 CPU systems.

The programming port on the power supply is a multipurpose port, supporting ASCII, CCM, or SNP protocol communications. However, starting with the release of version 3.13 of the ECLiPS programming software, only the ASCII protocol is supported for the SCM. This change was made to increase communications reliability over the SCM ports. Details on using the SCM are found in Chapter 3 of this manual.

This chapter is divided into the following sections:

- Programming Port
- Serial Port Parameters
- Programming Serial Communications
- SNP and CCM Communications Protocols

Programming Port

The programming port is provided on the power supply of the system. In systems using expansion racks, only the serial port on the CPU rack (Rack 0) power supply is used. Only the 331 and 340 CPUs support expansion racks.

This port is a 15-pin, RS-422/485 compatible port. See the *Series 90-30 Programmable Controller Installation Manual, GFK-0356,* for detailed cabling and other functional details of this port.

The common cabling used is the Miniconverter Kit (IC690ACC901), which includes an RS-422 to RS-232 converter, 6 foot serial extension cable, and a 9-pin to 25-pin Converter Plug assembly.

Note

This port uses hardware handshaking for flow control. If the connected device does not support hardware handshaking, communications can be completed by connecting the CTS line to the RTS line at the programming port connector or the RS-422 to RS-232 converter.

The programming port on all 90-30 State Logic CPUs is a multi-purpose port with four possible uses:

1. **Interfacing to the ECLiPS and OnTOP Software Packages**
2. **Communicating to SNP Protocol Devices**
3. **Communicating to CCM Protocol Devices**
4. **Communicating to ASCII Devices such as Bar Code Readers and Dumb Terminals**

Establishing ECLiPS and OnTOP Communications

ECLiPS and OnTOP are connected to the State Logic Controller through the serial port located on the power supply. If the programming port has been used to communicate with another device, such as an SNP or CCM device, the controller automatically switches the port to accept ECLiPS or OnTOP communications. The switching of communication modes occurs when either software product attempts to establish communications. No instructions or menu selections are required to make this conversion.

If the programming port parameters have been changed by a program instruction, those parameters must be reset to the original settings before ECLiPS or OnTOP can connect with the controller. *See the section on serial port parameters in this chapter.*

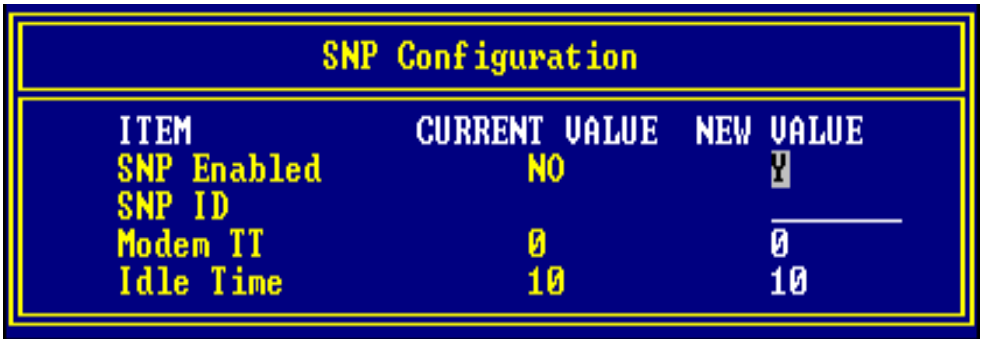
Note

Release 3 of ECLiPS/OnTOP and the State Logic CPU initiates a new set of default serial port parameter settings. Products released before version 3 are not compatible with version 3 and greater products. *See the section on serial port parameters to see the current default settings.*

Starting SNP Communications

To switch the programming port to SNP protocol, perform the following steps:

1. Start the State Logic Program Running.
2. Select the SNP SETUP Option from the DEBUG Mode or OnTOP PROJECT Menu.
3. Fill in the Displayed SNP Configuration Form.



Enter Y to Enable SNP Protocol
Enter the SNP ID - Default is 0
Fill in Modem Turn Around Time when Using a Modem
Normally Use 10 for 90-30 Idle Time

4. Press <F9> Key to Start SNP Communications on the Programming Port
5. Answer YES to the next displayed option to enable SNP and switch to Program Mode. ECLiPS now returns to Program Mode
6. Connect the Serial Cable from the PLC to the SNP Device and Start the SNP Device

Starting CCM Communications

To switch the programming port to CCM protocol, perform the following steps:

1. Start the State Logic Program Running
2. Select the “CCM Setup” Option from the Debug Mode or OnTOP PROJECT Menu
3. Choose the “Select CCM protocol station address” selection.
4. Enter the CCM Address for this Controller (1 - 90)
5. Again select the “CCM Setup” option from the Debug Mode PROJECT Menu.
6. Choose the “Enable CCM Protocol Port” Option.
7. Answer YES to the next displayed option to enable CCM and switch to Program Mode. ECLiPS now returns to Program Mode

When the programming port uses CCM protocol, ECLiPS and OnTOP no longer communicate with the controller, therefore ECLiPS returns to Program Mode and OnTOP exits to the starting menu.

Communicating with an ASCII Device

The State Logic Controller can send and receive serial information through the programming port under the control of the State Logic program. Use the program keywords, WRITE and READ, to control the ASCII communications.

To communicate from the State Logic controller to a serial device, follow these steps.

1. Start the Program Running with ECLiPS or OnTOP
2. Exit ECLiPS or OnTOP Normally Using the "QUIT" Option and Return to DOS
3. Connect the ASCII Device to the Programming Port

Note

ECLiPS and OnTOP use a special communications protocol so be sure to exit these software packages by returning to DOS, before connecting another device to the programming port. *See the section on Programming Serial Communications in this chapter for an explanation of program instructions that interact with ASCII devices.*

Serial Port Parameters

This section discuses the parameters that define how the serial port functions. The State Logic program instruction to change the port parameter settings is explained along with a table describing each of the parameters.

Setting the Port Parameters

A State Logic program keyword, Set_Commport, is used to change parameter settings of a serial port. This instruction is also used to change the Serial Communication Module port parameters although the SCM port parameters can also be changed in the system configuration screens.

Often the program is constructed so that a switch connected to a %I input circuit can trigger a change of port parameters. This method is useful for changing the parameters of the programming port, when another device is connected which requires parameters different from the ECLiPS/OnTOP settings. Use one switch setting to change the parameters and the other setting to return to the ECLiPS/OnTOP settings.

Note

Once the programming port parameters have been changed, they must be reset to the default settings before ECLiPS or OnTOP can communicate with the controller. The parameter settings can always be returned to the default settings by cycling power to the controller with the battery disconnected. The default settings are listed in Table 10-1.

EXAMPLE Set_Commport Instruction:

```
Set_Commport Bar_Code_Port with Baud_Rate=4800, Data_Bits=7,
Parity=E, Stop_Bits=1, Auto_Echo=Y, Xon_Xoff=N, Receiver_on=Y,
Stop_Transmit=Y, Respond_Backspace=Y, Line_Feed=Y,
End_of_Message=hOd.
```

When a Set_Commport command is executed, the named port's parameters are set to the new values. The programming port parameters maintain their settings over a power cycle unless the controller battery is disconnected or has a low charge when power is removed from the system.

Note

The Serial Communications Module port parameters are reset to the configured settings each time the controller starts executing its program. Therefore, if the SCM parameters are changed in the program, they get reset to the default settings after a RUN/HALT cycle or after a power cycle.

The parameters may be changed as often as necessary by the program. A single project can use a single communication port to communicate to several ASCII devices each requiring different settings.

Note

When using the Set_Commport instruction, care must be taken that the instruction is not in a State that gets executed every scan. If a Set_Commport instruction is executed while a message is being received or sent, some of the transmitted data is lost.

The easiest way to add the Set_Commport instruction to the program is to have ECLiPS enter the command. To have ECLiPS enter this instruction, follow these steps:

1. Select the "LIST" Option from the Program Mode Menu
2. Choose the "COMMUNICATION PORTS" Option
3. Highlight the Desired Port and Press the Right Arrow Key
4. Set the Correct Values in the Form
5. Press <F9> to Save the Changes
6. Highlight the Desired Port and Press <Enter>
7. Select the “Insert Reconfiguration Data for Port” Option

The Set_Commport command is entered into the State Logic Program at the cursor location. This same instruction can also be typed in by hand or copied from another part of the program.

The Set_Commport instruct controls the Serial Communication Module ports in addition to the programming port. The SCM port parameters are also controlled by the SCM configuration selections. The configuration selections are downloaded with the project and are executed when the program starts running. The SCM port parameters can be changed by the Set_Commport instruction and are reset to the configuration options each time the program starts running again.

Serial Port Parameters

Parameter	Settings	Description
Baud Rate	**19.2K,** 9600, 4800, 2400, 1200, 300	Data Transfer Rate
Utilize XON/XOFF	Y or **N**	Software Flow Control
Data Bits	**8**, 7 or 6	Number of bits per transmitted character
Parity	Even, **Odd**, None	Error checking
Stop Bits	2, **1**, or 0	Number of bits indicating end of a character.
Auto Echo	**Y** or N	Every character received is immediately transmitted out the transmit line.
Receiver Always On	**Y** or N	If the receiver is OFF, any characters sent to the controller are ignored. The default setting for this parameter is for the receiver to be OFF unless a READ term is executed in the program. When the receiver is Always ON, data received when there is no active READ is saved for the next READ that is executed by the controller.
Stop Transmit on Receive	**Y** or N	All transmission of characters is stopped when the receiver is receiving a character.
Respond to Backspace	**Y** or N	When a backspace character is received, the previous character received is deleted. When disabled, the backspace character is merely saved as other characters would be.
Issue LF After CR	**Y** or N	A line feed is sent after every carriage return.
End of Message Character	**Hex 0D**	The ASCII value of the character that indicates that a message being received by the controller is complete. When the end of message character is received, the conditional READ instruction is TRUE and the previously received characters are assigned to the READ variable.
Enable Hardware Handshaking	Y or **N**	Use handshaking lines to control the flow of data through the port.

Table 11-1. Serial Port Parameters (Default ECLiPS and SCM Settings in BOLD)

Programming Serial Communications

The State Logic program communicates through the serial ports using WRITE and READ instructions. For example:

```
State: Get_Setpoint
   Write "Enter New Setpoint" to Operator_Panel.
    Read SetPoint1 from Operator_Panel, then go to Check_Setpoint State.
```

The WRITE instruction in this example sends the string of characters, "Enter New Setpoint", to the port defined as the Operator_Panel. This same port is then monitored for input by the READ instruction in the next Statement. The serial data received through this port is assigned to the variable Setpoint1, which is used elsewhere in the program.

In this example, the WRITE and READ instruction information is directed to a particular port, Operator_Panel. Operator_Panel is defined in ECLiPS as a specific serial port. The port name is tagged to the appropriate port using the "Communication Ports" option from the LIST menu or from the "System Configuration" option of the DEFINE menu. There is also an option to define a name as a Comm Port name, when using the “Search for Undefined Words” option. The programming port and each of the Serial Communication Module ports are all named.

The READ and WRITE Instructions may also be used without being directed to any particular port, for example:

```
State: Choose_Recipe
        Write "Enter Recipe Number".
        Read Recipe_Number, then go to StartBatch State.
```

Since no port is specified in this example, information is sent to the programming port, located on the Power Supply of the CPU rack..

The READ instruction is used to bring in serial data to Integer, Floating Point, String, or Character variables. This instruction forms a Conditional Expression which becomes TRUE only when a complete message is received. If the type of data that is input does not match the data type of the variable used in the READ, the READ instruction remains false and the GO is not executed.

EXAMPLE:

```
State: Get_Float_Number
  Read Scale_Weight from Operator_Interface go Manual_Scale_Data.
```

In this example, if Scale_Weight is defined to be a floating point number, the READ instruction is expecting to receive floating point data. If the data received includes character data such as "12R.456", the Read instruction remains unsatisfied and the State Logic Program does not execute the GO to the Manual_Scale_Data State.

Note

The READ instruction is a conditional instruction that must be used with a GO functional instruction in the same Statement. When data is received the GO instruction is executed. There can be no other instructions in the Statement besides the READ and the GO instructions.

For a more detailed discussion of programming serial input and output, look up the READ and WRITE instructions in the Programming Instructions chapter.

SNP and CCM Communications Protocols

SNP and CCM protocols are standard open communications protocols defined by GE Fanuc and documented in the GE Fanuc protocol manuals. This section refers to the SNP and CCM protocols used by the programming port on the power supply. The SNP and CMM protocols were formerly supported for the SCM ports. However, beginning with version 3.13 of the ECLiPS programming software, this support was dropped. If an additional CMM or SNP port is required, use a CMM or PCM module. Different features and a different implementation of these protocols is provided by the CMM and PCM modules. *See the manuals on those products for more information about using these modules for SNP and CCM communications.*

The State Logic CPU always acts as a slave in a master-slave architecture. The master initiates all communications, data retrieval is accomplished by polling. The SNP and CCM protocols define the message structure, framing, error checking and handling, and timing for all message types. Use the CCM or PCM modules for protocol master capabilities.

Protocol Data Types and Reference Numbers

State Logic provides additional word based data types beyond those offered by most PLCs. These unique data types are:

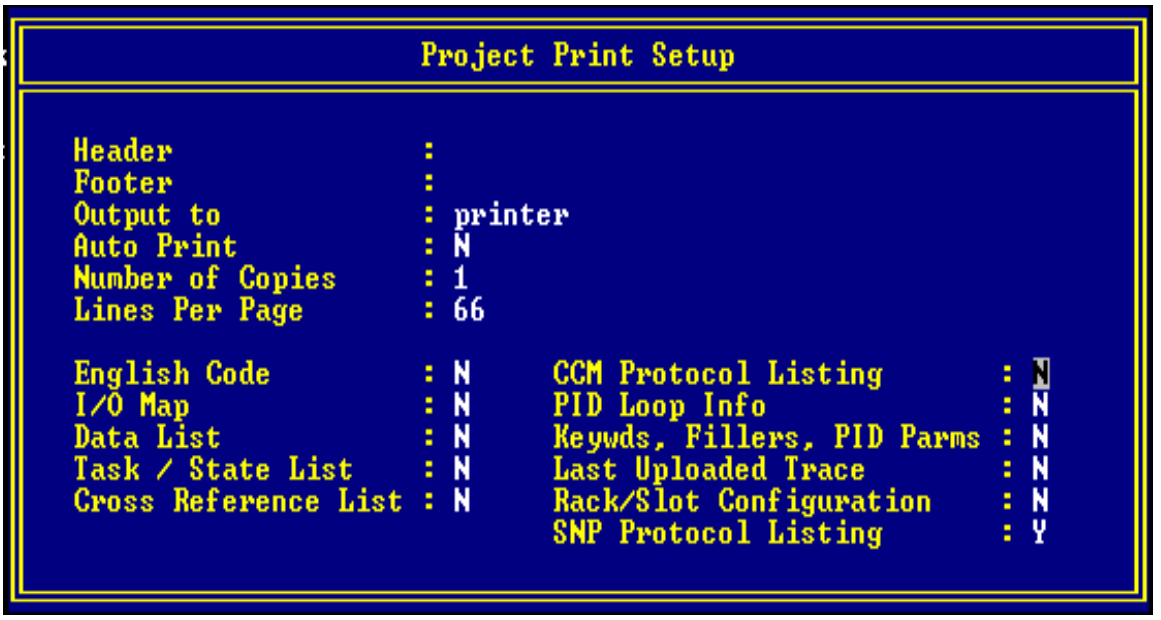
- Current State of a Task
- Integer Variables
- Floating Point Variables
- Scaled Analog I/O Values (Floating Point)
- String Variables
- Character Variables
- PID Parameters

All of these data types are identified as register locations in the master device accessing this data. The following section describes how this data is mapped to register locations.

Displaying Protocol Reference Number Mapping

To list the data type and reference number for every data name that is used in the State Logic program plus the current State of Task information, use the ECLiPS PRINT function.

Select the “Print Project Data” option from the Program Mode PROJECT menu to display the following form:



To print the SNP information, enter 'Y' for the SNP Protocol Listing option, as displayed. The CCM listing is printed when the CCM Protocol Listing option is set to 'Y'. This form interprets any entry other than PRINTER in the "Output To:" blank, as a file name. The information is written to a file when a file name is entered in this blank.

%R Register Data

The %R register references are used to access data types unique to State Logic. Each State Logic register data type occupies a specific range of registers. Individual data references use a different number of registers depending on the data type. The following table displays the ranges, the number of registers, and the data type of the State Logic register data.

Variable Type	Register Range	Data Type	Registers per Variable
Integer Variable **Note:** Number limited to 250 in 313 and 323 CPUs	1-1000	Integer	1
States of Tasks **Note:** Number of Tasks Limited to 254 in the 90-30	2001-3000	Integer	1
Floating Point Variable Note: Number limited to 61 in 313 and 323 CPUs	3001-4000	Floating Point	2
Analog Input	5001-5512	Floating Point	2
Analog Output	6025-6280	Floating Point	2
String Variable **Note:** Number limited to 20 String Variables in 313 and 323 CPUs	6601-7419	ASCII	41
Character Variable	7421-7484	ASCII	1
PID Parameters	7501-8940	Floating Point	42

Table 11-2. State Logic SNP and CCM %R Register Data Types

Note

The SNP or CCM device connected to the 90-30 State Logic CPU should always be configured as if they were connected to a 341 CPU. Some of these devices limit the number of registers accessible to 512 if configured to be connected to another CPU model.

When accessing the register data types, use the register number displayed in the print out of SNP or CCM numbers as described above. Use the data type and the number of registers specified in this table.

For example, to access the first two floating point variables reference them as registers 3001 and 3003 each with a length of two registers.

States of a Task- There is one register assigned to each Task. The value of that register reflects the current State of that Task. The SNP and CCM Protocol Listings display the number for each State and the register number for each Task.

Caution

The registers are assigned to the Tasks in the order that the Tasks appear in the program. First register assigned to first Task. Similarly, numbers are assigned to each State according to the order that they appear in the Task. Therefore, the register numbers assigned to the Tasks and the values assigned to each State may change as the program changes.

When changing the current State of a Task, care should be taken that the new State specified by the new value actually exists. If the State does not exist, an attempt to put the controller into a non-existent State results in a critical error.

String Variables - The State Logic strings are a maximum of 80 characters long and always delimited by a null character (ASCII 0). There are 41 registers reserved for each string variable. The first two string variables are accessed at %R6601 and %R6642 each with a length of 40 registers.

Note

When changing the value of a State Logic string variable via the protocols, specify only a maximum of 40 registers. Never write data to register 41 for any string variable.

PID Loop Parameters - There are 21 floating point parameters associated with each PID loop. These parameters are accessed in the 42 registers starting with the register displayed in the printout for each PID loop name. Each parameter uses two registers and is stored in the following order:

Parameter Number	PID Parameter Name	Parameter Number	PID Parameter Name
1	Status	12	Control Variable
2	Update Time	13	Control Variable Minimum
3	Gain	14	Control Variable Maximum
4	Reset	15	Bias
5	Rate	16	Lower Limit
6	Setpoint	17	Lower Limit Minimum
7	Setpoint Minimum	18	Lower Limit Maximum
8	Setpoint Maximum	19	High Limit
9	Process Variable	20	High Limit Minimum
10	Process Variable Minimum	21	High Limit Maximum
11	Process Variable Maximum		

Table 11-3. PID Loop Parameter Data

The first parameter, Status, is the only parameter that is not a floating point value even though two registers are reserved for this parameter. This parameter contains bit information about the PID loop operation. The status information is contained in following bit locations:

- Bit 0 - Block Down
- Bit 1 - Block Up
- Bit 2 - Track Mode
- Bit 3 - Inverse Mode
- Bit 8 - Low Limit Status (Read Only)
- Bit 9 - High Limit Status (Read Only)

If the bits are set, the corresponding status is true. The other bits in these registers are reserved and should not be changed.

Accessing Analog Data

State Logic controllers provide SNP and CCM access to analog data through the %R register data types. Direct access to analog data is a feature provided by State Logic CCM protocol that is not normally available in CCM slave devices.

If the analog channel is scaled, the value in the register is the floating point scaled value. Make sure the master device supports IEEE 32-bit floating point data when accessing scaled analog data.

If the analog channel is not scaled, the unscaled integer value is available in the first register. Two registers are always reserved for the channel, whether or not it is scaled. The second register has a zero value.

SNP devices can access the %AI/AQ data types directly. The values available using these references are always unscaled, integer values, even when that channel is scaled.

To change an analog output from the SNP device, always use the %R register designation for the output channel. Analog outputs cannot be changed by using the %AQ designation. For both CCM and SNP, if the channel is scaled, use floating point data to make the change, otherwise use integer data. Remember that only the first of the two registers is used when the analog channel is unscaled.

Note

Forced analog input values are not available using any remote device (CCM or SNP protocols, Ethernet Module, CMM, PCM, GBC, and GCM+). Only the real world input value is available even when the channel is forced in the State Logic controller.

Accessing Digital Data

Serial protocol devices may always read the condition of any State Logic digital point but may not be able to change some of the types under certain conditions.

The following rules explain how each digital data type is changed from an SNP device:

- Digital Outputs (%Q) may never be changed directly from an SNP or CCM device, since these points are OFF by default and always controlled by the State Logic controller. To control an output, either use SNP or CCM to change the State of the Task which controls the output, or program that output to be ON when a different digital type, such as a %G, is turned ON by the SNP device.
- Digital Inputs (%I) configured to input module circuits cannot be changed. Digital inputs not configured to module locations can be changed. Once an SNP or CCM device turns ON an unconfigured %I point, it remains ON until a remote device turns the input OFF.
- Global Digital Points (%G) can be configured to be sent or received as global data using either the GBC or GCM+. When being used as global data, %G bits should not be changed by another device. Those points not configured can be controlled by an SNP device. Once

turned ON by a remote device, unconfigured %G points remain ON until turned OFF by a remote device. The State Logic program can read, but cannot control these unconfigured points.

- Internal Flags (%M) may all be turned ON by the SNP or CCM device. These flags are always OFF by default, just as the digital outputs are. Therefore, when a flag is turned ON using a serial protocol, it remains ON for only one scan and is then turned OFF by the State Logic operating system. If the State Logic program is testing the flag when the SNP or CCM device turns it ON, the test is true.

SNP Access to Digital Data

There are four digital State Logic types. The SNP types used to access these types are listed in the following table:

State Logic Data Type	SNP Data Type
Discrete Input	%I
Discrete Output	%Q
Internal Flag	%M
Discrete Globals	%G

Table 11-4. State Logic SNP Digital Data Types

CCM Access to Digital Data

The State Logic implementation of CCM provides direct access to more digital data types than the original CCM protocol definition, since %G points and internal flags (%M) can be accessed in the State Logic implementation.

All CCM references to digital points use the digital input (%I) data type including references to %Q outputs. All of the State Logic digital data types are mapped into different ranges of the %I reference numbers.

Use the “CCM Protocol Listing “ option on the Print Project Data form to get a mapping of State Logic digital points to %I locations. The following table shows the %I reference number ranges for each of the State Logic digital types.

Data Type	CCM Number	CCM Type
Digital Input, %I	1 - 512	Discrete Input (%I)
Discrete Output %Q	1025 - 1536	Discrete Input (%I)
Discrete Global %G	4097 - 5376	Discrete Input (%I)
Internal Flag	5001 - 6000	Discrete Input (%I)

Table 11-5. State Logic Data Types and CCM References

Note

This method of CCM access to digital data applies only to access through the programming port on the power supply. *See the relevant manuals for using the CMM and PCM ports to access digital data using the CCM protocol.*

Appendix A Keyboard Functions

This appendix lists how the keys are used by the ECLiPS and OnTOP software.

Function Hot Keys

Key	Functions			
	<Key>	<Alt + Key>	<Ctrl + Key>	<Shift + Key >
<F1>	Help	Toggle Insert/ Overtype Mode	Project Error Help	
<F2>	Save Project	Translate and Download	Retrieve Project	Translate and Error Check
<F3>	Menu			
<F4>	Define/ View Current Word	Search for Undefined Words		
<F5>	Find / Replace Next	Find Text	Replace Text	Replace All
<F6>	Add State	Add Task		
<F7>	Go to Another Task	Last Project Error		
<F8>	Mark a Block	Copy a Block	Move Block	Remove a Block
<F9>	Save/Exit Form	Next Page in Form		
<F10>	Toggle Program/ Debug Modes			

Other Hot Keys

Key	Function	Mode
<Ctrl + D>	List Digital Points	Program/Debug
<Ctrl + A>	List Analog Channels	Program/Debug
<Ctrl + N>	List Numeric Variables	Program/Debug
<Ctrl + S>	List String/Character Variables	Program/Debug
<Ctrl + Q>	Quit Current Mode	Program/Debug
<Ctrl + K>	List Keywords	Program
<Ctrl + W>	List Filler Words	Program
<Ctrl + P>	List PID Loops	Program
<Ctrl + U>	Undelete the last Deleted Block **(PASTE)**	Program
<Ctrl + R>	Reset Run-Time Error Message Window	Debug
<Ctrl + V>	View English in Debugger	Debug
<Ctrl + V>	Display ECLiPS Version	Program
<Ctrl + E>	Enable Monitor Display in Debugger	Debug
<Ctrl + F>	Force Table	Debug
<Ctrl + T>	Trace Upload/Display	Debug

Miscellaneous Keys

Key	Functions	
	<Key>	<Ctrl + key>
<Insert>	Add/Paste	
<Delete>	Remove/Cut Character/Item	
<Home>	Left side of Screen/Field	
<End>	Right side of Screen/Field	
<Pg Up>	Scroll Up	Top of Project / List / Menu
<Pg Down>	Scroll Down	End of Project / List / Menu
<Up>	Up 1 Line/Field	
<Down>	Down 1 Line/Field	
<Left>	Left 1 Character/Field	
<Right>	Right 1 Character/Field	
<Tab>	Insert 4 spaces / Next Item in List/Menu	
<Enter>	Carriage Return / Select Item	
<Esc>	Cancel Operation	

Appendix B Language Structure

Notational Conventions

This section rigorously defines the syntax for the State Logic language with a notational convention similar to that used for Bakus-Naur representations for programming languages. First the notation conventions are explained, then the syntax is represented in a top down manner starting with the program.

Bold Italics	**- Identifies Keywords**
[ ]	- Encloses terms which are optional
{ }	- Encloses terms which may be repeated
< >	- Encloses a generic description of a term
|	- Indicates that the term before or after may be used at this point.
( )	- Group Terms Together

Program Hierarchy

As an example of how this notational system works, compare the explanation of the program hierarchy with the notation listed in the table at the bottom of this page.

Program - One or more Task Groups

Task Group - One or more Tasks

Task - The keyword Task followed by a colon, then a Task Name, then optionally the keyword Start_In_Last_State, then one or more States.

State - The keyword State followed by a colon, a State Name and then zero or more Statements.

Statement - A Functional Expression and optionally a Conditional Expression. Either of the expressions can come first.

Term	**Structure**
Program	{ <Task Group> }
Task Group	{ <Task> }
Task	***Task***: <Task Name> {<State>} [***Start_In_Last_State***]
State	***State***: <State Name> [{ <Statement> }]
Statement	([<Conditional Expression>] <Functional Expression>) | (<Functional Expression> [<Conditional Expression>])

Functional Structures

Term	Structure
Functional Expression	{ Functional Term }
Functional Term	< Turn On Discretes Term > | < Assign Values Term > | < Change Active States Term > | < Send Serial Information Term > | < PID Control Term > | < Change Serial Port Configuration Term > | < Execute Perform Functions Term > |
Turn On Discrete Term	***Actuate***{ <Digital I/O Name> | < Internal Flag Name > }
Assign Values Term	< ***Make*** Term > | < Math-Assignment Term > | < ***Set_Bit/Clear_Bit*** Term >
Make Term	***Make***(< Numeric Assignment Term > | < Character Assignment Term > | < String Assignment Term>)
Numeric Assignment Term	(<Numeric Variable Name> | <Analog I/O Name>) ***equal*** < Numeric Value >
Character Assignment Term	<Character Variable Name> ***equal*** <Character Value>
String Assignment Term	<String Variable Name> ***equal*** <String Value>
Math-Assignment Term	< ***Add*** Term > | < ***Subtract*** Term > | < ***Multiply*** Term > | < **Divide** Term >
Add Term	***Add*** (< Numeric Constant > | < Variable Name>) < Variable Name >
Subtract Term	***Subtract*** (< Numeric Constant > | < Variable Name>) < Variable Name >
Multiply Term	***Multiply*** (< Numeric Constant > | < Variable Name>) < Variable Name >

Functional Structures Continued

Divide Term	***Divide*** (< Numeric Constant > | < Variable Name>) < Variable Name >
Set_Bit/Clear_Bit Term	(***Set_Bit*** **|** ***Clear_Bit***) (<Integer Variable Name> <Integer Number>)
Change State Term	(***Go*** <State Name>) | (***Make***<Task Name>***equal***<State Name>)| ((***Suspend_Task*** | ***Resume_Task***) <Task Name>
Send Serial Data Term	***Write***" <Serial Data> " [<Port Name>]
PID Loop Control Term	<***Start_PID*** Term> | <***Stop_PID*** Term>
Start PID Term	***Start_PID*** <PID Loop Name>
Stop PID Term	***Stop_Pid***<PID Loop Name> [***with*** <Numeric Constant>]
Port Configuration Term	***Set_Commport*** <Port Name> <Parameter Value List>
Perform Function Term	***Perform***<Function Name> ***with*** <Parameter Value List>

Conditional Structures

Term	Structure
Conditional Expression	< Test Conditional > | < Character Input Conditional >
Character Input	***Read*** <Variable Name> [***from*** <Communications Port Name>]
Test Conditional	***If***[***NOT***] <Conditional Term> [{(***OR***|***AND***)[***NOT***]<Conditional Term>}]
Conditional Term	<Digital Test Conditional> | <Timer Test Conditional> | <Relational Test Conditional> | <Current State Test Conditional>
Digital Test Conditional	(<Digital I/O Name> | <Flag Name>) [{ (***AND*** | ***OR***) (<Digital I/O Name> | <Flag Name>) }] (***ON*** | ***OFF***)
Timer Conditional	(<Numeric Constant> | <Integer Variable Name>) ***seconds***
Current State Conditional	<Task Name> (***equal*** | ***not_equal***) <State Name>
Relational Test Conditional	<Numeric Relational Term> | <Character Relational Term> | <String Relational Term>
Numeric Relational Term	<Numeric Value> <Relational Operator> <Numeric Value>
Character Relational Term	<Character Value> (***equal*** | ***not_equal***) <Character Value>
String Relational Term	<String Value> (***equal*** | ***not_equal***) <String Value >

Value Expressions

Term	Structure
Numeric Value	<Numeric Constant> | <Calculation> | <Numeric Variable Name> | <Analog I/O Name> | <PID Value>
Calculation	(< Numeric Value > < Operator > < Numeric Value >) | <System Functions> (<Numeric Value>)
System Functions	***SIN | COS | TAN | ARCTAN | SQRT | EXP | LN | RANDOM***
Numeric Constant	<Floating Point Number>|<Integer Number>
PID Value	<PID Loop Name> <PID Parameter Keyword>
Character Value	< Character Variable Name > | ' <Character> '
String Value	<String Variable Name > | "<Character String>" - Up to 80 characters

Appendix C Standard Predefined Keywords

ECLiPS comes with a set of Keywords supplied. Up to 10 synonyms may be added for each Keyword, and the Default Keyword may be changed. The total number of keywords plus synonyms is 100. The Keywords are broken into four categories, Conditional terms, Functional terms, Operators and miscellaneous words that modify the meaning of a Statement. In the following tables, the default keyword is displayed in bold print with some suggested synonyms in normal print.

Conditional Terms

Keyword, Synonyms	**Meaning / Examples**
If, When	Test conditions and values, actions executed when test returns TRUE condition. **When** the Forward_Limit_Switch is ON, go . . . **If** Count is > 1, go . . . **If** 3.56 seconds, go . . .
Read	Get input from programming or SCM serial port **Read** Name from Port_1.

Functional Terms

Keyword, Synonyms	Meaning / Examples
Energize, Start, Actuate, Turn, Run, Open, Turn_On	Turn on a Digital Point(s) **Start** Conveyor_Motor. **Energize** Forward_Solenoid. **Actuate** Backwash_Pump, and Backwash_Pump_Light.
Add	Add a value to a variable **Add** 2 to Count.
Divide	Divide a variable by a value **Divide** Count by 2.
Go	Make another State the Active State If Switch1 is On, **go** to the Motion State.
Halt	Stop the process immediately If Alarm is On, **Halt**.
Make, Put, Place, Set	Assignment operator initiator **Make** Total = 0.
Multiply	Multiply a variable by a value **Multiply** Count by 2.
Perform, Execute	Invoke an ECLiPS "Perform" function **Perform** Display_Date_Time with...
Set_Commport	Change comm port settings while running **Set_Commport** Port_1 with Baud_Rate=9600, Data_Bits=8, Parity=N, Stop_Bits=2, Auto_Echo=Y, Xon_Xoff=Y, Receiver_On=N, End_of_Message_Char=h0d.
Start_PID	Invoke a PID Loop **Start_PID** Main_Loop.
Stop_PID	Halt a PID Loop and set the output value. **Stop_PID** Main_Loop with 234.5.
Subtract	Subtract a value from a variable **Subtract** 1 from Count.
Set_Bit	Set a condition TRUE or a bit of a 16 bit integer value to 1 **Set_Bit** Flowmeter_Counter HSC_OUTPUT_ENABLE. **Set_Bit** Integer_Variable_1 0.
Clear_Bit	Set a condition FALSE or a bit of a 16 bit integer value to 0 **Clear_Bit** Resolver_Counter HSC_RESET_PRELOAD. **Clear_Bit** Integer_Variable_2 15.
Suspend_Task	Stop the execution of a Task - no State is active. If Level > alarm, **Suspend_Task** Automatic.
Resume_Task	Restart Task in State active when the Task was suspended. If Level < alarm, **Resume_Task** Automatic.
Write	Send data out the comm port **Write** "Error Message" to Operator_Console.

Operators

Keyword, Synonyms	Meaning / Examples	Precedence
( )	Parentheses - Used to group terms to change order of operation. Up to 18 levels of parentheses are permitted. Parentheses may be used in mathematical expressions and with relational conditional terms.	1
ARCTAN(exp)	Arctangent: function returns an angle in radians where -65535 <= exp <= 65535 Make Var = **ARCTAN**(Hyp * 2).	2
COS(exp)	COSINE: exp is the angle in radians where -65535 <= exp <= 65535 Make Near = **COS**(Test_Value).	2
EXP(exp)	e to a power: Make Inverse = **EXP**(Transfer).	2
LN(exp)	Natural logarithm (base e): Make Test_Value = **LN**(Input - 3.4)	2
RANDOM	Random number generator: Generates a random number (0 - 1) Make Sim_In = Seed * **Random**	2
SIN(exp)	Sine: exp is the angle in radians where -65535 <= exp <= 65535 Make Vector1 = **SIN**(Gauge).	2
SQRT(exp)	Square Root: Make Out_Pot = **SQRT**(Flow_Meter).	2
TAN(exp)	Tangent: exp is the angle in radians where -65535 <= exp <= 65535 Make Slope = **TAN**(In_Flow).	2
^	Exponential Operator Make Count = Amount ^ 2.	2
Test_Bit	Determine bit values in integer variables If Test_Bit (Status_Variable 5) is 1 . . .	2
%, Modulus	Modulus Operator - integer operands only If Count % 5 = 0, go...	3
***, Times**	Multiplication Operator Make Count = Amount * 2.	3
/, Divided_By	Division Operator Make Count = Amount / 2.	3
+, Plus	Plus Operator Make Count = Amount + 2.	4

Operators Continued

Keyword, Synonyms	Meaning / Examples	Precedence
-, Minus	Minus Operator or Negative sign Make Count = Amount - 2.	4
Bitwise_And	AND bits operator Value = Code Bitwise_And Mask	4
Bitwise_Or	OR bits operator Setup = Code Biwise_Or Mask.	4
<, Less, Under	Less than Operator If Count < 5, go...	5
<=, =<	Less then or equal to Operator If Count <= 5, go...	5
=, Is, Equal, Equals, Into	Equal Operator If Count = 5, go...	5
>, Greater, Above, More	Greater than Operator If Count > 5, go...	5
>=, =>	Greater than or equal to Operator If Count >= 5, go...	5
<>, Not_equal	Not Equal Operator If Count <> 5, go...	5
AND	AND Operator for Conditional and Functional Terms If Count > 5 **AND** Top_Switch is ON Actuate Pump_5 **AND** Pump_6.	7
OR	OR Operator for Relational Conditional Terms Only If Vat < 98 degrees **OR** Fuel < 12	8
NOT	NOT Operator for Relational Conditional Terms Only If **NOT** Inlet_Pressure > 100 psi	6

Miscellaneous Keywords

Keyword, Synonyms	Meaning / Examples
State	Identifies the Name of State Logic States **State**: PowerUp go to the Motion **State**.
Task	Identifies the Name of State Logic Tasks **Task:** Main If the Main **Task** is in the PowerUp State, go...
AM	Time Suffix If Time is past 3:00 **AM**, go...
Friday	Day of week number 5 If day_of_week = **Friday**, go...
From	Used in "Read" Terms Read Name **from** Port_1.
Max_Time	Used to set the maximum time diagnostic for a State State: PowerUp **Max_Time** 2.5
Monday	Day of week number 1 If day_of_week = **Monday**, go...
Not	Logical "NOT" in a conditional expression If **not** (Count > 1 or Count < 10), go...
Off, False, Not_True,Not_Tripped	Test for Digital I/O for not set state If Switch1 is **Off**, go...
On, True, Tripped	Test for Digital I/O for set state If Switch1 is **On**, go...
Or	Logical "OR" in a conditional expression If Count > 1 **or** Count < 10, go...
Inactive	Name of the State in Which No Actions Occur Put the Manual Task into the **Inactive** State.
PM	Time suffix If Time is past 3:00 **PM**, go...
Saturday	Day of week number 6 If day_of_week = **Saturday**, go...
Seconds	Used for comparison in a Timer Term If 3.2 **seconds** have passed, go...
Sunday	Day of week number 7 If day_of_week = **Sunday**, go...
Thursday	Day of week number 4 If day_of_week = **Thursday**, go...
Tuesday	Day of week number 2 If day_of_week = **Tuesday**, go...
Wednesday	Day of week number 3 If day_of_week = **Wednesday**, go...
With	Prefix for data that is needed by a function Stop_PID Kiln_Temperature **with** 45.679
Start_In_Last_State	This keyword is used to configure a Task to start in the State that was active when the program stopped. Task: Master_Control **Start_In_Last_State**

Appendix D Errors

There are two types of error codes, **Translation Errors** and **Run-time errors**. Translation error codes are generated by ECLiPS during the error checking stage of the project translation. Run-Time Errors are generated by the State Logic CPU during program execution.

Translation Errors

The ECLiPS error check process is completed in two sweeps. If any errors are found during the first sweep, the translator does not perform the second sweep.

A list of the translator errors is displayed after the error checking is completed. To move the cursor to the line where the error is located, high light the error in the list with the arrow keys, then press the <Enter> key. The List of errors can be redisplayed by selecting the “List Translator Error” option from the FIND menu or using the hot key, <ALT+F7>.

Error	Error Description
0	**Invalid or Missing Task Name.** Every Task must start with the following line: "Task: Task_Name" Task_Name may be up to 20 characters and must be a unique name in the project. It must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally. If the previous State does not end with a period this message may come up. In addition, if the Task Name is followed by a period this may appear.
1	**Invalid or Missing State Name.** Every State must start with the following line: "State: State_Name" State_Name may be up to 20 characters long and must be unique within the current Task. It must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally. If the previous State does not end with a period this message may come up. In addition, if the State Name is followed by a period this may appear.
2	**Maximum number of States reached.** The maximum number of States per Task is 254. The maximum number of States per Project depends on the value in the Setup menu.
3	**Maximum number of Tasks reached.** The maximum number of Tasks allowed is 255.
4	**Undefined Character in Text.** The character in quotes has not been defined. Names must start with a letter and may contain only letters, numbers and the "_" character. All other characters must be defined as operators such as +, *, etc.
5	**Invalid use of the "Perform" Statement.** If one of the following 3 "Perform" statements is used, it must be the only statement in the State. Get_User_Input User_Menu Display_Date_&_Time

6	**Invalid use of a "Read" Statement.** A "Read" may only be used along with a "Go" in a sentence and only one "Read" is allowed per State. Incorrect: Read StringInput Correct: Read StringInput go to the EvaluateInput State
7	**No "PowerUp" State in Task.** Every Task must have one and only one "PowerUp" State. It may appear anywhere in the Task.
8	**Invalid Multiple Comparison in "If" Statement**. In a conditional statement, each condition must be a complete statement. Valid Statement: If var1=1 and var2=1 and var3=1 Invalid Statement: If var1=var2=var3=1
9	**Multiple Exponent Operators Invalid.** ECLiPS limits the use of exponents to one per statement. Invalid Statement: make var1 = var2^var3^var4 Equivalent Valid Statements: make var1 = var2^var3 make var1 = var1^4
10	**Invalid Exponent Operator.** The exponent function can be used with either the + or - operator. All other operators are invalid in a exponent operation Incorrect: Make Value1 = Value1 ^ *3. Correct: Make Value1 = Value1 ^ -3.
11	**Maximum Number of Write Statements Reached.** The maximum number of write statements allowed in each state is 32.
12	**Invalid Command.** The word or character in question is not defined as a valid command that the translator needs to start or complete the sentence.
13	**Missing Functional Term in Statement.** A Conditional statement was found with no Functional statement anywhere else in the sentence. This is not allowed. Example Error: If Valve1 is on. Correct If Valve1 is on go ValveOn State.
14	**Two Consecutive "ANDS" in a row.**
15	**Invalid Digital or Internal Flag name.** The name in the expression has not been defined as Digital Point or Internal Flag. Use the "List" or "Define" function to define it properly. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.
16	**Expecting Numeric Value.** A numeric value is needed to complete the statement.
17	**Not Used**
18	**Invalid Variable Name.** The name found has not been defined as a variable. Use "List" or "Define" to properly define it. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.
19	**Expecting "FROM" as the next word instead of...** The word "From" or a synonym for it is needed to complete the "Read" statement if a Comm Port name is used. The word in the quotes should be defined as a comm port name.
20	**Invalid Comm Port Name.** The name found at the end of the statement must be defined as a comm port name. Use "List" or "Define" to properly define it.
21	**Expecting "SECONDS" as the next word instead of...** The word "Seconds" or a synonym for it is needed to complete this type of statement instead of the word in quotes.
22	**Not Used**
23	**Expecting quoted string instead of...** A set of characters in quotes is needed to complete this statement. Information that is assigned to or compared with a string variable must be in quotes.
24	**Matching "(" not found.** A "(" was found without a following "(".
25	**Matching ")" not found.** A ")" was found without a following "(".
26	**Not Used**

27	**Missing either "ON" or "OFF" after Digital List.** The word found at the end of the Digital expression must be a synonym for either "On" or "Off" in order to complete the expression. Incorrect: If Output1 go to OutsareUs State Correct: If Output1 is on go to OutsAreUs State
28	**Expecting an Operator instead of.....** Expecting an operator instead of what is in the quotes.
29	**Not Used**
30	**Error Reading Project** The Project File has been corrupted and cannot be read. Consult your DOS manual for instructions on how to determine the status of the file. The Restore Backed up file can be used if there is a backup version of the project. In addition, the file may be recovered from the backup versions of the project. See section on Program Features for further information on backup and backup files.
31	**Invalid Number or Number Variable.** The variable assigned to a numeric data type must be a numeric variable.
32	**Invalid Perform Name.** The Perform names are limited to those found on the List of Performs.
33	**Not Used**
34	**Expecting "WITH" as next word instead of....** Either the word "With" or a synonym for it must appear in this type of statement instead of the word that is seen in the quotes.
35	**Missing "="** Either the "=" sign or a synonym for it must appear in this type of statement.
36	**Not Used**.
37	**Not Used**
38	**Missing Number in Exponent.** The exponent operator was used without a following number.
39	**Invalid Parameter.** The Parameters allowed for each perform function are predetermined. Use of the ADD a Perform Function form will ensure that the correct Parameters are used.
40	**Invalid Parameter Name.** Each Parameter is looking to be filled by certain pre-defined parameter names. The help menu associated with each Perform Function Form provides a list of the appropriate parameter names.
41	**Expecting "Yes" or "No" instead of.....** The word "Yes" or "No" must appear in this type of statement. The information in the quotes should be either yes or no or a synonym for yes or no.
42	**Too many Parameters for Perform.** Attempt to rewrite the perform function using the form function found under the ADD menu selection.
43	**Invalid String Constant.** A string type variable may only be assigned up to 80 characters in double quotes. Any % character in a string must be immediately followed by #xx where xx represent valid hexadecimal digits, or by another % character.
44	**Invalid Character Constant.** A character type variable may only be assigned a single character in single quotes.
45	**Invalid Assignment.** A variable of a particular type may only be assigned values that are of the same type.
46	**Digital Outputs and Flags Cannot be Turned OFF.** The statement contains a synonym for the word "OFF" and this is not allowed. I/O that is Actuated in a State is ON, ALL others are implicitly OFF.
47	**Missing ")".** A "(" was found without a following ")" parenthesis must come in matched pairs.
48	**Missing "(".** A ")" was found without a preceding "("parenthesis must come in matched pairs.
49	**Data Type Mismatch.** An assignment or comparison was attempted between two different data types.
50	Not Used.
51	**Two Operators used together.** Only one operator can be used in each expression. Example of incorrect usage: make var1 = var2 +* Var2.
52	**String Assignment is too long.** Max. is 80. Because of the storage limitations in the controller, strings are limited to a length of 80 characters.
53	**Invalid or Missing Operator.** A valid operator (+-=<>*/%) is needed to complete the statement in question. All "If" and "For" statements require a comparison operator (<>=) and "Make" statements require an assignment operator (=).
54	**Expecting a "."** All Sentences must end with a Period.

55	**Duplicate or Missing Task Name.** A State name is either missing on the line with the "State:" or the State name entered there is duplicated somewhere else in the same Task.
56	**Duplicate or Missing Task Name.** A Task name is either missing on the line with the "Task:" or the Task name entered there is duplicated somewhere else in the same Project.
57	**Invalid Comparison.** Comparisons must be made between variables and constants of the same data type. Comparing numeric with strings, or discrete with analog could give this error message.
58	**Misplaced Parenthesis.** The "()" characters must follow operators as in "1 * (2+3)" and may not follow operands as in "1 (* 2+3)".
59	**Expression too long, it must be split.** You must break the comparison into smaller multiple expressions. Try breaking it where an "And" was used or making a prior assignment for a complex mathematical expression so that only the name must appear in the comparison.
60	**Not Used**
61	**Invalid Integer or Integer Name.** The name used has not been defined yet or the number used is out of the Integer range of -32767 to 32768. You can define it with the "List" or Define Current Word" Function. Place the cursor on the name and press the F4 Key. All Data Elements must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.
62	**Expecting Integer.** The name used is defined as another variable type.
63	**Not Used**
64	**Expecting an Operand instead of.....** The information in the quotes should be an operator or a synonym of an operator.
65	**Expecting "=" or "<>" instead of.....** The operand "=" or "<>" or a synonym for one of them must appear in this type of statement.
66	**Not Used**
67	**Not Used**
68	**Expecting "GO" after "READ" Command.** A "GO" must follow a "READ" in the sentence so that the "READ" is not repeated inadvertently. For example, "Read User_Name, then go to Process_User_Name". This rule is due to the way a "READ" statement is handled by the controller and the trouble it can cause if this restriction is not imposed.
69	**Not Used**
72	**Too many conditional terms in Sentence.** Too many conditional terms were used in one statement. The statement in question should be broken into smaller statement sizes.
72	**String Constant too long.** The maximum for a string of characters enclosed in quotes is 512. Try breaking up the statement into smaller pieces.
73	**Can't open output file.** A problem has occurred accessing the disk to create the file that will get downloaded to the controller. Try the operation again.
74	**Unbalanced Parenthesis in Numerical Expression**. Parenthesis must come in matched pairs.
75	**Internal Flag overflow.** Too many Internal Flags have been used. ECLiPS uses them to create the code that comes from "Perform" and "For" statements. Therefore, you must reduce your use of either those types of statements or the Flags.
76	**Name too long.** All names may be us to 20 characters. They must start with a letter but may contain numbers and the "_" character. Upper and Lower case are treated equally.
77	**Number out of range.** The valid range for integer numbers is -32768 to 32767. The valid range for floating point numbers is +-1.1E-38 to 3.4E+38. Floating point numbers must have a decimal point followed by at least one digit. Floating point numbers may have a maximum of 7 significant digits, not counting the exponent.
78	**Power out of range, must be between -32767 and 32768.** When an exponent is used the power that the statement is raised to must be within the acceptable range.
79	**Successive Power Operators not allowed.** The exponent operator may not be used twice in a row.
80	**Expecting "(".** The statement used must have a "(" at the proper place.
81	**Not Used**.

82	**Maximum Number of Seconds is 600.00.** This is the maximum number of seconds that may be used. The resolution is 1/100 second and 60000 is the maximum value available.
83	**Mixed String and Character Types.** The value of a string may not be stored in a character variable and a character variable must be assigned a single character in single quotes.
84	**Invalid use of Wait Statement.** A "WAIT" may only be accompanied by a "GO" in a sentence and must appear before the "GO". This is due to the abuse this statement would endure if this restriction was not imposed. The "WAIT" statement is a simple timer that starts upon entering the State and is satisfied when the time is reached. It is intended to be used as a process delay.
85	**Expecting "GO" after "WAIT" command.** A "WAIT" must be followed by a "GO" and no other statements may appear in the same sentence. This is due to the abuse this statement would endure if this restriction was not imposed. The "WAIT" statement is a simple timer that starts upon entering the State and is satisfied when the time is reached. It is intended to be used as a process delay.
86	**Digital I/O and Flags not allowed in "WRITE".** The status of a Digital Point or Internal Flag may not be sent in a "WRITE" statement.
87	**Invalid PID Loop Name.** The name in the statement is not a valid PID Loop name. Try defining it as such.
88	**Not Used.**
89	**Invalid Element in Address Expression.** A valid numeric expression must appear after the "@" operator.
90	**Time Elements must be set individually.** Hour, Minute, and Second must be set individually as in "Hour = 13, Minute = 10".
91	**Time Value is out of range.** The value of time must be between 0 and 23:59.
92	**Time Value may contain only Hours and Minutes.** The time variable is accurate to only minutes, therefore it may not contain seconds.
93	**Internal String Overflow.** Too many Internal Strings have been used. ECLiPS uses them to create the code that comes from the "Perform" statements for Display_Date_&_Time, and User_Menu. Therefore, you must reduce your use of either those types of statements or the Strings.
94	**Invalid PID Loop Parameter Value.** An invalid value has been given to a PID Loop input or output. Use the "Define" or "List" function to enter valid values (Analog, Integer or Floating Point Variable) for all the input and output of each defined PID Loops.
95	**Missing PID Loop Parameter Name.** When performing a change to the parameters of a PID Loop the correct Parameter name must be used. A list of the PID Parameter names is available under the list function.
96	**Invalid PID Stop Value.** A PID Loop may be stopped only with a valid number.
97	**Not Used**
98	**Only one "GO" allowed per sentence.** . Since the "GO" is always placed at the very end of a sentence, the first one will be executed and no others would ever be executed.
99	**"HALT" and "GO" not allowed in the same sentence.** Since the first of the two found in the English would be executed and the other would not be seen, both are not allowed.
100	**Too Many Functional Terms in sentence.** The total number of allowable characters used by Functional statements in the sentence has exceeded the maximum. The sentence should be broken up.
101	**Ambiguous Sentence, Conditionals must be Grouped.** Two Conditional statements (For, If, Until, etc.) may not have a Functional statement in between them. The best way to avoid this error is to break up the sentence into smaller pieces and branch to another state if necessary.
102	**Project Translation Canceled by User.**
103	**Not Used**
104	**Floating Point Value Needs Leading Zero.** A Floating Point value must start with a number. ".1234" is not valid. You must place a leading zero on the value as follows: "0.1234".
105	**Invalid Bit Offset, Valid Range is 0 - 15.** When using the Set_Bit command, the value of anyone of the 16 bits in a register can be set. An attempt was made to address a bit that is outside of the 16 bits. The range of addressable bits is 0 to 15.

106	**Translator is out of Internal Integers.** The State Logic CPU needs to use some of the available Integers to run this program. In order for the program to run, some of the Integer Variable Definitions have to be deleted.
107	**Not Enough Memory for Translator to Continue.** Call Adatek Technical Support.
108	**Invalid High Speed Counter Parameter.** Does Not Apply to the 90 - 30 State Logic CPU.
109	**Bit Field Not Allowed in Write Term.** Displaying the status of a discrete variable is not allowed in a write term.
110	**Not Used.**
111	**Not Used.**
112	**Functional Other than Go Required with For.**
113	**State Too Long Translator Cannot Continue.** Does not apply to the 90 - 30 State Logic CPU
114	**Invalid Parameter Name**
115	**Missing Parameter.** A required parameter is missing from the Perform Function Call.
116	**Parameter Type Mismatch.** The actual parameter type does not match the type specified in the Perform Function Template.
117	**Too Many Parameters Found.** More actual parameters were found in the Perform Function Call than were specified in the Perform Function Template. A Perform statement must end in a period.
118	**Invalid PowerTRAC Monitor Parameter.** The word following the name of a PowerTRAC Monitor block must be a valid parameter name. Refer to the manual for the complete list of possible parameters.
119	**Specified Block is Not Defined.** The specified PCIM/BLOCK number is in the Custom I/O Scan Table, but the block is not defined. Either define the block, or remove the entry from the Custom Scan Table.
120	**Specified Block is Not Scanned.** The specified PCIM/BLOCK number is defined, but has not been included in the Custom I/O Scan Table. Either delete the block, or add the entry to the Custom Scan Table.
121	**Specified Block is Not Analog.** The specified PCIM/BLOCK number is defined in the Custom I/O Scan Table, and is not an Analog Block. Only analog blocks are allowed in the table.
122	**Specified Point is Not Scanned.**
123	**Not Used.**
124	**Not Used.**
125	**Invalid register length.** The R_Register length is out of range. Valid lengths are 1 to 40. The length + the start register must also be within the valid range of R_Register numbers.
126-129	**Not Used.**
130	**Too Many Variables of type.** When retentive variables and non-retentive variables of the same type are used. The total of retentive variables + non-retentive variables of that type must be at least 1 less than the system total for that type of variable. For example, the system allows 1000 integers, if 13 retentive integers are used, only 986 non-retentive integers are allowed, not 987.
131	**Expecting Colon.** Every Task must start with the following syntax: "Task: Task_Name" Every State must start with the following syntax: "State: State_Name" Blanks are allowed after the ":", but not before.
134	**PID Bit Field in Make Statement.** Pid Control and Status Bits cannot be used in conjunction with the MAKE keyword. These bits must be set and cleared with the SET_BIT and CLEAR_BIT keywords.
135	**Translator Halted - Maximum number of errors exceeded.** The translator has found the maximum number of errors that are allowed. Use the FIND List Translator Errors or <ALT><F7> to identify the other errors in the translation than attempt to retranslate the project.

Runtime Errors

These errors occur during program execution. The State Logic controller responds differently depending on the error and the CPU error response configuration.

The State Logic controller sends a message out the serial port to ECLiPS or OnTOP for each runtime error detected.

There are two categories of runtime errors:

- Non-Critical
- Critical

The CPU can be configured to either continue running (Diagnostic) or halt when either of the errors occurs. In addition the faults are logged in the fault table and some errors are displayed in the event queue.

Non-Critical Errors

Integer Overflow - Value out of the integer range (-32768 to 32767) was assigned to an integer variable

Floating Point Overflow - A value outside the allowed floating point range was assigned to a floating point variable

Divide by zero

Faults and Alarms- ***See the Chapter on Online Functions for a detailed discussion of the fault handling System***

Perform Function Errors as Listed Below:

- **BCD Function Errors:**

  BCD in called with invalid variable type: The variable type used to store BCD data must be Float or Integer

  BCD output called with too few parameters: The minus sign pattern and null character parameter must be defined

  Too large a number to output in BCD allocated: There are not enough digits to show all of the numbers to the left of the decimal point

  BCD output called with invalid variable type: The variable type used to store BCD data must be Float or Integer

  Define Table Errors:

  Error Define_table: called with too large a table number : Table numbers must be from 1 to 100

  Error Define_table: table all ready defined: Table number all ready defined the State Logic CPU will use the first set of table definitions and ignore second definitions

Error Define_table: called with illegal table type : Table Type must be: I for integer, F for Float, S for string, or D for digital data type.

Error Define_table: called with illegal save_over_halt: Save over halt cannot be used with this data type.

- **Copy Table Errors**

  Error Copy_table_to_table: called with too large a table number: The table number must be a value from 1 to 100.

  Error Copy_table_to_table: called with non_defined table: The tables used must be defined

  Error Copy_table_to_table: mis_matched table types: Both tables must be of the same type

  Error Copy_table_to_table: table to be copied larger than the other table: The table to be copied must be smaller than the table it will be copied into.

- **Init Table Errors**

  Init_table_digital: called with mismatched number_of_values: The number of Number_of_values parameter must be equal to the number of values specified in the parameter table

  Init_table_digital: called with too large a table number: The table number must be a legal value for the platform in use.

  Error Init_table_digital: called with non_defined table: The table must be defined before an Init table option can be called.

  Init_table_digital: called with non_digital table: The table type must match the type of perform used

  Init_table_digital: called with row number out of range: The row_number parameter must be a value that is valid for the table

  Init_table_digital called with column number out of range: the column_number parameter must be a value that is valid for the table.

  Init_table_digital called with too many values: The maximum no. of values that can added for each perform is 28.

- **Swap Table Function Errors**

  Error Swap_table_value_dig: called with too large a table number: The table number must be a value from 1 to 100.

  Error Swap_table_value_dig: called with non_defined table: The table must be defined before a swap perform function is performed.

  Error Swap_table_value_dig: called with non_digital table: The table type must match the type of perform used

  Error Swap_table_value_dig: called with row number out of range: The row_number parameter must be a value that is valid for the table

Error Swap_table_value_dig: called with column number out of range: the column_number parameter must be a value that is valid for the table.

Error Swap_table_value_int: called with illegal swap type: The variable that is swapping the data must be of the same type as the table

- ## Shift Register Function Errors

Error shift_reg par (followed by the bad parameter number): There was an error in the way the specified parameter number was used

- ## String Manipulation Function Errors

Error String_manipulation starting_character parameter: The staring_character parameter must be a value that is within the range of the string

Error String_manipulation ending_character parameter: The ending_character parameter must be a value that is within the range of the string.

Error String_manipulation integer too big: The integer value being extracted from the string is outside of the range -32767 to 32767.

Error String_manipulation not ASCII integer value: the value must be an ASCII integer to extract as an integer.

Error String_manipulation not ASCII float value: the value must be an ASCII floating point to extract as a floating point value.

Error String_manipulation concatenated string too long: The string to be added to the string value is too long.

Error String_manipulation string too long: String lengths are limited to 80 characters

Error String_manipulation not enough digits for integer: The number of digits being used must be equal to the number of spaces in the string

Error String_manipulation not enough digits for float: The number of digits being used must be equal to the number of spaces in the string

Operation not a valid character: When performing a String_Manipulation, the following characters are used to define the operation: s for store string, E for extract string, i for save integer, I for extract integer, f for save floating point, F for Extract floating point, C for Concatenate, L for string length, or M for match character. The operator is case dependent.

- ## Time Counter Function Errors

Error Time_counter: called with wrong number of parameters: When assigning a time counter, the time increment H for hour, M for minute, S for second, T for tenths must be specified.

Error Time_counter: no tenth of second counters available: The maximum number of time counters that can be used at one time is 30.

Error Time_counter: no second counters available: The maximum number of time counters that can be used at one time is 30.

Error Time_counter: no minute counters available: The maximum number of time counters that can be used at one time is 30.

Error Time_counter: no hour counters available: The maximum number of time counters that can be used at one time is 10.

Error Time_counter: integer variable has already been assigned: The integer variable being assigned has all ready been assigned to a time counter.

Error Time_counter: integer variable has not previously been assigned: The time counter integer variable must be assigned before being enabled or halted.

Critical Errors

Invalid State Logic CPU Instruction

Attempt to Change a Task to an Undefined State

Appendix E Specifications

Platform Number	311/313/323	331/340
Tasks	256	256
Task Groups	16	16
States	3000	3000
States Per Task	254	254
Statements per State	Unlimited	Unlimited
Integer Variables (range -32768 to +32767)	250	1000
Floating Point Variables 32 bit IEEE Format	61	497
String Variables	8	20
String Variable Size	80 Characters	80 Characters
Character Variables	64	64
PID Loops	20	20
Internal Flags	500	1000
Global Digital Circuits % G	1280	1280
Digital Circuits % I or % Q	512	1024
Maximum Number of Tables	10	20
Maximum Table Memory	1 K	4000 Bytes
Analog Output Circuits %AQ	64	128
Analog Input Circuits %AI	128	256
Timers	Unlimited	Unlimited
Timer Resolution	1/100 second	1/100 second
Characters per Write Instruction	512	512
Write Instructions per State	32	32
State Changes Listed in Trace Display	100	100
Force Table Size	32 Digital 32 Analog	32 Digital 32 Analog
Monitor Table Size	6 entries	6 entries
Monitor Tables	10	10
I/O and Variable Names	3000	3000

%

%G, 7-9
%G Bits, 3-7

A

Actuate, 7-10, 7-11
Add, 7-12, C-2
Add Menus, 6-2
Alarm, 10-10
AM, C-5
Analog, 3-6, 3-8, 6-10
Analog, 7-6, 7-11, 7-19
Analog Modules, 2-15
Analog Scaling, 6-11
And, 7-18, C-4
And, 7-20
Anti-Reset Windup, 9-4, 9-11
APM, 3-19
ARCTAN, C-3
ASCII, 3-24, 7-5
ASCII Communications, 11-4
ASCII Control Characters, 7-15
ASCII Variable, 7-10
Assignment Term, 7-21
Auto Echo, 11-6
Autoexec.bat, 2-9
Automatic Backup, 6-6
Axis Positioning Module, 3-19

B

Backup, 6-6, 10-14
Baseplates, 2-4
Battery, 2-4
Battery Fault, 10-9
Baud Rate, 11-6
BCD, 8-19
Bitwise_And, C-4
Bitwise_Or, C-4
Block Editing Operations, 6-2
Block_Down, 9-5, 9-9
Block_Up, 9-4, 9-9
Building Global Data Blocks, 8-7
Bumpless Transfer, 9-3

C

Calculations, 7-21
Cascaded PID, 9-4
CCM, 2-4, 3-7, 3-21, 3-24, 11-8
CCM Communications, 11-3
CCM Digital Data, 11-14
CCM Master, 3-10
CCM Number, 11-14
CCM Protocol, 10-3
CCM Protocol Listing, 6-8
CCM Protocol Mapping, 11-8
CD Manual, 1-2
CD ROM, 1-2
Change, 10-13
Change Path, 6-5
Changing Active State, 7-12
Character Variables, 5-9, 7-10
Clear_Bit, 7-12, C-2
Clock, 7-7
Clock Functions, 10-6
Clock Variables, 7-7
Clock/Calendar, 7-22, 10-13
CMM, 3-7
Comm_Request, 3-5, 3-10, 3-13, 3-15, 3-22
Command Bit, 9-11
Comments, 5-19
Communications Coprocessor Module, 3-7
Communications port, 7-20
Communications Request, 8-13
Computer Requirements, 2-6
Conditional Expression, 7-3
Conditional Term, 4-7, 5-6, 7-18, C-1
Config.sys, 2-7, 2-9, 2-10
Configuration, 3-1
Configuration, 2-12
Configuration Documentation, 6-8
Configuring PC Memory, 2-8
Control Characters, 7-16
Control Variable, 9-8
Copy, 6-3
Copy Projects, 6-5
Copy_Var_To_Integer, 8-7
COS, C-3
CPU, 2-12
CPU, 2-2
CPU Configuration, 10-12
Critical Errors, 2-13, D-10
Current State, 4-11, 7-12, 7-20
Cursor Control Keys, 6-4

D

Data Type Conversions, 8-7
Datagrams, 3-14
Date, 7-22

Day, 7-7
Day_of_Week, 7-7
Debug Mode, 2-5, 10-1
Define, 6-11
Define Menu, 6-11
Define_Table, 8-3
Diagnostics, 4-11
Diagnostics, 4-4
Digital, 7-18
Divide, C-2
Divide, 7-12
Documentation, 5-18, 6-7
DOS, 2-9
Double Integer, 8-10
Download, 6-9, 10-4

E

Eclips, 2-5
Eclips Lite, 2-5
ECLiPS Port, 2-13
ECLIPS.KWD File, 6-9
Editor, 6-2
EEPROM, 10-5
End of Message Character, 11-6
Energize, C-2
Error Check, 6-9
Escape Characters, 7-16
Ethernet Interface Module, 3-11
Event Queue, 10-8
Expanded memory, 2-8
Exponent, C-3
Expression, 7-2
Extended Memory, 2-8

F

Fault, 2-13, D-7
Fault System, 10-8
Fault Table, 10-10
Fault Variables, 7-8, 10-11
File Types, 6-9
Filler Words, 5-7, 7-4, 7-23
Find, 6-3
Finite State, 4-2, 4-4
Floating Point, 3-21, 7-4
Floating Point Variables, 5-8, 7-6
Flow Control, 11-6
Force, 10-13
Foreign Modules, 3-33
Formatting, 7-15
Friday, C-5
From, C-5
From, 7-20
Function Keys, A-1
Functional Expression, 7-3, 7-23
Functional Term, 4-7, 5-6, 7-10, C-1
Fundamentals, 1-3

G

Gain, 9-2
GBC, 3-14
GCM+, 3-5
GE Fanuc Manuals, 1-2
Genius Bus Controller, 3-14
Genius Communications, 3-5
Global Data, 3-5, 3-14, 3-18, 7-9
Global Data Blocks, 8-7
Go, 7-3, 7-23, 11-7, C-2
Go, 7-12
Grammatical Rules, 7-23

H

Halt, C-2
Hardware Key, 1-2, 2-7
Help, 1-2, 6-2
Help Index, 6-2
Hierarchy, 7-2
High Speed Counter, 3-2, 8-15
High_Limit_Status, 9-9
Hints, 5-18, 10-15
Hints, 5-12
Hot Key, 6-2, 6-3, 6-13, A-2
Hotline, 1-2
Hour, 7-7

I

I/O Modules, 2-4, 2-14
If, C-1
Import, 6-6
Inactive, C-5
Inactive State, 7-14
Insert, 6-2
Installation, 2-6, 2-10
Integer, 7-4
Integer Variables, 5-8, 7-6
Internal Flags, 5-8, 7-9

K

Key, 1-2, 2-7
Keywords, 5-7, 7-4, C-1

L

Language, 7-4
List, 6-12
List Menu, 6-3
LN, C-3
Logging Data, 10-3
Low_Limit_Status, 9-9

M

Make, C-2
Make, 7-10
Make Term, 7-11
Manual/Auto station, 9-3
Manuals, 1-2
Master, 3-10
Math Assignment, 7-11
Mathematical Calculations, 7-21
Max_Time, C-5
Max_Time, 7-3
Memory, 2-9
Memory, 2-8
Memory Usage, 10-8
Mini-Converter Kit, 2-11
Minute, 7-7
Model, 4-4
Monday, C-5
Monitor Tables, 10-6
Month, 7-7
Motion Control, 3-19
Move, 6-3
Multiply, C-2
Multiply, 7-12

N

Names, 2-8, 2-14, 5-4, 5-18, 6-10, 7-4
Natural Logarithm, C-3
Non-Critical Errors, 2-13
Non-Critical Errors, D-7
Non-Critical Run-time Error, 10-12
Not, C-4, C-5
Not, 7-20
Notation Conventions, 1-6
Numeric Data Types, 7-4
Numeric Variable, 7-6
Numerical Expressions, 7-22

O

Off, C-5
On, C-5
On-Line, 1-3
Online Modify, 6-12
Online Program Changes, 6-12
OnTOP, 2-5, 2-6, 2-9, 6-9, 7-21, 10-1
OnTOP, 7-15, A-1
Operator Precedence, 7-22
Operators, C-1
Or, 7-18, C-4
Or, 7-20
Or, C-5
Over Type, 6-2

P

Parallel Port, 2-7
Parenthesis, 7-22
Password, 2-11
Paste, 6-3
Paste, 6-14
PC Requirements, 2-6
PCM, 3-21
Perform, 7-10, C-2
Perform Function Parameters, 8-2
Perform Functions, 7-17, 8-1
PID Initialization, 9-5
PID Loop, 7-17
PID Loop Documentation, 6-8
PID Loop Tuning, 9-7
PID Loops, 2-13
PLC, 2-2
PM, C-5
Power On Behavior, 10-11
Power Supply, 2-4
Power Up, 2-13
Precedence, 7-22
Print, 6-7
Printer, 2-7
Process Variable, 9-8
Program Files, 2-9, 2-10
Program Mode, 2-5, 6-2
Program Scan, 5-9
Programmable Coprocessor Module, 3-21
Programming, 1-3
Programming Port, 11-2

Programming Port, 2-4
Project Menu, 6-4

Q

Quick Reference, 7-10, 7-18

R

Racks, 2-13
Random, C-3
Rate, 9-2
Read, 3-24, 7-23, C-1
Read, 7-20
Read Instruction, 11-7
Read Term, 5-16
Receiver Always On, 11-6
Reference Number, 2-14, 6-10
Registration, 1-2
Registration Form, 1-2
Relational Term, 7-19
Remove, 6-3
Reset, 9-2
Respond to Backspace, 11-6
Resume_Task, 7-14, C-2
Retrieve, 6-5
RS-232, 3-24
RS-485, 3-24
RS-485 Communications, 3-25, 3-28
RTU, 3-7
Runtime Errors, 2-13, D-7

S

Saturday, C-5
Save, 6-5
Save Over Halt, 6-10, 7-5
Scan Rate, 10-8
Scan Time Optimization, 7-13
SCM, 3-24
SCM Hardware Handshaking, 3-26, 3-28, 3-31
SCM Serial Cable, 3-26
Search and Replace, 6-3
Second, 7-7
Seconds, 7-19, C-5
Security, 2-9
Security, 2-11
Send_Hsc_Data, 3-5, 8-15
Serial, 7-15
Serial Cable, 2-11, 3-28
Serial Communication, 4-8, 7-21
Serial Communications, 1-3
Serial Communications Module, 3-24, 11-5
Serial Communications Programming, 11-7
Serial Default Settings, 11-4
Serial Devices, 2-4
Serial I/O, 2-4
Serial Parameter Changes, 11-5
Serial Port, 2-4, 2-11
Serial Port, 7-17
Serial Port Parameters, 11-4, 11-6
Set_Bit, 7-12, C-2
Set_Commport, 11-5, C-2
Set_Commport, 7-17
Setpoint, 9-8
Setting Up Hardware, 2-6
Setup, 1-3
Setup Menu Option, 2-8
Shift Register, 8-18
Simulation, 10-4
SIN, C-3
SNP, 2-4, 3-7, 3-24
SNP Analog Data, 11-12
SNP Communications, 11-3
SNP Digital Data, 11-14
SNP ID Number, 3-9
SNP Master, 3-10
SNP PID Data, 11-11
SNP Protocol Listing, 6-8
SNP Protocol Mapping, 11-8
SNP Register Data, 11-9
Specification, 2-3
Square Root, C-3
Start_In_Last_State, 7-3, C-5
Start_PID, 7-10, 7-17, C-2
State, 5-2, 7-2, 7-3
State, 4-2, 7-12, 7-23, C-5
State Changes, 7-14, 10-13
State Diagram, 4-2
State Logic Control, 2-12
Statement, 4-5, 5-3, 7-2, 7-3
Statement, 7-23
States, 2-8
Status Bits, 9-11
Stop Bits, 11-6
Stop Transmit on Receive, 11-6
Stop_PID, 7-10, 7-17, 9-5, C-2
String Manipulation, 8-10
String Variables, 5-8, 7-10, 11-11
Subtract, 7-12, C-2
Sunday, C-5

Suspend_Task, 7-14
Suspend_Task, C-2
Synonyms, 5-7, C-1
System Configuration, 2-12, 11-7
System Status, 10-8
System Variable, 10-13

T

Table Functions, 8-3
Tangent, C-3
Task, 4-4, 4-5, 4-9, 5-2, 7-2, 7-3, C-5
Task, 4-3, 5-14, 7-23
Task Groups, 6-9, 7-2
Task Interaction, 4-10
Technical Support, 1-2
Term, 7-2
Terminal Log, 10-2
Test_Bit, 8-18
Text Menu, 6-3
Thursday, C-5
Time, 7-22
Time Counter, 2-13, 8-13
Time Variables, 5-9, 7-7
Timer, 5-16, 7-19
Trace, 6-8, 10-9
Track_Mode, 9-9
Track_Mode, 9-4
Training, 1-2
Translate, 6-9
Translation Errors, D-1
troubleshoot, 10-15
Tuesday, C-5
Tuning Parameters, 9-10
Turn On, 7-11

U

Update Time, 9-8

V

Variables, 5-7, 6-10, 7-5, 7-11
Version, 10-8, 10-14
Version, 2-9, 6-6
View, 10-7

W

Wednesday, C-5
When, C-1
Windows Nt, 2-8
With, C-5
With, 9-5
Word, 7-2
Write, 3-24, C-2
Write, 7-10, 7-15
Write Instruction, 11-7
Write Term, 5-15, 10-3

X

XON/XOFF, 11-6