GE
Intelligent Platforms

State Logic Products

ECLiPS English Control Language Programming System For Series 90-70 PLC

User's Guide

GFK-0731B *March* 2010

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 Intelligent Platforms assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Intelligent Platforms 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.

* indicates a trademark of GE Intelligent Platforms, Inc. and/or its affiliates. All other trademarks are the property of their respective owners.

©Copyright 2010 GE Intelligent Platforms, Inc.
All Rights Reserved

Contact Information

If you purchased this product through an Authorized Channel Partner, please contact the seller directly.

General Contact Information

Online technical support and GlobalCare	http://www.ge-ip.com/support
Additional information	http://www.ge-ip.com/
Solution Provider	solutionprovider.ip@ge.com

Technical Support

If you have technical problems that cannot be resolved with the information in this guide, please contact us by telephone or email, or on the web at www.ge-ip.com/support

Americas

Online Technical Support	www.ge-ip.com/support
Phone	1-800-433-2682
International Americas Direct Dial	1-780-420-2010 (if toll free 800 option is unavailable)
Technical Support Email	support.ip@ge.com
Customer Care Email	customercare.ip@ge.com
Primary language of support	English

Europe, the Middle East, and Africa

Online Technical Support	www.ge-ip.com/support
Phone	+800-1-433-2682
EMEA Direct Dial	+352-26-722-780 (if toll free 800 option is unavailable or if dialing from a mobile telephone)
Technical Support Email	support.emea.ip@ge.com
Customer Care Email	customercare.emea.ip@ge.com
Primary languages of support	English, French, German, Italian, Czech, Spanish

Asia Pacific

Online Technical Support	www.ge-ip.com/support
Phone	+86-400-820-8208
	+86-21-3217-4826 (India, Indonesia, and Pakistan)
Technical Support Email	support.cn.ip@ge.com (China)
	support.jp.ip@ge.com (Japan)
	support.in.ip@ge.com (remaining Asia customers)
Customer Care Email	customercare.apo.ip@ge.com
	customercare.cn.ip@ge.com (China)

Content of this Manual

Chapter 1. Getting Started – This chapter has general foundational information about the ECLiPS software product. There is a general overview of the product, instructions on using this manual, installation procedures, hardware requirements, and sources of information about using ECLiPS.

Chapter 2. 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.

Chapter 3. Creating A Control Program – This chapter 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.

Chapter 4. Programming Tutorial – This chapter presents a step by step procedure for writing a control program for a specific application. Follow along duplicating each keyboard entry on your own computer. Watch for the text displayed in **bold italics**, these are the lines that you should enter.

Chapter 5. Helpful Hints – The hints and suggestions found in this chapter have been developed by ECLiPS and State Language programmers in an effort to make the user's first programming experience as productive as possible. There are two parts to this chapter, one giving hints for programming and the other has suggestions for using the tools provided as part of ECLiPS.

Chapter 6. Online Tutorial – This chapter is a tutorial that explains the steps necessary to get a control program running in the State Logic Processor. This chapter covers configuring the Series 90–70 CPU with Logicmaster 90 and how to load, execute, monitor, and interact with an ECLiPS control program.

Chapter 7. Creating Program Documentation – This chapter describes how to use the ECLiPS program documentation features. The topics covered are printing program documentation, including the English program, I/O map, Data Listing, Task and State listing, and Cross Reference Listing. Printing information for using the CCM protocol and documenting the ECLiPS program are also covered.

Chapter 8. Reference – This chapter is designed to provide information about the details of the ECLiPS control language and how to use the ECLiPS software package. The chapter starts with a description of the State Logic Language and Keywords then discusses the setup of the Series 90–70 control system, then each of the menu options is explained, and finally the ECLiPS specifications are displayed.

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

We Welcome Your Comments and Suggestions

At GE Intelligent Platforms 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

Chapter 1 Getting Started 1-1

Overview 1-1
How to Use this Manual 1-2
Brief Description of the Manual Sections 1-2
Hardware Requirements 1-3
Installation 1-3
Getting Help 1-4
Register Your Product 1-4

Chapter 2 State Logic Control Theory 2-1

State Logic Control Theory 2-1
The Concept of Finite States 2-1
What Makes State Control Logic Different 2-4
A Collection of Tasks is a State Logic Program 2-4
Developing State Logic Programs with ECLiPS 2-5
Tasks – Sequences of States 2-6
States – The Building Blocks of a Task 2-6
Statements – The Command Set for State Descriptions 2-7
Communication Functions 2-8
Scan Overview 2-9
Interaction Between Tasks 2-9
Creating Process Diagnostics 2-11

Chapter 3 Creating A Control Program 3-1

Outline the Application 3-1
Identify the Tasks 3-1
Identify The States 3-2
Identify the Statements 3-3
Writing The Program 3-3
Using English Names in the ECLiPS Program 3-4
Statement Structures 3-5
Constructing Statements 3-6
Using Keywords, Synonyms and Filler Words 3-6
Using Variables 3-7
Program Scan 3-9

Contents

Chapter 4 Programming Tutorial 4-1

Tutorial Procedure 4-1
The Control Application 4-2
Outline the Application 4-3
Identify the Tasks 4-3
Identify the States 4-3
Identify the Statements 4-6
Writing the Program 4-8
Identify the I / O 4-8
The Basics of ECLiPS 4-9
Program the Fill Station Task 4-11
Define Undefined Words 4-14
Checking Your Work and Making Changes 4-16
Program the Mix Station Task 4-17
Conveyor 4-18
Advanced Programming Terms and Diagnostics 4-19

Chapter 5 Helpful Hints 5-1

Programming Hints 5-1
Outputs are OFF by Default 5-1
Write Term Considerations 5-2
Calculations and a Scanning Operating System 5-3
Task Design 5-3
Read Term Considerations 5-4
Timer Considerations 5-5
Documentation Hints 5-6
Hints for Using ECLiPS Features 5-10
How to Use the ECLiPS Menus 5-10
Using ECLiPS Hot Keys 5-10
ECLiPS Word Processing Functions 5-10
How to Use ECLiPS Lists 5-11

Chapter 6 Online Tutorial 6-1

Setting Up the Series 90–70 State Logic Control System 6-1
Configuring the State Logic Processor 6-1
Enter the Initialization Ladder Program 6-2
Enable Outputs and Start Program Running 6-3
DownLoad a Program 6-3
Debug Mode Screen 6-4
Put SLP in Run Mode 6-5
Toggle Simulation Mode 6-5
RUN or HALT Program 6-5
Controlling and Observing the Fill Station 6-5
Monitor 6-5
View 6-6
Controlling and Observing the Mix Station 6-6
Change 6-6
Display 6-6
Force 6-7
Trace 6-7
PLC I/O 6-7
PID Loop Tuning Screen 6-7
Viewing and Clearing CPU Faults 6-7

Chapter 7 Creating Program Documentation 7-1

Program Documentation Features 7-1
Header/Footer 7-2
Directing the Output 7-2
Documentation Options 7-2
Commenting the ECLiPS Program 7-8

Chapter 8 Reference 8-1

Language Description 8-1
Program Structure 8-1
Language Structure Notational Conventions 8-2
Functional Expressions 8-3
Conditional Expressions 8-9
Mathematical Calculations 8-13
Variables 8-13
Language Structure Summary 8-16
Grammatical Rules 8-19
Using the System Clock 8-19
Standard Predefined Keyword Set 8-20
Filler Words 8-24
PID Loops 8-24
Perform Functions 8-32
Perform Function Forms 8-33
Specialized Perform Functions 8-44
Keyboard Definitions 8-45
Series 90–70 State Logic Control System 8-46
SLP / CPUInterface 8-46
How to Setup Your System 8-54
Serial Ports 8-58
Analog Scaling and Update Rates 8-63
State Engine Scan Considerations 8-63
Other State Engine Setup Options 8-68
Making a Permanent Copy of the Terminal Log 8-68
Simulation Mode 8-68
Setting the System Clock 8-69
ECLiPS Menu System 8-69
Program Mode 8-69
Debug Mode 8-81
Online Modify 8-88
Setup and ECLIPS Memory Usage 8-89
Quit 8-90
Specifications 8-90

Figure 2-1. State Diagrams 2-2
Figure 3-1. Program Scan 3-9
Figure 3-2. Statement Scan 3-10
Figure 4-1. Tutorial Application Drawing 4-2
Figure 4-2. Main Menu 4-10
Figure 4-3. Project Menu 4-11
Figure 4-4. Conveyor Task 4-18
Figure 5-1. Program Scan 5-7
Figure 5-2. Statement Scan 5-8
Figure 5-3. Program Scan with GO Terms 5-9
Figure 5-4. Statement Scan with GO Terms 5-9
Figure 6-1. Sample Logicmaster Configuration Screen 6-2
Figure 6-2. Debug Mode Screen 6-4
Figure 8-1. PID Algorithms 8-25
Figure 8-2. Cascaded PID Loops 8-26
Figure 8-3. SLP / CPU Interfaces 8-49
Figure 8-4. Series 90–70 State Logic Setup Decision Tree 8-54
Figure 8-5. Program Scan 8-65
Figure 8-6. Statement Scan 8-66
Figure 8-7. Program Scan with GO Terms 8-67
Figure 8-8. Statement Scan with GO Terms 8-67
Figure 8-9. List Menu 8-71
Figure 8-10. PID–Loop Tuning Screen 8-87

Table 8-1. State Engine Data Types and CCM References 8-59
Table 8-2. PID Parameter Table 8-62

GE Intelligent Platforms North America, Inc.
Software License Agreement

GFK-0731B

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THIS PACKAGE. OPENING THIS PACKAGE SIGNIFIES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED ALONG WITH ANY OTHER ITEM THAT WAS INCLUDED IN THE SAME CATALOG NUMBER FOR FULL CREDIT.

You, as the Customer, agree as follows:

1. DEFINITIONS

"Application Software" shall mean those portions of the Licensed Software, in object code form only, created by GE.

"Designated Computer" shall mean the one (1) computer upon which Customer shall run the Licensed Software.

"Licensed Software" shall mean the Application Software plus any other software, in object code form only, supplied by GE pursuant to this Agreement. The Licensed Software may include third party software, including but not limited to operating systems, licensed to GE. If no operating system software is included in the software provided under this Agreement, you must make provision for any required operating system software licenses.

2. LICENSE

2.1 Except as provided in section 2.2 below, you are granted only a personal, non–transferable, nonexclusive license to use the Licensed Software only on the Designated Computer. You may copy the Licensed Software into machine readable form for backup purposes in support of your use of the Licensed Software on the Designated Computer, limited to one copy. No other copies shall be made unless authorized in writing by GE. You may not reverse compile or disassemble the software. The Licensed Software, comprising proprietary trade secret information of GE and/or its licensors, shall be held in confidence by Customer and protected from disclosure to third parties. No title to the intellectual property is transferred. You must reproduce and include all applicable copyright notices on any copy.

2.2 If you are an authorized GE distributor or an Original Equipment Manufacturer who incorporates the Licensed Software into your equipment for sale to an end user, you may transfer the Licensed Software to an end user provided that the end user agrees to be bound by the provisions of this Agreement.

2.3 GE licensors having a proprietary interest in the Licensed Software shall have the right to enforce such interests, including the right to terminate this Agreement in the event of a breach of its terms pertaining to such proprietary interests.

2.4 EXCEPT AS PROVIDED IN SECTION 2.2 ABOVE, IF YOU TRANSFER POSSESSION OF ANY COPY OF THE LICENSED SOFTWARE TO ANOTHER PARTY WITHOUT WRITTEN CONSENT OF GE YOUR LICENSE IS AUTOMATICALLY TERMINATED. Any attempt otherwise to sublicense, assign or transfer any of the right, duties or obligations hereunder is void.

2.5 If the Licensed Software is being acquired on behalf of the U.S. Government, Department of Defense, the Licensed Software is subject to "Restricted Rights", including the legend to be affixed to the software as set forth in DOD Supplement to the Federal Acquisition Regulations (DFAR's) paragraph 252.227–7013(c)(1). If software is being acquired on behalf of any other U.S. Government entity, unit or agency, the Government's rights shall be as defined in paragraph 52.227–19(c)(2) of the Federal Acquisition Regulations (FAR's).

3. WARRANTY

3.1 GE warrants that the Application Software will be in substantial conformance with the specifications in the manual pertaining thereto as of the date of shipment by GE. If, within ninety (90) days of date of shipment, it is shown that the Application Software does not meet this warranty, GE will, at its option, either correct the defect or error in the Application Software, free of charge, or make available to Customer satisfactory substitute software, or, as a last resort, return to Customer all payments made as license fees and terminate the license with respect to the Application Software affected. GE does not warrant that operation of the Application Software will be uninterrupted or error free or that it will meet Customer's needs. All other portions of the Licensed Software are provided "as is" without warranty of any kind.

3.2 WITH RESPECT TO THE SOFTWARE WHICH IS THE SUBJECT OF THIS AGREEMENT, THE FOREGOING WARRANTIES ARE EXCLUSIVE AND ARE IN LIEU OF ALL OTHER WARRANTIES WHETHER WRITTEN, ORAL, IMPLIED OR STATUTORY NO IMPLIED OR STATUTORY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE SHALL APPLY.

4. LIMITATION OF LIABILITY

4.1 IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE) OR OTHERWISE SHALL GE OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PENAL DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFIT OR REVENUES, LOSS OF USE OF THE LICENSED SOFTWARE OR ANY PART THEREOF, OR ANY ASSOCIATED EQUIPMENT, DAMAGE TO ASSOCIATED EQUIPMENT, COST OF CAPITAL, COST OF SUBSTITUTE PRODUCTS, FACILITIES, SERVICES OR REPLACEMENT POWER, DOWN TIME COSTS, OR CLAIMS OF CUSTOMER'S CUSTOMERS AND TRANSFEREES FOR SUCH DAMAGES EVEN IF GE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4.2 EXCEPT AS PROVIDED IN SECTION 5, INDEMNITY, IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT OR WARRANTY, TORT (INCLUDING NEGLIGENCE) OR OTHERWISE, SHALL GE LIABILITY TO CUSTOMER FOR ANY LOSS OR DAMAGE ARISING OUT OF, OR RESULTING FROM THIS AGREEMENT, OR FROM ITS PERFORMANCE OR BREACH, OR FROM THE LICENSED SOFTWARE OR ANY PART THEREFORE, OR FROM ANY SERVICE FURNISHED HEREUNDER, EXCEED THE QUOTED CHARGES FOR THE LICENSED SOFTWARE. ANY SUCH LIABILITY SHALL TERMINATE UPON THE TERMINATION OF THE WARRANTY PERIOD AS SET FORTH IN SECTION 4.

4.3 If GE furnishes Customer with advice or other assistance which concerns Licensed Software or any portion thereof supplied hereunder or any system or equipment on which any such software may be installed and which is not required pursuant to this Agreement, the furnishing of such advice or assistance will not subject GE to any liability, whether in contract, warranty, tort, (including negligence) or otherwise.

4.4 The products to be licensed or sold hereunder are not intended for use in any nuclear, chemical or weapons production facility or activity, or other activity where failure of the products could lead directly to death, personal injury or severe physical or environmental damage. If so used, GE disclaims all liability for any damages arising as a result of the hazardous nature of the business in question, including but not limited to nuclear, chemical or environmental damage, injury or contamination, and Customer shall indemnify, hold harmless and defend GE, its officers, directors, employees and agents against all such liability, whether based on contract, warranty, tort (including negligence), or any other legal theory, regardless of whether GE had knowledge of the possibility of such damages.

5. INDEMNITY

5.1 GE warrants that the Application Software shall be delivered free of any rightful claim for infringement of any United States patent or copyright. If notified promptly in writing and given authority, information and assistance, GE shall defend, or may settle, at its expense, any suit or proceeding against Customer so far as based on a claimed infringement which would result in a breach of this warranty and GE shall pay all damages and costs awarded therein against Customer due to such breach. In case the Application Software is in such suit held to constitute such an infringement and its use is enjoined, GE shall, at its expense and option, either procure for Customer the right to continued use, or replace same with a non–infringing product or part, or modify the Application Software so that it becomes non–infringing, or remove the software and refund the license charge pertaining thereto (less reasonable depreciation for any period of use) and any transportation costs separately paid by Customer. The foregoing states the entire liability of GE for patent and copyright infringement by the Licensed Software or any part thereof.

5.2 The indemnity under the preceding paragraph shall not apply to any use of Application Software in conjunction with any other product in a combination not furnished by GE as a part of this transaction. As to any such use in such combination, GE assumes no liability whatsoever for patent and copyright infringement and Customer will hold GE harmless against any infringement claims arising therefrom.

6. TERM AND TERMINATION

6.1 You may terminate the license granted hereunder at any time by destroying the Licensed Software together with all copies thereof and notifying GE in writing that all use of the Licensed Software has ceased and that same has been destroyed.

6.2 GE, upon thirty (30) days notice, may terminate this Agreement or any license hereunder if Customer fails to perform any obligation or undertaking to be performed by it under this Agreement or if Customer attempts to assign this Agreement without the prior written consent of GE. Within twenty (20) days after any such termination of this Agreement, Customer shall certify in writing to GE that all use of the Licensed Software has ceased, and that same has been returned or destroyed, in accordance with GE instructions.

6.3 Sections 4, 6 and 7 of this Agreement shall survive any expiration or termination and remain in effect. Termination of this Agreement or any license hereunder shall not relieve Customer of its obligation to pay any and all outstanding charges hereunder nor entitle Customer to any refund of such charges previously paid.

7. EXPORT

7.1 If you intend to export (or reexport), directly or indirectly, the software products or technical information relating thereto supplied hereunder or any portion thereof, it is your responsibility to assure compliance with U.S. export control regulations and, if appropriate, to secure any required export licenses in your own name.

8. GENERAL

8.1 This Agreement shall be governed by the laws of the State of Virginia, without regard to its conflict of law provisions. The provisions of the United Nations Convention on the International Sale of Goods shall not apply to this Agreement.

Should you have any questions concerning this Agreement, you may contact GE by writiing to:
GE, P.O. Box 8106, Charlottesville, VA 22906.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US AND SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT. FURTHER, NO CHANGE OR AMENDMENT TO THIS AGREEMENT SHALL BE EFFECTIVE UNLESS AGREED TO BY WRITTEN INSTRUMENT SIGNED BY A DULY AUTHORIZED REPRESENTATIVE OF GE.

Chapter 1 Getting Started

This section has general foundational information about the ECLiPS software product. There is a general overview of the product, instructions on using this manual, installation procedures, hardware requirements, and sources of information about using ECLiPS.

Overview

ECLiPS stands for “English Control Language Programming System”. ECLiPS is a complete environment (programming tool and on–line debugger) for creating and monitoring State Logic type control programs using natural English terms, phrases and sentences.

The State Logic control programs created by ECLiPS are executed by one of the hardware platforms with the State Engine operating system. The platform for this version of ECLiPS is the State Logic Processor (SLP) installed in a GE Series 90™ PLC system. The SLP is a module that is installed into a Series 90 PLC rack along with the Central Processing Unit (CPU).

The SLP accesses the CPU register and I/O tables through the Series 90 backplane and the CPU controls the I/O through its normal program execution cycle. This is a multiprocessor system, since the CPU may also execute a control program while the SLP is executing a control program.

How to Use this Manual

State Logic differs significantly from traditional approaches to control, therefore, it is very important to read sections 2 and 3**, State Logic Theory** and **Creating an ECLiPS Program**.

After reading these sections, install ECLiPS in the computer and follow along with the Programming Tutorial in section 4. The Programming Tutorial is designed to be completed without connecting to any State Engine controller.

There are also sections on helpful hints and documenting the program, but the reference section is the section most often used after the initial exposure to ECLiPS. The reference section has details about the ECLiPS State Language, using ECLiPS functions, and interfacing to the State Engine.

Notational Conventions:

- All text that should be entered at the keyboard are printed in **bold italics**.
- All references to individual keys are enclosed in angle brackets <>.
- Sample program lines to show examples but not necessarily entered into your computer are displayed in a box.

- Displays showing computer screens are all captures of actual ECLiPS displays with rounded corners on the surrounding box.
- References to menu options appear between double quote marks.
  "Make a New Project"

Brief Description of the Manual Sections

1. **Getting Started**

   Getting Started is the section you are now reading. Getting Started tells you how to install ECLiPS on your DOS based computer and other particulars related to accessing information.

2. **State Logic Control Theory**

   ECLiPS is an interface that allows you to tap into the substantial power and flexibility of State Logic control. 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.**

3. **Creating an ECLiPS Control Program**

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

4. **Programming Tutorial**

   This section walks you through the creation of a simple application programmed in State Logic with ECLiPS.

5. **Helpful Hints**

   This section contains information to make your first programming efforts more efficient with hints on programming and ECLiPS features.

   These are the most important user hints and suggestions that we have learned in feedback from ECLiPS users.

6. **Online Tutorial**

   This section introduces you to the various on–line and process monitoring tools and techniques available to help debug and troubleshoot the State Logic program.

7. **Creating Program Documentation**

   This section explains what types of program documentation can be created.

8. **Reference**

   The Reference Manual is a comprehensive explanation of the ECLiPS commands and procedures. This section contains a glossary of terms.

Hardware Requirements

1. IBM PC compatible or PS2
2. 640K RAM - Extended or expanded memory optional
3. DOS version 3.1 or higher
4. Hard Disk
5. 5.25 inch or 3.5 inch floppy disk drive
6. Serial Port
7. Any printer (Optional)
8. Color or monochrome monitor

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>. Choose the “INSTALL” option. Be ready to specify the hard drive where ECLiPS is to be installed. Follow the instructions for inserting other disks. The installation program displays a message when the installation is complete.

ECLiPS is copy protected so that only one installation is allowed per set of distribution disks. If an attempt is made to run ECLiPS without proper installation, a message is displayed saying that this is an unauthorized version of ECLiPS. The installation program does not allow a second installation from the distribution disks.

If there is a need to move ECLiPS to another computer use the UNINSTALL option of the installation program. This option removes ECLiPS from the computer it is installed on. The distribution disks are modified to allow ECLiPS to be installed again on another computer.

To run ECLiPS, make sure that \ECLIPS\S90–70 is the current directory by typing **CD\ECLIPS\S90–70**and then press <Enter>. Now type **ECLIPS** to start the program.

Getting Help

There are three ways to get help:

1. ECLiPS Help System. ECLiPS has a built in help system that can always be accessed by pressing the key on your keyboard marked <F1>. This help system is context sensitive meaning that ECLiPS provides the helpful information you need based on the location of the cursor on the screen or the highlighted menu option at the moment you ask for help.

   More information about using the ECLiPS help system can be found in the reference section of this manual.

2. ECLiPS Reference Manual. The reference section of this manual contains helpful information organized by the command, function or procedure name. Use the main index at the back of this manual or the reference section index at the beginning of the reference section to locate information.

3. GE has personnel specially trained throughout the country to provide customer support for ECLiPS and other Adatek products which work together with GE control products. Call GE Intelligent Platforms technical support line at 1–800–828–5747.

Register Your Product

Be sure to fill out the registration card in the inside pocket of the front manual cover. By registering your product, you can be assured that you are informed about new upgrades and other State Logic product updates.

Chapter 2 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. It has not been used until recently in industrial control design since few products offered easy access to the strategy. Adatek products provide a means of applying this philosophy easily in all automation applications.

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 often forms a complete loop of activity. 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.

a80001

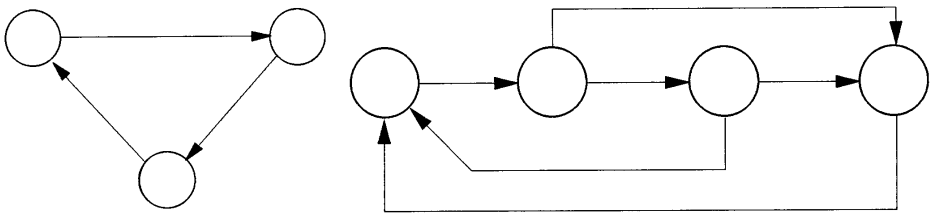


Figure 2-1. State Diagrams

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 until Tank_Gauge equals 35 gallons,
then go to the Add_Chemicals State.

State: Add_Chemicals
When the Chemical_Management Task is in the Mixing State,
turn Pump_2 on. When Tank_Guage 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.

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.

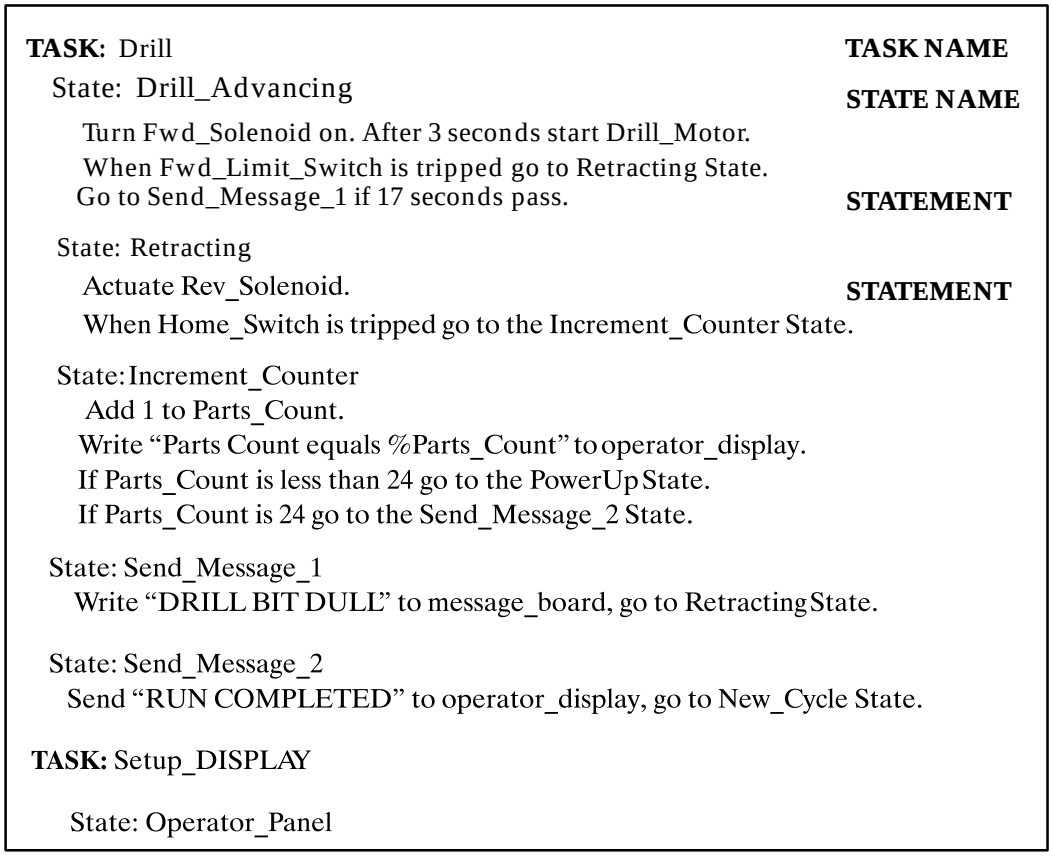
A Collection of Tasks is a State Logic Program

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 Engine 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.



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 until 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.
```

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.
```

Example 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_Motor 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 and Conditional.

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, when
Go	Read, get
Add, Subtract, Divide, etc.	
Make, Set	
Write	
Start_PID, Stop_PID	
Suspend_Task, Resume_Task	
Perform	

Communication Functions

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.

State: Start_Process

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.

Any one of the serial ports may be set up for the CCM2 communications protocol. Using this protocol enables the State Engine controller to be a slave on a CCM2 network. Typically the protocol is used to communicate with Graphical User Interface Software such as CIMPLICITY, Genesis, The FIX, INTOUCH, Factory Link, Screenware II, etc.

Scan Overview

The State Engine which executes the control program continuously scans the inputs and the control program. These scans occur hundreds of times each second. 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 Engine 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 Engine 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 doesn't provide a squirt of gas into the cylinder during the time the Starting System Task is in the Starting State the engine won't 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 be transitioned 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 engine 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 START−UP − 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.

Example Diagnostic Task

Chapter 3 Creating A 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.

If you are having trouble identifying the Tasks in your particular application please don't hesitate to call our customer support line at (800) 323–3343. You may call even if you don't have a specific question but would simply like to discuss the concept of Task architecture.

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 to 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, ie, 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

Task names are arbitrary. 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, the first 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 card. 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.

Statement Structures

The greatest difference between the outline and the ECLiPS program is in the expression of the Statements. 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 can be made to hold their values over a halt or power cycle.

When using ECLiPS to write programs or monitor running controllers, or generate diagnostics, the user needs to only 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.

Time Variables

The time variables are used to view or change the values of the Real Time Clock. 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.

Register Variables

Register Variables are stored in the CPU %R memory locations and may be either floating point, integer, or character data type. Register variables are used when a CPU Ladder Logic Program or some other module in the Series 90 system needs to access the data used in the State Logic program.

Program Scan

The State Engine 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 (.).

a80002

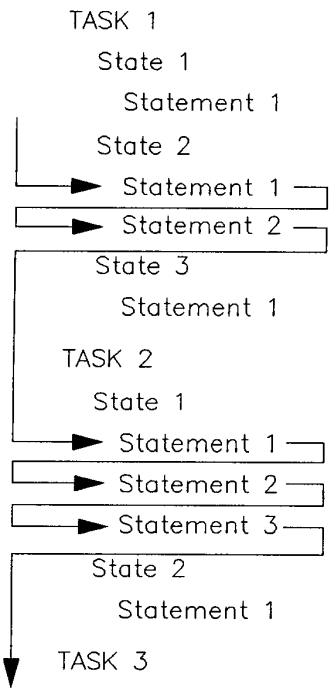


Figure 3-1. Program Scan

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.

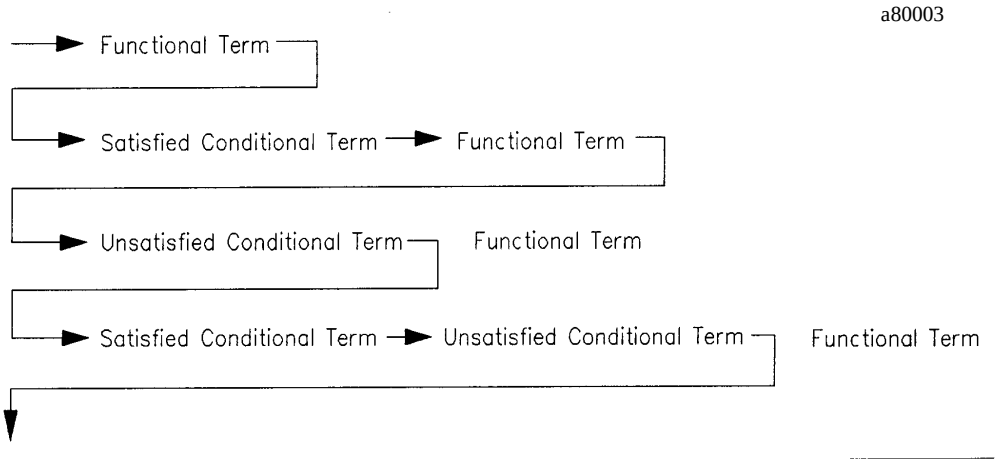


Figure 3-2. Statement Scan

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.

```
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.

Chapter 4 Programming Tutorial

This section presents a step by step procedure for writing a control program for a specific application. Follow along duplicating each keyboard entry on your own computer. Watch for the text displayed in ***bolditalics***, these are the lines that you should enter.

To best understand this tutorial, you should first read the sections on **State Logic Control Theory** and **Creating a Control Program,** sections 2 and 3.

Tutorial Procedure

All that is required to follow along with this tutorial is a computer with ECLiPS installed. All of the examples in this section deal with programming and can be executed without connecting ECLiPS to a State Logic Processor. On the other hand, the Online Tutorial section describing run–time debugging and monitoring functions, does require ECLiPS to be connected to an SLP.

The programming procedure described in the **Creating an ECLiPS Control Program** section, is used in this tutorial. The first step in creating a control program is to write a general outline of the application:

- Identify the Highest Level of the Outline – The Tasks
- Decompose each Task into its Sequence of States
- Write the Statements which Describe the Activity of Each State

The next step is to write the control program from this outline:

- Modify the outline to specify Names of Program Elements
- Define these Names and Configure the I/O Circuits

The Control Application

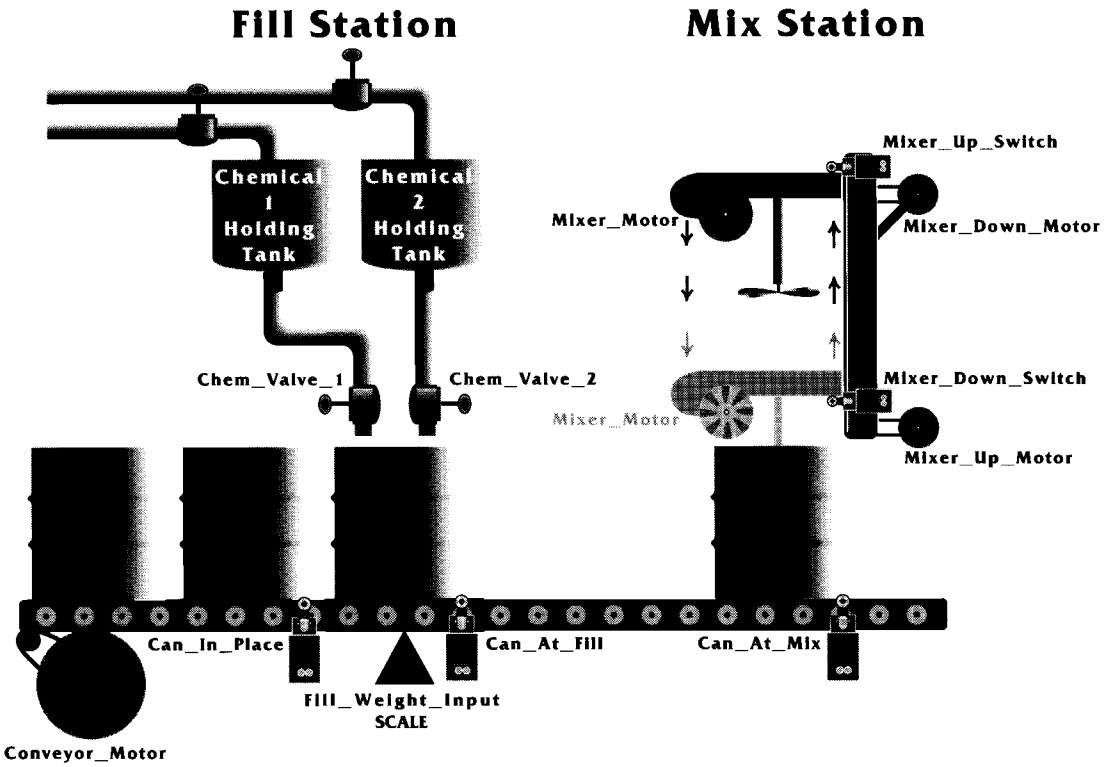


Figure 4-1. Tutorial Application Drawing

The application used for this tutorial is a conveyor belt which moves cans to two stations along the belt. The first station is a filling station which fills the can with two different chemicals, and the second station mixes the chemicals.

When a can arrives at the fill station, 20 pounds of chemical 1 are poured into the can. Then 10 pounds of chemical 2 are poured into the can. When the can is removed from the station, there is a wait until the next can arrives, and then this sequence of operations is repeated.

The mixing station waits for a can to arrive, then lowers a mixer into the can. The mixer is turned ON for 10 seconds and then raised to the up position. The process restarts after the can is removed from the station.

The conveyor waits for a can to be placed at a starting position. Then, if both stations are finished with their operations, the conveyor starts moving. When a can gets to the fill station or the mix station, the conveyor stops. The process now repeats by waiting for another can to be placed in the starting position.

Outline the Application

The first step in creating an ECLiPS program as shown in section 3 is to outline the application using a top down approach. The control problem is first described in general terms. These terms are successively broken down into more specific details. First Tasks are identified, then each Task is defined by a sequence of States, and each State is defined by a collection of Statements.

Identify the Tasks

A Task is a sequence of events or steps. (See the discussions about Tasks in the **State Logic Control Theory** and **Creating a Control Program** sections). Several Tasks are used to describe sequences which execute during the same time period yet are independent from each other. Tasks describe sequential activity which may occur in parallel with other Task activities.

In choosing the Tasks for this problem, notice that the fill station and the mix station operate completely independently from each other. No occurrence at one station has any effect on the operation of the other station. The operation of each station can also be described as a sequence of events. Therefore, the operation of each station is chosen to be a separate Task.

The operation of the conveyor is chosen as the third Task, since its operation can be described as a sequence of events and some of the actions occur simultaneously to the actions of both stations.

I. Fill Station
II. Mix Station
III. Conveyor

Outline with Tasks Only

Since each Task operates concurrently with the other Tasks, the order of Tasks in the outline has no meaning.

Identify the States

Now each Task is decomposed into the States which define the Task. The Tasks are defined in the order that they appear in the outline. (See the sections on **State Logic Control Theory** and **Creating a Control Program** for more information about States).

Fill Station States

Looking at the Fill Station Task sequence of events, we see that there are four distinct steps, each defined by the status of the process at that point in time.

Step 1: On power up the fill station waits for a can to arrive at the station. When a can arrives, the status of the process changes.

Step 2: Chemical 1 starts to pour into the can. Chemical 1 continues to pour into the can until 20 pounds of chemical 1 are in the can.

Step 3: Now chemical 2 starts pouring into the can. The next State of the process starts when 10 pounds of chemical 2 are in the can.

Step 4: The last State is waiting for the can to be removed from the fill station. When the can is removed the procedure repeats by starting again with step 1.

When the process starts, the fill station Task waits for a can to arrive at the station. Normally this State might be called Wait for Can, but since this is the State that is active on start up it is named Power Up. Every Task must have a State named Power Up which is active when the program starts execution.

The other States are given brief descriptive names: Pour Chemical 1, Pour Chemical 2, and Wait for Can Removal. With the naming of the Fill Station States, the outline at this stage is as follows:

I. Fill Station
  A. Power Up
  B. Pour Chemical 1
  C. Pour Chemical 2
  D. Wait for Can Removal
II. Mix Station
III. Conveyor

Outline with Tasks and Fill Station States

Mix Station States

The Mix Station has five distinct steps which describe the sequence of operations at this station.

Step 1: During this step the station is waiting for a can to arrive at the station. After a can arrives the next step starts.

Step 2: Now the mixer is lowered into the can. When the mixer is down, the process goes to the next step.

Step 3: The action of this step is to mix the contents of the can. The process goes to the next step after the contents have been mixed for 10 seconds.

Step 4: During this step the mixer is raised. After the mixer is all the way up the last step begins.

Step 5: The final State waits for the can to be removed. When a can is removed the procedure repeats by going to step 1 again.

I. Fill Station
- A. Power Up
- B. Pour Chemical 1
- C. Pour Chemical 2
- D. Wait for Can Removal

II. Mix Station
- A. Power Up
- B. Lower Mixer
- C. Mix Chemicals
- D. Raise Mixer
- E. Wait for Can Removal

III. Conveyor

Outline Showing Fill Station and Mix Station States

Conveyor States

The conveyor operation is described by a series of five steps.

Step 1: During the starting step the Conveyor waits for a can to be placed on the conveyor. When a can is in place the next step starts.

Step 2: This step is a check on the Fill Station to see if it is finished with its operations. When the Fill Station is either waiting for a can to be removed (Step 4) or waiting for a can to arrive (Step 1) the next step is started.

Step 3: Similar to the previous step, this step checks the Mix Station to see if it has completed its operations. If the Mix Station is either waiting for a can to be removed or a can to arrive the conveyor may be started and the next step is started.

Step 4: This step starts the conveyor moving and continues until the can placed in the starting position has moved enough for another can to be placed on the conveyor.

Step 5: During this step the conveyer continues to run until a can reaches either one of the two stations, then the procedure restarts at the first step.

I. Fill Station
  A. Power Up
  B. Pour Chemical 1
  C. Pour Chemical 2
  D. Wait for Can Removal
II. Mix Station
  A. Power Up
  B. Lower Mixer
  C. Mix Chemicals
  D. Raise Mixer
  E. Wait for Can Removal
III. Conveyor
  A. Power Up
  B. Fill Station Wait
  C. Mix Station Wait
  D. Start Conveyor
  E. Moving Cans

Outline with All Tasks and States

Identify the Statements

This section fills in the lowest and most specific level of the outline. The actions and transitions of each State are identified and placed into the outline. The Statements are entered for each Task in turn. The Statements within the text appear in **bold type.**

Fill Station Statements

There is one transition and no action in the Power Up State (see sections 2 and 3 for a discussion of the terms used here). The condition of the Statement is whether or not a can is at the station and the target State is the Pour Chemical 1 State. The Statement for the Power Up State can be written: **If a can is at the Fill Station, go to the Pour Chemical 1 State.**

For the Pour Chemical 1 State, the action is pouring the chemical. The Statement for this action can be written **Pour Chemical 1**. The transition is to go to Pour Chemical 2 State when 20 pounds of chemical 1 are in the can. A Statement for the transition is: **When the weight of the can contents is over 20 pounds, go to Pour Chemical 2 State.**

Pour Chemical 2 State is almost identical to Pour Chemical 1 State. **Pour Chemical 2. If the weight of the can contents is over 30 pounds, then go to Wait for Can Removal State.**

The last State, Wait for Can Removal, has no action and one transition. The Statement for this State is: **When there is no can at the station, go to the Power Up State.**

I. Fill Station
  A. Power Up
    1. If a can is at the Fill Station, go to the Pour Chemical 1 State.
  B. Pour Chemical 1
    1. Pour Chemical 1.
    2. When the weight of the can contents is over 20 pounds, go to Pour Chemical 2 State.
  C. Pour Chemical 2
    1. Pour chemical 2.
    2. If the weight of the contents of the can is over 30 pounds, then go to Wait for Can Removal State.
  D. Wait for Can Removal
    1. When there is no can at the station, go to the Power Up State.

Fill Station Task Outline

Mix Station Statements

The Statements for the Mix Station are very similar to those for the Fill Station. The completed outline for the Mix Station follows.

II. Mix Station
  A. Power Up
    1. If a can is at the Mix Station, go to the Lower Mixer State.
  B. Lower Mixer
    1. Lower the mixer.
    2. When the Mixer is in the down position, go to Mix Chemicals State.
  C. Mix Chemicals
    1. Mix the chemicals for 10 seconds, then go to the Raise Mixer State.
  D. Raise Mixer
    1. Raise the mixer.
    2. When the Mixer is in the up position, then go to the Wait for Can Removal State.
  E. Wait for Can Removal
    1. When there is no can at the mix station, go to the Power Up State.

Mix Station Task Outline

Conveyor Statements

III. Conveyor
A. Power Up
1. If a can is at the start position, go to the Fill Station Wait State.
B. Fill Station Wait
1. When the Fill Station is in the Power Up State or the Fill Station is in the Wait for Can Removal State, go to the Mix Station Wait State.
C. Mix Station Wait
1. When the Mix Station is in the Power Up State or the Mix Station is in the Wait for Can Removal State, go to the Start Conveyor State.
D. Start Conveyor
1. Run Conveyor.
2. When a can is not in the start position, go to the Moving Cans State.
E. Moving Cans
1. Run Conveyor.
2. When a can is at the Fill Station, go to the Power Up State
3. When a can is at the Mix Station, go to the Power Up State.

Conveyor Task Outline

One item to note in these Statements is that the Start Conveyor State and Moving Cans State both specify that the conveyor be running. These Statements are consistent with the idea that the actions or outputs for each State are specified in that State. If an output is ON in one State it is not automatically ON in the next State, but must be explicitly stated to be ON in the next State.

Writing the Program

This section demonstrates writing the ECLiPS control program from an outline of the control application. First the I/O for the application are defined on paper, then the actual entry of the control process is described keystroke by keystroke.

Identify the I/O

The outline has been written in general terms describing the activities of the application. To write the actual program real world outputs and inputs must be specified. The actual assignment of names to input and output circuits may be done before, during, or after the program is written. In this section the actual I/O devices and their program names are specified.

The table in this section identifies the actual I/O devices for this application and the names to be used in the program. See the **Effective Use of English Names** part of the **Creating a Control Program** section of this manual.

Clever, descriptive names that fit well to the primary attribute of that circuit's function is strongly encouraged. This practice pays 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. Names can contain letters, numbers and the underscore character (_) but must begin with a letter. The name must be a continuous string of characters.

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.

Right_Forward_Clamp
RightForwardClamp

Digital Inputs	**Names**
Switch detecting can at start of conveyor	Can_In_Place
Switch detecting can at fill station	Can_At_Fill
Switch detecting can at mix station	Can_At_Mix
Switch indicating mixer in up position	Mixer_Up_Switch
Switch indicating mixer in down position.	Mixer_Down_Switch
Digital Outputs	
Valve for chemical 1	Chem_Valve_1
Valve for chemical 2	Chem_Valve_2
Mixer motor	Mixer_Motor
Mixer up motor	Mixer_Up_Motor
Mixer down motor	Mixer_Down_Motor
Conveyor motor	Conveyor_Motor
Analog Input	
Weight scale at the fill station	Fill_Weight_Input

Conveyor Project I/O Names

The Basics of ECLiPS

This section discusses the fundamentals of operating ECLiPS, bringing us to the point that we may begin programming. After you have followed the installation instructions from section 1, enter **ECLIPS** at your keyboard then press <Enter> to start the ECLiPS program.

The display on the monitor has a bar on top which displays the name of the current project and a two line instruction bar at the bottom. Look to the top and bottom bars for information on the actions of certain keys or other helpful information throughout your work with ECLiPS.

ECLiPS is a menu driven program which means that most of the available operations are chosen from a series of menus. The main menu is displayed when ECLiPS is first started.

The bar at the bottom describes how to use the menu. Use the up and down arrow keys to move the highlight bar to the desired choice then press <Enter>. Selections can also be made by pressing the highlighted letter of the desired choice. Some options also display a hot key, which shows the keys to press to activate the menu option without going through the menu. The first screen shows the main menu.

a80005

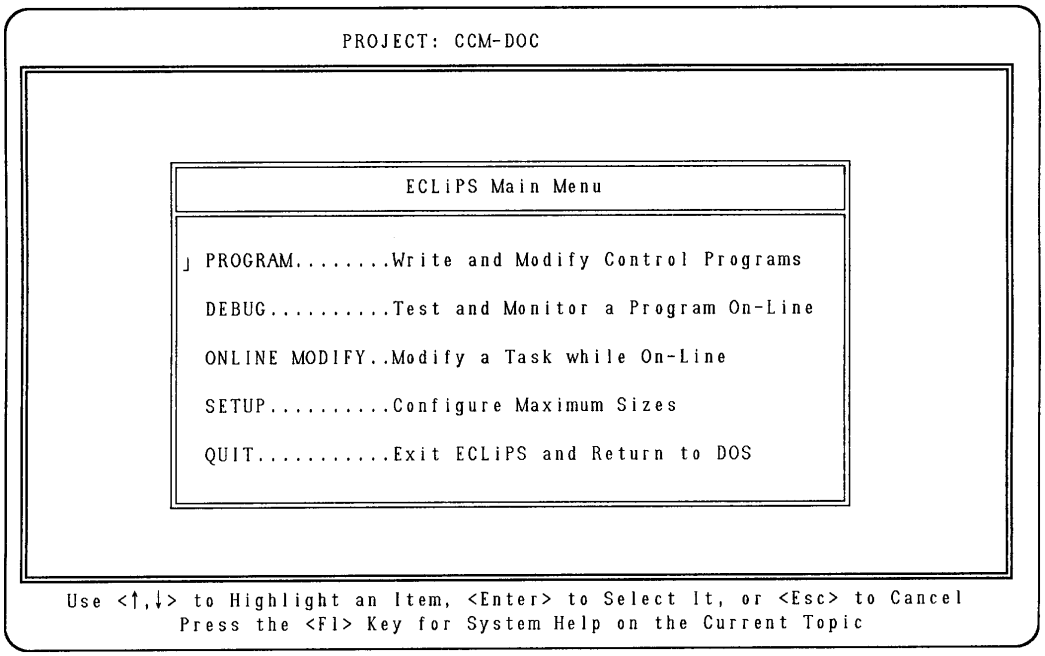


Figure 4-2. Main Menu

The instruction bar also indicates that pressing <Esc> cancels the operation and that pressing <F1> enters the HELP system. The function of these keys is consistent throughout the program. Selections from all menus are made the same way.

Pressing <F1> always provides help information on the current operation, therefore it is referred to as a context sensitive help system. While the help screen is displayed, pressing <F1> again displays functions which may be executed by pressing other function keys. Some help screens display instructions on how to view additional help screens. To return from the help screens press <Esc>.

Choose the PROGRAM menu by using the up / down arrow keys to highlight your choice and then pressing the enter key. Now press <F3> to view the PROGRAM menu. You should spend some time exploring the menu options, since these options represent most of the functions available in ECLiPS. The reference section has details on the use of these options. Also use the context sensitive help which explains the highlighted option of each menu when you press <F1>. Select the PROJECT choice by moving the highlighted bar with the arrow keys or merely pressing <P>.

a80006

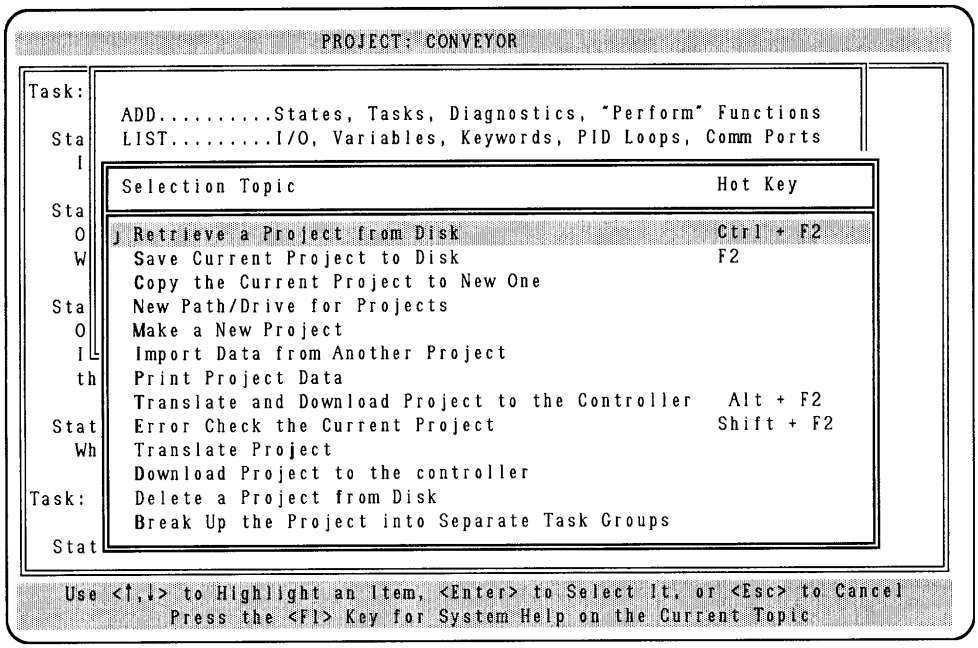


Figure 4-3. Project Menu

The project menu is now displayed with several choices relating to specific projects. Notice that the project menu is overlaid on top of the program menu. There is a check by project on this menu showing the choice which caused the current menu to be displayed. The older menu becomes the current menu if the escape key is pressed.

Choose the “Make a New Project” choice from the menu. A box appears indicating that the operation is making a new project and that the name of the project must be entered. ECLiPS will create some new files with this name and various extensions, so the name must conform to MS–DOS file name requirements, ie. eight characters, first character a letter, letters, numbers, and the _ and $ characters allowed. Enter the name of the project; **CONVEYOR**.

The next two options define the type of project that is create. The first option selects the type of CPU (731 / 732, 771 / 772, 781 / 782) in the Series 90–70 control system. Select the CPU that you are using. This option can be changed at any time.

The next option selects the method of information transfer between the SLP and the CPU. The Co–processor option selects the SLP to do the transfer, while the Controller options (large and small) selects the CPU to do the transfer. The Controller options are for high speed applications and the Co–processor option provides more flexibility. The Controller options use a ladder program in the CPU to accomplish the transfer. The small version is for applications where the CPU is not doing any control, but the large version can be used when the CPU has a Ladder Logic program that is also doing some control operations.

Select the Co–processor option for this exercise.

Program the Fill Station Task

In this section we program the Fill Station Task with ECLiPS. A box now appears for the name of the first Task. Enter the name of the first Task now: **Fill_Station**. Remember that names in ECLiPS have no spaces.

ECLiPS first asks whether to include the Watchdog Timer Task and whether to restart this Task in the State that was active when the program was halted. Select NO for both of these options.

ECLiPS now asks for a diagnostic value for the maximum length that the PowerUp State should be active. This is a diagnostic option which we will discuss later. For now just hit the <Enter> key which tells ECLiPS not to include this option.

ECLiPS has named the first State of this Task PowerUp. Every Task must have a Power-Up State which is the State that is active when the program starts.

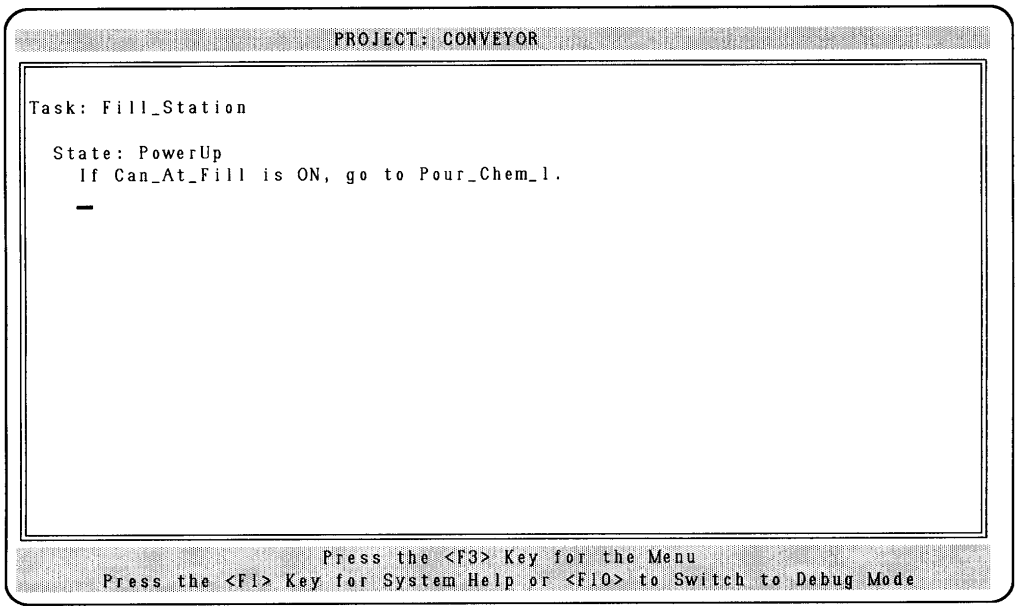
ECLiPS now displays the first Task's name and its first State PowerUp. Next enter the first Statement at your keyboard:

If Can_At_Fill is on, go to the Pour_Chem_1State.

This is a complete description of the first State. This Statement varies slightly from the Statement in the outline to accommodate direct references to I/O devices and the naming conventions.

This is how your screen should now appear:

a80007



To enter the next State hit <F3> to view the menu, select ADD, then "Add a New State". Enter the State name: **Pour_Chem_1**. You are asked to specify the maximum time for the State to be active each time a State is created. From now on until we discuss this option just hit <ENTER> to ignore this option. Now describe this State by entering the following Statements:

OpenChem_Valve_1.
When Fill_Weight_Input is above 20 pounds, go to Pour_Chem_State_2.

The first Statement turns on the output which controls the valve for the first chemical. The second Statement defines the only transition of the State which will happen after 20 pounds of chemical 1 have entered the can.

It is not necessary to use the ADD functions to enter a new State. To enter the next State, make some space at the end of the program by pressing the <Enter> key twice. Now just enter at the keyboard the following:

State: Pour_Chem_2

Enter the description of this State as follows:

Open Chem_Valve_2.
If the Fill_Weight_Input is greater than 30 pounds,
then go to Wait_for_Can_Removal.

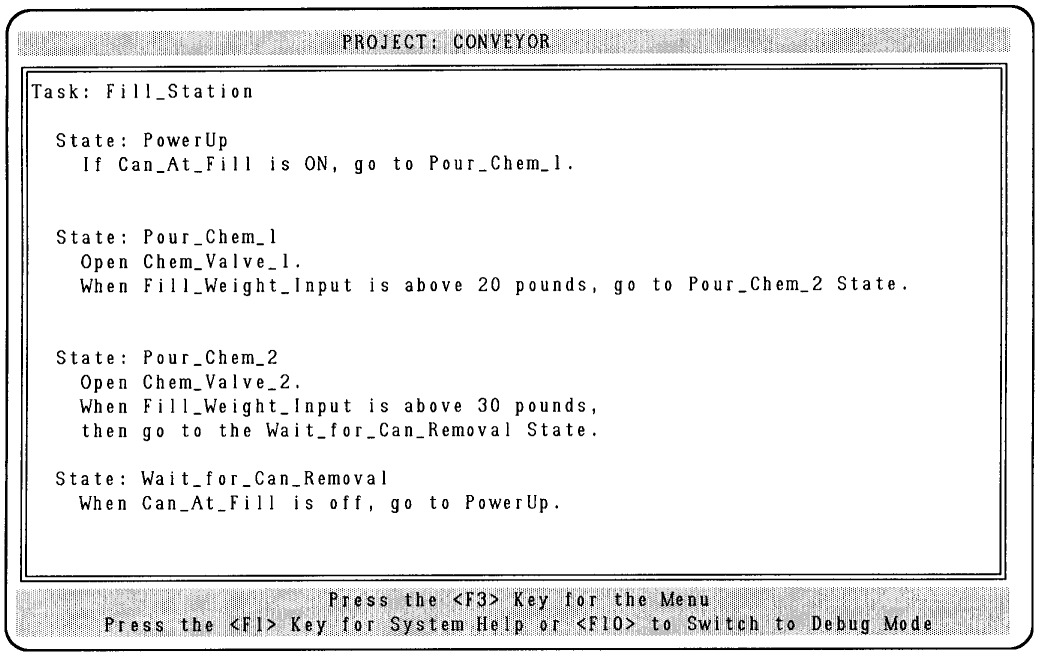
ECLiPS provides some of the flexibility of expression in that functions can be specified in more than one choice of expressions. These last two States are very similar in function but have been specified in different ways. See the Basic Programming Structure section in the **Reference** section of this manual.

Enter the next State called Wait_for_Can_Removal as follows:

State: Wait_for_Can_Removal
When Can_At_fill is OFF, go to PowerUp.

Notice that ECLiPS is not case sensitive, ie. it does not recognize differences between upper and lower case letters. The following is how your screen should appear:

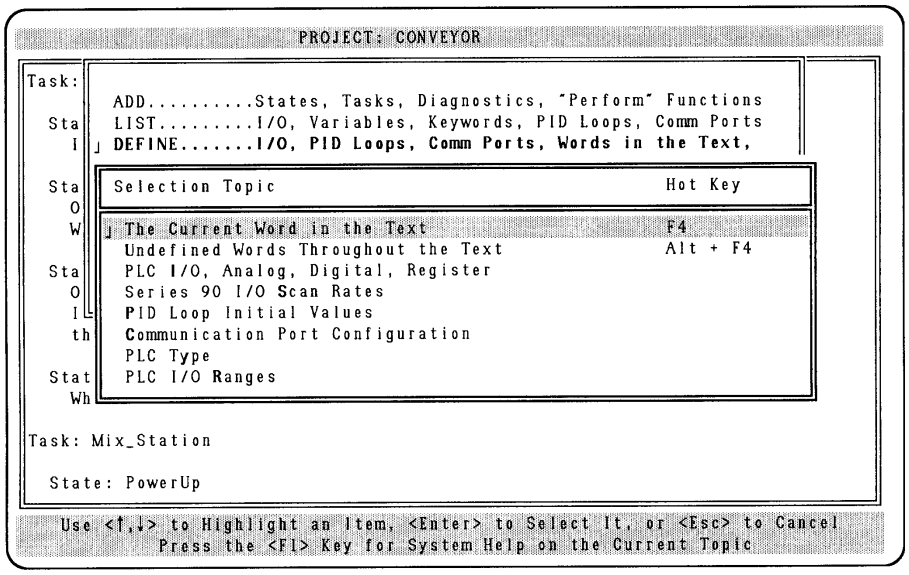
a80008



Define Undefined Words

All names and undefined words used in the program must be defined for the control program to execute. Call up the menu by hitting <F3>, then select the DEFINE choice. The choices on the DEFINE menu offer several different ways to define the critical words of the program.

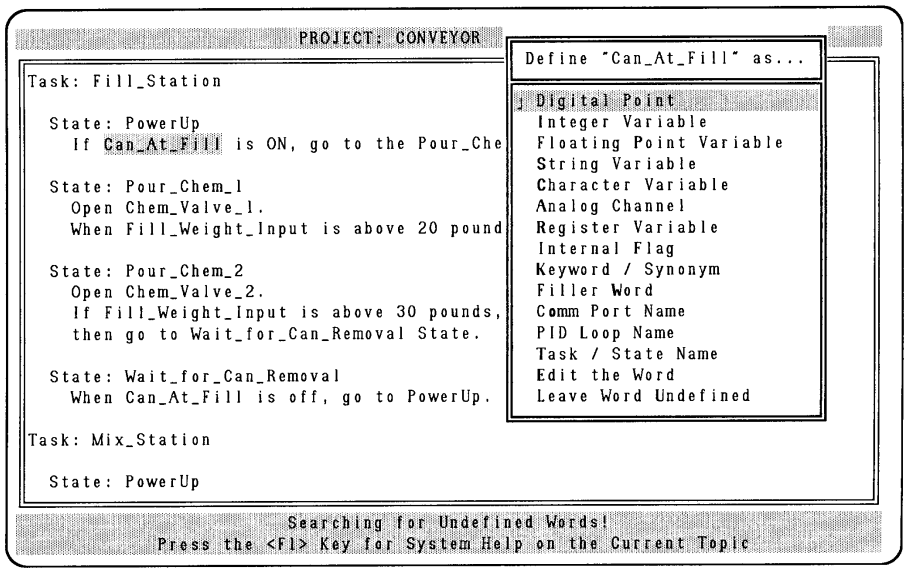
a80009



The "The Current Word in Text" option defines the word where the cursor is now located (hot key <F4>). The "Digital Point Names and Addresses" and "Analog Channel Configuration and Scaling" choices define only I/O names. The "Undefined Words Throughout the Text" (hot key <ALT + F4>) choice finds all of the undefined words in the program, displaying them to be defined one at a time.

Lets start by selecting the "Undefined Words Throughout the Text" option. The first undefined word found is Can_At_Fill.

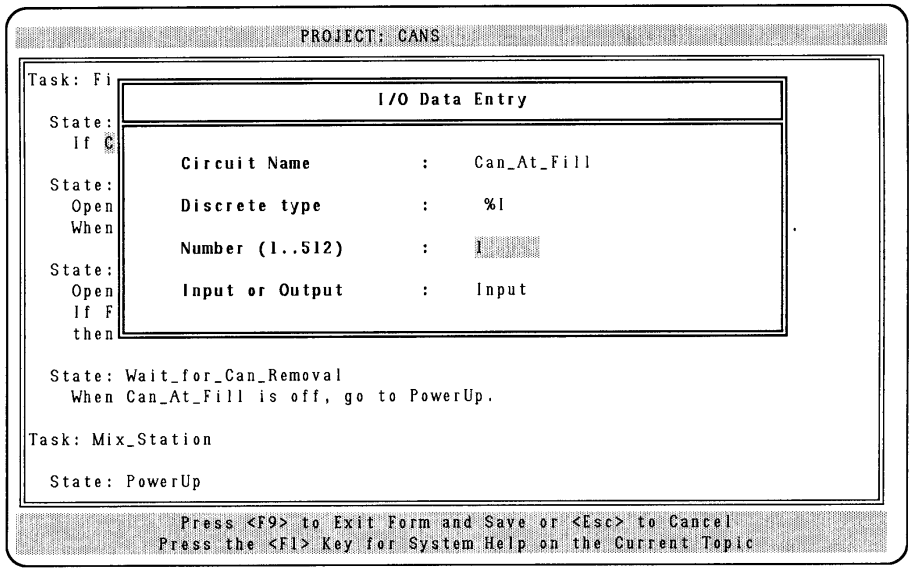
a80010



A menu now appears with choices to identify the type of word. Can_At_Fill we know is a limit switch and therefore we choose “Digital Point” from the menu.

Next a form is displayed to identify how this digital point is stored and accessed in the CPU of the Series 90 control system.

a80011



Fill in the form as shown in the I/O Data Screen above. The name is displayed for the first option. The second option is the discrete type. Press any key and a list of all of the types is displayed. Select %I for the Can_At_Fill input by highlighting the %I option and pressing <Enter>. Now enter 1 for the number of this %I discrete point. Next select Input for the Input/Output option. Inputs can be tested, and outputs can be either tested or changed.

The mapping of the discrete types to physical hardware I/O modules is accomplished by the configuration of the CPU with the Logicmaster 90 software package. See the section on CPU Configuration in the Debug Mode Tutorial Section of this manual or you can find CPU configuration information in the Reference section.

The next undefined word, Chem_Valve_1, is now displayed. Again define Chem_Valve_1 as a digital point, but make it a %Q discrete type, number 1, and Output. With respect to the CPU, %I’s are inputs and %Q’s outputs.

After completing the definition of Chem_Valve_1 digital point, ECLiPS finds Fill_Weight_Input as the next word to define. This word is the name of an analog input so choose the “Analog Channel” option from the define menu. The analog options are similar, the only change being the analog types. Choose %AI for this analog input, number 1, and Input.

After these options are chosen, ECLiPS asks if you want to enter scaling values for this analog channel. Scaling an analog channel means that the raw data used by the I/O module is converted into a different range of values for the State Logic Program. The converted value is a floating point value while the raw values are integers.

Select the YES option to scale the Fill_Weight_Input analog channel. A form is displayed to specify how raw analog values (D/A or A/D) are converted to engineering units. The range of raw values is selected during the configuration of the CPU using Logicmaster 90.

Assuming that the analog channel configuration is 0 – 4095, fill in the A / D or D / A values in the form as 0 for the Low Point and 4095 for the high point. The Fill_Weight_Input represents a scale which measures from 0 to 50 pounds. Enter 0 for the Low Point engineering units value and 50 for the High value. All references to this analog input in the program, return a floating point value from 0 to 50.

Chem_Valve_2 is the next undefined word, which must be defined as a digital output. The next word to define is pounds. The definition of this word demonstrates how engineering units are handled in ECLiPS. The analog circuit configuration is set up so that the value returned represents the units desired. The units specified in the ECLiPS program serves documentation purposes, but has no effect on the operation of the control program. Define this word as a filler word by choosing the Filler Word option from the define menu.

This concludes the programming of a complete State Logic Task. Notice that this Task by itself could be a completely independent program, if the Fill Station were the only part of the application to control. Take some time to explore the menus which offer some of the ECLiPS convenience features. The next section describes how to check the program and make changes.

Checking Your Work and Making Changes

At this point all words have been defined, but there may be errors in the program and most often changes need to be made throughout the life of a program. This sections discusses checking the program for errors and how to make program changes.

To check the project for errors choose PROJECT from the menu, and then the "Error Check the Current Project" option from the next menu (hot key <Shift + F2>). ECLiPS may ask if you want to check for undefined words. Answer NO to this option. If ECLiPS finds no errors, a form is displayed showing the statistics on the current project. If there is an error, an error message is displayed in the bottom bar of the screen. To demonstrate how errors are displayed, delete the period following the last Statement of the program. Now check the project for errors. When the error message is displayed, press the <CTRL + F1> to view the help message for this error.

To view definitions of the words for the project or to view other system attributes choose the LIST option from the menu. A list of topics to view appears. Choose from the list of topics to view the specific word definitions. The bar at the bottom of the screen shows several functions available to manipulate or use these definitions. The functions available are inserting a new definition, deleting a definition, and editing the definition. In addition the word defined may be entered into the program at the current cursor location. The LIST option is very useful with large applications, since it is difficult to remember all of the words that have been defined.

To make changes in the text of the program, ECLiPS provides the basic functions of a word processor. To move through the text use the arrow keys, <HOME> and <END> keys, and <PAGE UP> and <PAGE DOWN> keys. Use the backspace and delete keys to erase text, the enter key to add new lines, <ALT + F1> to toggle between insert mode and overtype mode, and the tab key to indent. To perform more advanced text editing operations, use the FIND and TEXT options from the menu. Press the <F1> key twice displays a keyboard summary containing several pages of the functions available from different key combinations.

To make changes to I / O names and specifications choose the LIST option from the menu, then select DIGITAL or ANALOG. Next highlight the I / O name from the list and press the right arrow key "–> ". At this point the name and all of the specifications for the I / O point may be changed.

Program the Mix Station Task

The tutorial continues by programming the Mixing Station Task of the example. On power up, the mix station is waiting for a can to arrive. The limit switch, Can_At_Mix, senses when a can is in place. After this switch is on, the mixer is lowered by turning the Mixer_Down_Motor digital output on. When the mixer is in mixing position, which is sensed by the Mixer_Down_Switch, the Mixer_Motor is turned on for 10 seconds. The mixer is then raised by turning Mixer_Up_Motor on. When the Mixer_Up_Switch is on, the mixer stops rising. The mixer waits for the can to be removed and another one to be set in place to repeat the operation.

Start programming this Task by first choosing the ADD option from the menu. Enter **Mix_Station** for the Task name. Add the rest of the Task as it appears in the following screen:

a80012

```
                    PROJECT: CONVEYOR

Task: Mix_Station

    State: PowerUp
        If Can_At_Mix is tripped, go to the Lower_Mixer State.

    State: Lower_Mixer  Max_Time 20
        Turn on the Mixer_Down_Motor
        until the Mixer_Down_Switch is tripped,
        then go to Mix_Chemicals.

    State: Mix_Chemicals
        Run the Mixer_Motor for 10 seconds, then go to Raise_Mixer State.

    State: Raise_Mixer  Max_Time 30
        Turn on the Mixer_Up_Motor
        until the Mixer_Up_Switch is tripped,
        then go to Wait_for_Can_Removal State.

    State: Wait_for_Can_Removal
        When Can_At_Mix is off, go to PowerUp.

                    Press the <F3> Key for the Menu
  Press the <F1> Key for System Help or <F10> to Switch to Debug Mode
```

Define all of the undefined words. The only required operation not discussed during the programming of the Fill Station is how to define a synonym for a keyword. Tripped must be defined as a synonym for ON. Choose “Keyword / Synonym” option from the LIST menu. A list of keyword and synonyms appears. Select ON in the list and press <Enter>, so that tripped is now a synonym for ON.

Lets try another new option. Use the maximum time diagnostic option for States Lower_Mixer and Raise_Mixer. These diagnostics are added so that if either the Mixer_Up_Switch or Mixer_Down_Switch fails, a message saying that either the Lower_Mixer or Raise_Mixer State has been active too long is displayed.

This Task introduces the use of the WAIT term Statement as shown in the Mix_Chemicals State. All timing functions start when the State they are in becomes active.

Conveyor

On power up the conveyor waits for a can to be placed in the start position, which is sensed by digital input, Can_In_Place. Now both the Fill_Station Task and the Mix_Station Task are checked to make sure they are not in the middle of their operations so that the conveyor can be started. Both of these Tasks must be in either the PowerUp or the Wait_for_Can_removal State before the Conveyor Task goes to the Start_Conveyor State. The conveyor runs until the starting can has moved off the Can_In_Place switch, then the conveyor runs until a can either reaches the fill station or the mix station. The conveyor now stops and waits until a new can is placed in the starting position.

Now you should be able to enter the Conveyor Task on your own:

a80018

```
                    PROJECT: CONVEYOR

Task: Conveyor

    State: PowerUp
        If Can_In_Place is ON, go to Fill_Station_Wait State.
    State: Fill_Station_Wait
        When the Fill_Station Task is in the PowerUp State or
        the Fill_Station Task is in the Wait_for_Can_Removal State,
        go to Mix_Station_Wait State.
    State: Mix_Station_Wait
        When the Mix_Station Task is in the PowerUp State or
        the Mix_Station Task is in the Wait_for_Can_Removal State,
        go to the Start_Conveyor State.
    State: Start_Conveyor
        Start Conveyor_Motor.
        When Can_In_Place is OFF, go to Moving_Cans State.
    State: Moving_Cans
        Run Conveyor_Motor.
        When Can_At_Fill or Can_At_Mix is ON, go to the PowerUp State.


                    Press the <F3> Key for the Menu
     Press the <F1> Key for System Help or <F10> to Switch to Debug Mode
```

Figure 4-4. Conveyor Task

There are several important points illustrated by the program lines for this Task. Notice the States Fill_Station_Wait and Mix_Station_Wait, how to test for multiple conditions using the OR logical operator. Also notice that in both the Start_Conveyor and Moving_Cans States that the conveyor motor is on. Start and Run are synonyms, but more importantly if an output is to be on in a State, it must be explicitly turned on, even if it were on in the previous State. The inputs Can_At_Fill and Can_At_Mix are used in the other Tasks and can also be used in this Task.

The State Engine scans the active States of each Task one Statement at a time. If a command to transition to another State is scanned, the remaining Statements are not scanned. The final two Statements of Moving_Cans State demonstrate these scanning characteristics, since if the first conditional is true, the next Statement is not scanned.

Check the program again for errors. Correct any problems found. Now save the program by choosing PROJECT from the menu and then the “Save Current Project to Disk” option.

Congratulations! You have just programmed an application using State Logic.

Advanced Programming Terms and Diagnostics

The program developed so far demonstrates the basics of the State Logic Control theory. This program is now changed to demonstrate how to send messages to a terminal, read information from a keyboard, add some diagnostics, and program a counter.

Read and Write Terms

Assume that after the program has been operating for awhile, a decision is made to use two different types of cans, types A and B. The operator enters the type of can after it is placed onto the conveyor. A message is sent to the screen prompting the operator to enter the type of can placed on the conveyor.

```
State: PowerUp
        If the Can_In_Place is on,
        write "Enter type of can (A or B), then <ENTER>." and
        go to the Operator_Input State.

State: Operator_Input
        Read Can_Type, then go to the Fill_Station_Wait State.
```

Enter the program changes as shown above. First the PowerUp State of the Conveyor Task is changed so that a message is sent to the screen after the Can_In_Place switch is tripped. A new State is added which reads the operator's entry at the keyboard. This State is called Operator_Input and becomes active after the Can_In_Place switch is ON. As soon as this input is entered at the keyboard, the State transitions to the Ready_To_Move State and the conveyor Task continues execution as before.

When adding States to a program, make sure the cursor is at the position where the new State is to appear before selecting the ADD option from the menu or merely type in the program changes as they appear above.

Define Can_Type to be a string variable. The write term is a functional term which must appear after the conditional term in the Statement. The write term sends everything that appears inside the double quote marks to the terminal, therefore the period is sent to the terminal as part of the message transmitted and does not indicate the end of the Statement.

The read term is a conditional term which is not satisfied or true until the <Enter> key is pressed signaling the end of the operator input. Therefore the read term must be used with some care so that a time critical Task is not delayed waiting for some input from the keyboard. As soon as the <Enter> key is pressed, the conditional is satisfied and Ready_To_Move becomes the new active State.

Counter

Assume also that the decision to change the application includes keeping track of the number of each type of can that has been placed on the conveyor. A counter is created for each type of can. ECLiPS provides a complete range of mathematical functions including trigonometric and transcendental functions.

A new State is created which becomes active after the operator completes his input. This State adds 1 to one of two variables holding the number of cans of the particular type. After the counters are set, we write the number of each can type to the screen.

Change States in the Conveyor Task as follows:

```
State: Operator_Input
        Read Can_Type, then go to the Counter State.

State: Counter
        If Can_Type = "A", make Number_A_Cans = Number_A_Cans + 1.
        If Can_Type = "B", Add 1 to Number_B_Cans.
        Write "The number of type A cans is %Number_A_Cans.".
        Write "The number of type B cans is %Number_B_Cans.".
        Go to Fill_Station_Wait.
```

Define the variables, Number_A_Cans and Number_B_Cans, to be integer variables. This change demonstrates the concept of variables. Variables hold some data which may be manipulated, compared, and written to the screen.

The string variable Can_Type is compared to two different given strings. Strings are enclosed in double quote marks, so in our write Statements we are writing out strings of characters.

We manipulate the two integer variables by incrementing them by 1, if the preceding conditional is true. These variables hold the value of the counter. Notice there is a short way to perform simple addition: "Add 1 to Number_B_Cans."

These program lines demonstrate the use of variables in write Statements. The % indicates that the value of the following variable is what is to be printed, not the variable name. See the reference section or help system for other formatting functions available for the write term.

Diagnostic

Programmed diagnostics provide alternate paths of program execution when machine faults occur. Often the alternate path produces an alarm condition which requires an operator response, stops the process, or merely records the event.

Perhaps in the process there is a problem with the valves which control the flow of the chemicals flowing into the can. These valves may not close completely causing the chemicals to flow onto the conveyor when the can is moved toward the mixing station.

To deal with this problem a diagnostic is added to the Wait_for_Can_Removal State of the Fill_Station Task. In this State the program is waiting for the can to be moved. If the weight of the can goes over 31 pounds in this State then we know one of the valves is stuck in the open position.

When the problem is detected the process transitions to an alarm State where a message is sent to the operator's screen and the Task halts until the operator resets the alarm condition.

To enter the diagnostic follow this procedure. Place the cursor at the end of the Wait_for_Can_Removal State. Now, choose the ADD option from the program menu and then the "Add an Analog Range Checking Diagnostic" choice from the next menu. A list displays all of the defined analog channels, only one in our case. Choose this channel, which is now entered into the diagnostic form displayed on the screen. Enter 31 as the high limit value, press <Enter> and then enter **alarm** in the first "Branch to:" blank. Alarm is the name of the State created to respond to our alarm condition. When complete press <F9> and answer yes that you want to save the entered data. A program line is entered into the text which performs this diagnostic.

Add the following States to the program:

```
State: Alarm
        Write "Chemical valves not closed.".
        Write "Hit any key then <ENTER> to reset this alarm".
        Go to Reset_Alarm State.

State: Reset_Alarm
        Read Alarm_Reset, then go to Wait_for_Can_Removal State.
```

Now when the faulty conditions occurs, the program displays the alarm message and how to reset the alarm so the process can continue. After the alarm condition is reset the process returns to the Wait_for_Can_Removal State. Assume that the problem has been corrected and the can has been removed or some of the chemicals removed so that the process does not reenter the alarm State. Note that the conveyor does not move until the Fill_Station Task is in the PowerUp State. Define Alarm_Reset to be a string variable.

This alarm diagnostic is the first instance where the States of a Task are not strictly sequential. States can be written in any order in the Task and GO Terms may transition to any other State in a Task.

Now that you have programmed your first applications using State Logic, you have a fundamental knowledge of ECLiPS which is sufficient for many applications.

Chapter 5 Helpful Hints

The hints and suggestions found in this section have been developed by ECLiPS and State Language programmers in an effort to make the user's first programming experience as productive as possible. There are two parts to this section, one giving hints for programming and the other has suggestions for using the tools provided as part of ECLiPS.

Programming Hints

Outputs are OFF by Default

One of the key featuresa 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

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.

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.

Calculations and a Scanning Operating System

Because the State Engine 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 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 the PowerUp State.
```

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.

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 %R CPU registers 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.

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.

- PowerUp
- CanAtFill
- FillWeightInput
- 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.

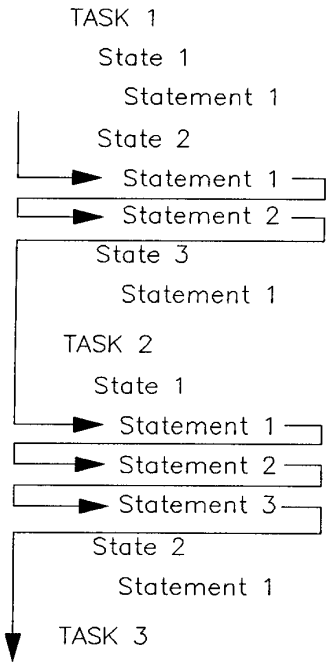
Comments

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.

Program Scan

The State Engine 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 (.).



a80002

Figure 5-1. Program Scan

The actions specified by Functional Terms are executed when the 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.

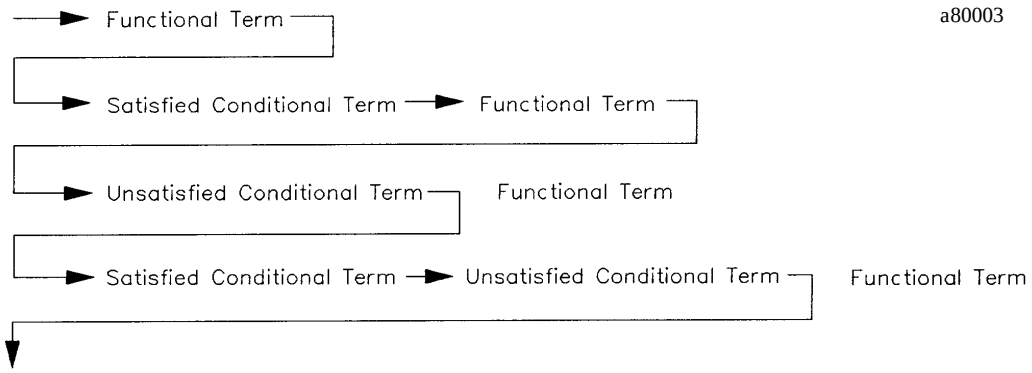


Figure 5-2. Statement Scan

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, none of the Functional Terms are executed.

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.

For example:

If Start_Pushbutton is pushed, go to Start_Up State.
GO to Restart_State.

causes the Start_Up State to become the active State when the process input represented by Start_Pushbutton is true. All Terms or Statements following this Statement are not executed and at the next controller cycle, the scan of this Task starts at the first Statement of the Start_Up State.

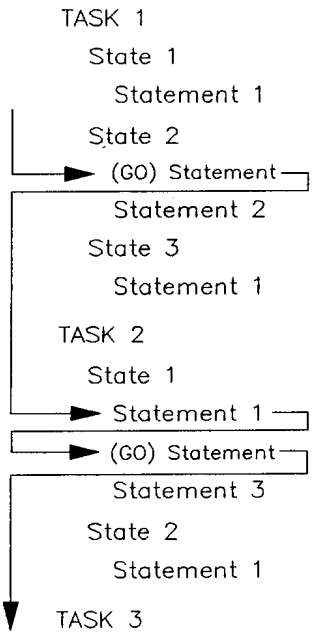


Figure 5-3. Program Scan with GO Terms

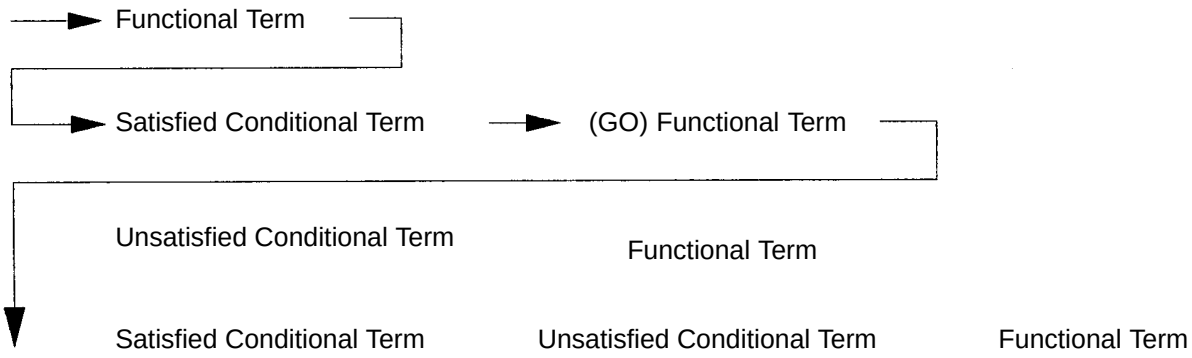


Figure 5-4. Statement Scan with GO Terms

During the program scan any changes to variable and analog values 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 conditions 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 Using ECLiPS Features

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).

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

The **Reference** section of this manual has a table of Hot Keys in the **Keyboard Definitions** section. 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 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 for Define Menu options; the <F5> key is used for the Find Menu options, <F6> for Add Menu options, and <F8> for the Text options.

ECLiPS Word Processing Functions

A program developed using ECLiPS is simply a document of ASCII text. The ECLiPS editor is fundamentally a word processor with specialized functions for editing a State Logic program.

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 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 <P age 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 6 Online Tutorial

This section is a tutorial that explains the steps necessary to get a control program running in the State Logic Processor. This section covers configuring the Series 90–70 CPU with Logicmaster 90 and how to load, execute, monitor, and interact with an ECLiPS control program.

To demonstrate these features, this section uses the Conveyor program completed in the **Programming Tutorial** section of this manual. The I/O definitions of this program may have to be changed to be consistent with available I/O hardware or the simulation mode can be used to run the program without being attached to any I/O hardware.

Most of the explanations of this section describe the use of ECLiPS's Debug Mode. To view in ECLiPS all of the topics of this section, ECLiPS must be connected to a Series 90–70 State Logic Processor through the programming port. Not all of the Debug Mode options are discussed in this section. For a thorough explanation of all Debug Mode menu options see the Debug Mode part of the Reference Section of this manual.

Setting Up the Series 90–70 State Logic Control System

The first step in getting your control program executing in the Series 90–70 State Logic control system is to setup the CPU of the system. Use Logicmaster 90 to configure the Series 90–70 PLC for each of the modules used in the system. Use the serial cable plus the 9–pin to 15–pin adaptor included with ECLiPS, to connect the computer running Logicmaster 90 to the Series 90–70 CPU. Refer to the Logicmaster 90 User's Manual for information about configuring the CPU and other modules used in the system.

Configuring the State Logic Processor

Use this description of configuring the State Logic Processor (SLP) as an example for configuring the other modules in your system. There are obviously differences between modules, but this description should get you started in the right direction.

Use the Logicmaster 90 configuration software to add the SLP to the Series 90 I/O configuration. Logicmaster 90 is used to describe the modules present in the PLC racks. Rack and slot location and other features for each module are entered by completing setup screens that describe the modules in a rack. This tutorial describes how to execute programs without using any I/O modules, so you may want to have just a CPU and an SLP in your Series 90–70 control system for the demonstration purposes of this tutorial.

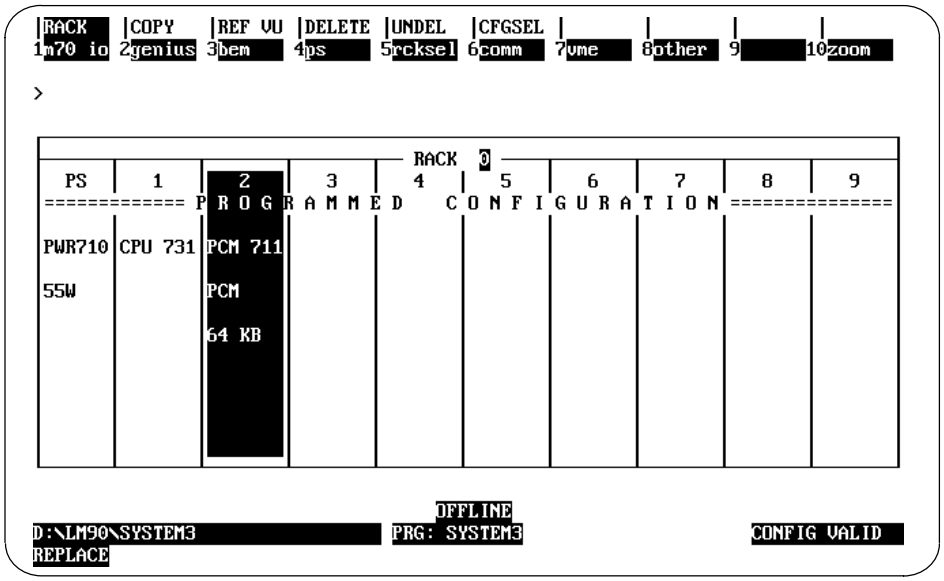


Figure 6-1. Sample Logicmaster Configuration Screen

From the main menu of the Logicmaster 90 configuration software, press I/O <F1>. The screen displays a representation of the modules in a rack. To add an SLP to the configuration, highlight the desired slot, then press Other <F8> and then PCM <F1>.

Now press Zoom <F10> to view the current configuration. Press <Enter> to enter the highlighted catalog number and display the PCM detail screen.

Now set the Configuration Mode to NONE for the 90–70 SLP. First highlight the Config Mode option and repeatedly press the <Tab> key until NONE is displayed on the screen. The serial ports are under program control and the parameters are initialized by the State Logic Processor.

Now press the <Esc> key to save the configuration and return to the rack display. The display should now show a PCM in the correct slot. Send the configuration to the PLC CPU and the configuration is complete.

Enter the Initialization Ladder Program

Because many of the CPU memory types are retentive (hold their value over a power outage), there must be a Ladder Logic program in the CPU which clears memory locations on power up, so that outputs that are ON when power is lost are not ON again when power returns. The program to clear the memory locations is included on the ECLiPS distribution disks. Follow the instructions in the Reference Section of this manual on Clearing Outputs on Power Up. This section is in the Series 90–70 State Logic Control System part, check the Table of Contents for the Reference section.

Enable Outputs and Start Program Running

During program execution the CPU must be in run mode with the outputs enabled. Use Logicmaster 90 to set the status of the CPU before executing the State Logic program.

To start the Series 90–70 CPU running the outputs with output scanning enabled follow these steps from the Logicmaster 90 first screen.

1. Press <F2> for the configuration package.
2. Press <F3> for the PLC Control and Status Screen.
3. Press <F1> for the RUN / STOP PLC Screen.
4. Press <TAB> until RUN / OUT EN is displayed in the highlighted block.

DownLoad a Program

To download a completed program to the State Engine, the ECLiPS computer must be connected with a serial cable to the SLP programmimg port, which is port 1 by default. Use the serial cable included with ECLiPS along with the appropriate adaptors for your system.

To download the program, choose the PROJECT choice from the Program Mode menu, then the “Download Project to the Controller” option. ECLiPS now provides the option of checking the program for undefined words. Choose either yes or no. If there are any errors in the program a message is displayed in the bottom bar of the screen. Use <Alt + F> to access help information describing the error. Follow this procedure to download the Conveyor program discussed in the **Programming Tutorial** section of this manual.

After a successful download a screen displaying the statistics of the project is displayed. Strike any key to go on line, ie. enter Debug Mode. The usual way to change modes is using the <F10> key. Try hitting the <F10> key now to change modes back to the Program Mode. Hit <F10> again to return to Debug Mode.

Debug Mode Screen

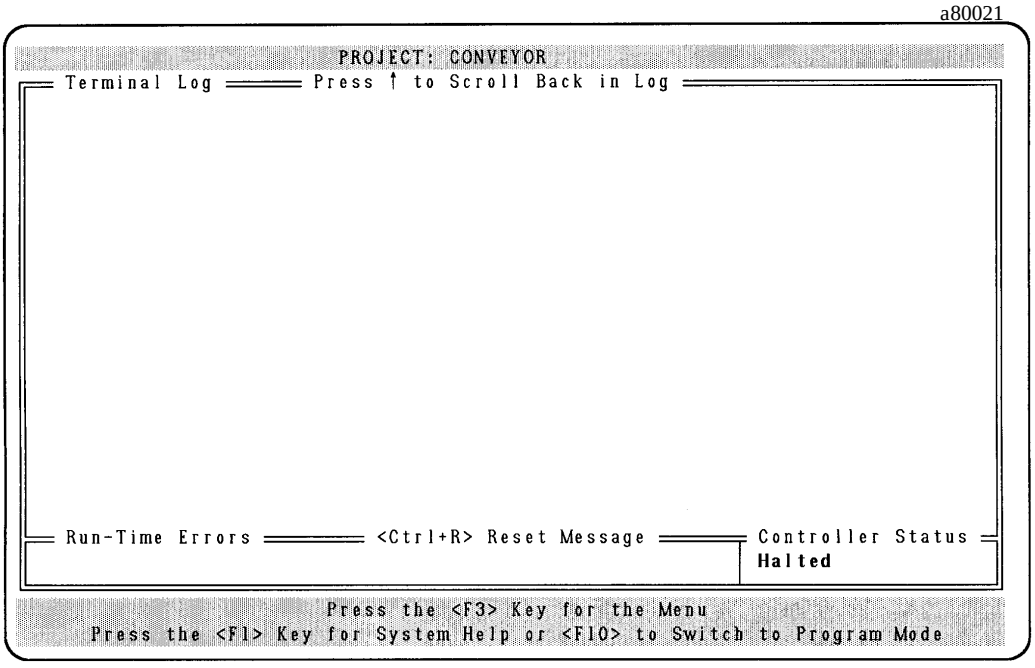


Figure 6-2. Debug Mode Screen

At the top of the screen is the familiar bar displaying the name of the project. As in the Program Mode the bottom bar shows the current use of some of the more important function keys. <F10> returns ECLiPS to program mode for editing the project. Notice that the help system continues to be accessed by pressing <F1>, and that the action of <F3> brings up the main menu for this mode just as it does in Program Mode.

Terminal Log

There is a window below the top bar titled “Terminal Log”. Any messages sent by the State Engine controller are displayed in this window. There are several different types of messages that the State Engine may send to the terminal log.

All programmed Write Terms which send characters to the serial port that connects ECLiPS to the State Engine are displayed in the Terminal Log. These messages may be operator instructions or menus for the operator to enter information into the control program.

All error messages are written into the terminal log with the time that the event occurs. This is an important record of events for troubleshooting or debugging a program, because messages displayed in the Run–T ime Error window overwrite the previous message, so that messages may be lost before they can be noted in this window.

The Terminal Log also records together with the current time, the response of the State Engine to the Debug Mode commands (Display, Change, and Force).

The first message is displayed at the top of this window with the next messages appearing below the previous ones. The oldest messages scroll out the top of the window when the window fills. Messages scrolled off the screen can still be viewed by pressing the up arrow key. After scrolling, you must press <End> to exit the scroll mode and again have access to the Debug Mode menu.

Error and Status Windows

There are two windows on the line below the Terminal Log window. The one on the left displays any run time error messages such as a divide by zero message or an open circuit fault message. Notice that the error message is cleared by pressing <Ctrl + R>. The smaller window on the right displays the current status of the controller, ie. running or halted. The status window also indicates when a discrete or analog I/O point is forced.

Put SLP in Run Mode

Press <F3> to view the menu, then select the "Project" option. Use this menu to put the SLP in Simulation Mode and start the Conveyor program running. Use the Simulation Mode to run the program without interacting with any I/O hardware.

Toggle Simulation Mode

Simulation Mode is used to disconnect the actual I/O from a running program. This mode is useful to test a program before actually connecting to the real world and to troubleshoot an existing program by disconnecting the machine from the State Engine controller. If you are using simulation mode for this tutorial, select this option from the PROJECT menu now. The controller must be in Halt Mode to activate the Simulation Mode.

RUN or HALT Program

Use these menu options on the PROJECT menu to start or stop program execution. Start the program running now.

Controlling and Observing the Fill Station

This section describes the features available to control and examine program execution. These examples assume that the State Logic Processor is executing in Simulation Mode, although these exercises are more informative using real world I/O.

The program must be exercised to observe how it executes. First lets create a monitor table to watch the program change States.

Monitor

Bring up the menu with the <F3> key. Next press <M> to get the Monitor menu. Select the "Add a New Monitor Table" option, and enter **States** for the name of the Monitor table. Select the "Current State of a Task" option from the list of categories to monitor. One at a time select each of the Task Names to monitor by highlighting the name in the list and pressing <Enter>. Now continue to press <Esc> until a monitor table is displayed at the bottom of the screen.

View

Now we are ready to have the program go through its steps. First lets display the program. Select the "View" option from the menu, and then select the "View English Text in Current Task Group" option. Notice <Ctrl + V> is the hot key for this selection.

With the arrow key move the cursor to the word Can_At_Fill and press the <Alt + F> key as shown in the bottom bar for functions. Choose the "Look up the Current Word and Display its Value" option. Notice the resulting window displays the word definition plus its current value. This option is also invoked by pressing the hot key <F4>.

Press <C> to change the value. From the resulting window force Can_At_Fill ON. Notice that the Fill Station Task is now in the Pour_Chem_1 State. Now move the cursor to the word Chem_Valve_1 and press the hot key <F4>. Chem_Valve_1 is ON as it should be when Pour_Chem_1 State of the Fill_Station Task is active.

Now move the cursor to the Fill_Weight_Input word and press <F4>, then change the analog input to 20.1. The current State is now Pour_Chem_2. Check the condition of Chem_Valve_2 and Chem_Valve_1. Change Fill_Weight_Input to 30.1 to go to the Wait_For_Can_Removal State. Put the cursor on Can_At_Fill again and clear the forced condition so that Fill_Station Task goes back to the PowerUp State. Remember to change the Fill_Weight_Input back to zero before repeating this cycle.

Controlling and Observing the Mix Station

In this section we explore other ways to change elements of the program and watch the results of those changes. First lets create a more detailed monitor table. Select the "Add a New Monitor Table" option again and call it **Mix Station**.

Enter all of the digital points used in the Mix Station Task:

Can_At_Mix
Mixer_Down_Motor
Mixer_Down_Switch
Mixer_Motor
Mixer_Up_Motor
Mixer_Up_Switch

Change

Now select the "Change" option from the menu. This option allows changes to other elements of the program. Select the "Current State of a Task" option. Now select the Mix_Station from the list of Tasks , and then select Lower_Mixer from the list of States. A message is displayed in the Terminal Log, stating the action that was executed and the current time. Notice that the Mixer_Down_Motor is now ON. Also note that if this State stays active for more than 20 seconds the Max_Time message is displayed in the Run–T ime Error window and in the Terminal Log.

Display

Now select the "Display" option from the menu. Several types of program information can be displayed from this menu. Select the "Current State of a Task" option, and then the Mix_Station Task from the list of Tasks. The display in the Terminal Log shows that the Current State is Lower_Mixer and the time.

Force

Now select the "Force" option from the menu and then "Set or Modify Force List". A window appears that shows all of the items forced and their current value. Press <Ins> to add an item to the list, and then select "Digital Points". Select the Mixer_Down_Switch and force it ON. The Mix_Station Task goes to the Mix_Chemicals State, notice the Mixer_Motor goes ON for 10 seconds and then the Mixer_Up_Motor comes ON. Force the Mixer_Up_Switch ON and the program goes to Wait_For_Can_Removal State and on the next scan to the PowerUp State, since the Can_At_Mix input is OFF.

Trace

To see a history of the execution of the program, select the "Trace" option from the menu. Select the "Upload Trace from Controller" option to see the Trace display.

The top line gives the time and date of the upload. Next a table of State changes is displayed, showing the starting State and the resulting State and which Task, with the most current State change at the top of the list.

The Trace display is a valuable tool for debugging and troubleshooting a program. The Tasks for which State changes are displayed can be changed so that unwanted changes are not included.

PLC I/O

It is often desirable to see what I/O memory locations are assigned to the program names. To inspect the names and definitions, select "PLCI/O" from the menu. Select the memory type and then the name from the list for the form that lists the I/O name, type, number, and input or output.

PID Loop Tuning Screen

This option only appears in the Debug Mode menu when the program loaded into the State Engine has a PID loop defined. After choosing this option, a list of defined PID loops is displayed. Select the desired PID loop from the list to use the tuning screen.

The tuning screen displays the values of the PID parameters and the status of the loop. The values are continuously updated as they change. Use this screen to tune the loop by changing parameter values.

Viewing and Clearing CPU Faults

Select the option for CPU faults to list the current fault table in the CPU. Both the PLC faults and the I/O faults can be displayed. You may clear the fault tables from this display. Note that the fault tables are not available when running in simulation mode.

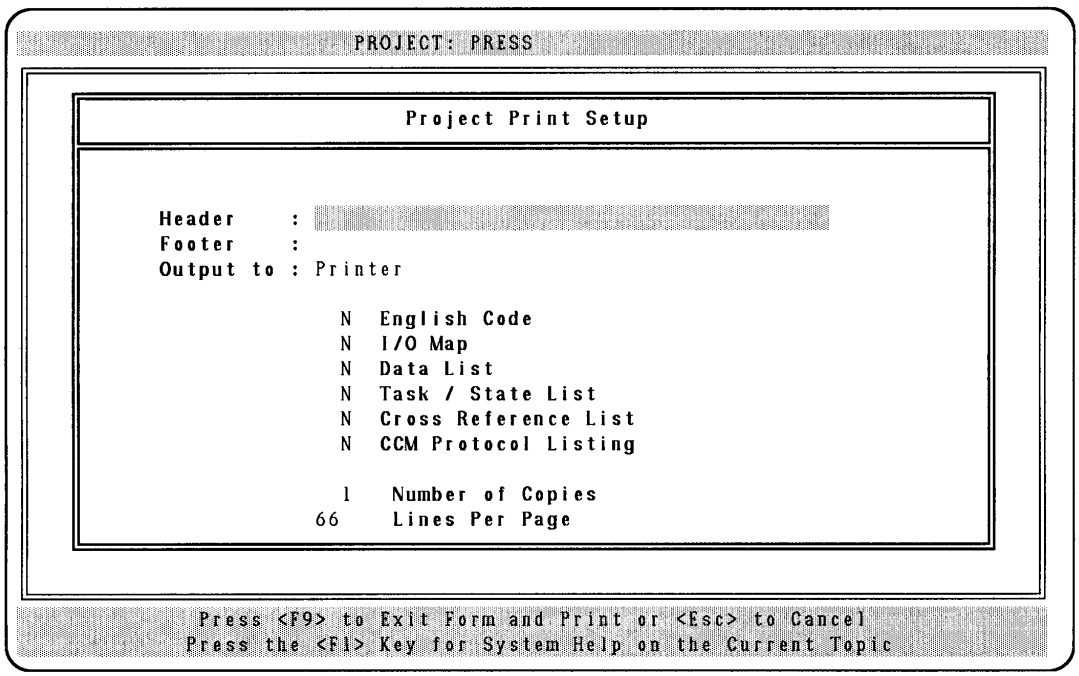
Chapter 7 Creating Program Documentation

This section describes how to use the ECLiPS program documentation features. The topics covered are printing program documentation, including the English program, I/O map, Data Listing, Task and State listing, and Cross Reference Listing. Printing information for using the CCM protocol and documenting the ECLiPS program are also covered.

Program Documentation Features

ECLiPS provides excellent self documenting features that are easily selected and printed from a Project Print Setup screen. To get into the Project Print Setup screen select <F3> from the Program mode of ECLiPS. From the menu select "Project" and then "Print" from the next menu. The following screen appears:

a80022



Header/Footer

The Header entry is printed on the top of each page of documentation. The footer entry is printed on the bottom of each page of documentation. Type the desired header and footer information in the highlighted box. The Header and Footer will accept up to 40 characters of information.

Directing the Output

The "Output To:" option specifies where the documentation is to be sent. If "Printer" is entered for this option, the documentation is sent to the printer connected to the parallel port. ECLiPS interprets any other name entered for this option to be a file name and the selected documentation options are sent to a disk to be stored in a file.

Documentation Options

This section describes each of the documentation options and shows a sample of the documentation produced when the option is selected. The Up and Down arrow keys are used to move the highlighted cursor to the options. To have a selection printed with the documentation package type a <Y> next to the option to be printed or a <N> to suppress a selection from being printed. Actually pressing any other key causes the form to toggle the display between Y and N for the highlighted option. For the "Number of Copies" and lines per page options type in the appropriate number. Press <ENTER> after selecting the number of packages to print. Press <F9> to start printing the documentation out the host computer's parallel port.

English Code

The ECLiPS program is printed when this option is selected. The name of the project and the date and time that the program was last modified is printed at the top of the page. A page number is printed at the bottom of the page. An example is listed below:

```
Project:  CONVEYOR    Last Modified:  8–28–90  8:56

Task:  Fill_Station

   State:  PowerUp
      If Can_At_Fill is on, go to Pour_Chem_1 State.

   State:  Pour_Chem_1
      Open Chem_Valve_2.
      If Fill_Weight_Input is above 30 pounds,
      Then go to Wait_for_Can_Removal.
                                  •
                                  •
                                  •
    Riverside Plan              Page 1
```

English Code

I/O Map

The I/O Map is a list of all of the names of CPU memory types defined in the State Logic program. The types included are %I, %Q, %AI, %AQ, %T, %M, %G, %S, %SA, %SB, %SC, and %R. The defined names are listed in numerical order according to type. The reference number for each name is also listed as follows:

```
I/O Map    Project:  CONVEYOR     Last Modified:  8–28–90  8:56

%I Discrete Inputs
        %I1     : Can_In_Place
        %I2     : Can_At_Fill
        %I3     : Can_At_Mix
        %I4     : Mixer_Down_Switch
        %I5     : Mixer_Up_Switch

%Q Discrete Outputs
        %Q1     : Chem_Valve_1
        %Q2     : Chem_Valve_2
        %Q3     : Mixer_Down_Motor
        %Q4     : Mixer_Motor
        %Q5     : Mixer_Up_Motor
        %Q6     : Conveyor_Motor

%AI Analog Inputs
        %AI1    : Fill_Weight_Input

Riverside Plant                 Page 1
```

I/O Map

Data List

The data list option prints all of the names of data elements that have been defined in the program. The data list groups the names according to the type of element and each group is listed in alphabetical order. The types are listed in the following table:

Digital Point Name
Analog Channel Name
Floating Point Variable
Integer Variable
String Variable
Character Variable
Internal Flag

Types of Data Elements Included in Data List

The following is an example of the Date List print out for the Conveyor program.

```
Data List   Project:  CONVEYOR    Last Modified:   8–28–90   8:56
                Digital Point Names

Can_At_Fill, %I2                  Can_At_Mix, %I3
Can_In_Place, %I1                 Chem_Valve_1, %Q1
Chem_Valve_2, %Q2                 Conveyor_Motor, %Q6
Mixer_Down_Motor, %Q3             Mixer_Down_Switch, %I4
Mixer_Motor, %Q4                  Mixer_Up_Switch, %I5
Mixer_Up_Motor, %Q5



                Analog Channel Names
Fill_Weight_Input, %AI1
```

Data List

Task / State List

The Task / State option prints each Task name of the program followed by all of the State names in that Task. This listing produces a good overview of the process especially when the Task and State names are descriptive of the operation performed. An example is listed below:

```
Task State List  Project:  CONVEYOR  Last Modified:  8–28–90  8:56
TASK: FILL_STATION
      STATE:  POWERUP
      STATE:  POUR_CHEM_1
      STATE:  POUR_CHEM_2
      STATE:  WAIT_FOR_CAN_REMOVAL

TASK: MIX_STATION
      STATE:  POWERUP
      STATE: LOWER_MIXER  MAX_TIME 20
      STATE:  MIX_CHEMICALS
      STATE: RAISE_MIXER  MAX_TIME 30
      STATE:  WAIT_FOR_CAN_REMOVAL

TASK:  CONVEYOR
      STATE:  POWERUP
      STATE:  FILL_STATION_WAIT
      STATE:  MIX_STATION_WAIT
      STATE:  START_CONVEYOR
      STATE:  MOVING_CANS
```

Task State List

Cross Reference List

The Cross Reference List produces a list of all the data elements of the program. Indented below each element is printed every Task / State combination where that element is used in the program.

This print out is a valuable tool when debugging or troubleshooting a program. An example of a Cross Reference List is shown below:

```
Cross Reference List  Project: CONVEYOR Last Modified 8–28–90  8:56
Can_At_Fill:
     Task:  Fill_Station
          State:  PowerUp
          State:  Wait_for_Can_Removal
     Task:  Conveyor
          State:  Moving_Cans

Can_At_Mix:
     Task:  Mix_Station
          State:  PowerUp
          State:  Wait_for_Can_Removal
     Task:  Conveyor
          State:  Moving_Cans

Can_In_Place:
     Task:  Conveyor
          State:  PowerUp
          State:  Start_Conveyor
```

Cross Reference List

CCM Protocol Listing

The “CCM Protocol Listing” option on the print setup form produces data for use in communicating to the State Engine using the CCM protocol. This protocol is typically used by host computers and computers running Graphical User Interface (GUI) programs to extract or change information in the State Engine. The CCM information printed lists the CCM type, the CCM type number, and the CCM number of the elements of the ECLiPS program.

The following example uses the Conveyor program created in the **Programming Tutorial** section:

CCM List Project:CONVEYOR Last Modified: 11–11–91 18:08

Digital Names

Name	CCM Type	CCM Type #	CCM Number
Can_At_Fill	DiscreteI/O	2 or 3	33
Can_At_Mix	DiscreteI/O	2 or 3	36
Can_In_Place	DiscreteI/O	2 or 3	43
		•	
		•	
		•	

Analog Channel Names

Name	CCM Type	CCM Type #	CCM Number
Fill_Weight_Input	Register	1	41

Task and State Names

Task Name	CCM Type	CCM Type #	CCM Number
FILL_STATION	Register	1	9501
State Name		State Number	
POWERUP		1	
POUR_CHEM_1		2	
POUR_CHEM_2		3	
WAIT_FOR_CAN_REMOVAL		4	

Page 1

CCM Protocol Listing

Commenting the ECLiPS Program

All ECLiPS programs inherently include automatic documentation due to the descriptive nature of the English program. There are times when the description of the process requires additional explanation. Explanations or comments may be entered at any line of the program.

To create a comment in ECLiPS simply type an exclamation mark (!) followed by the commenting text. All text following the **(!**) on the line, is considered to be a comment and is ignored by the ECLiPS translator. Some commenting techniques are shown in the following program segment.

```
!!!
```

Chapter 8 Reference

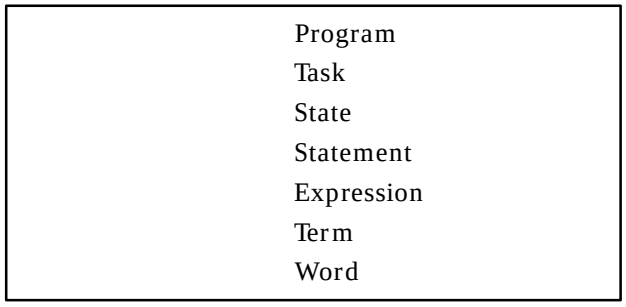
This section is designed to provide information about the details of the ECLiPS control language and how to use the ECLiPS software package. The section starts with a description of the State Logic Language and Keywords then discusses the setup of the Series 90–70 control system, then each of the menu options is explained, and finally the ECLiPS specifications are displayed.

Language Description

This section describes the details of using English to create a control program. The detailed information describes how to structure the English program using keywords, filler words, names, references to I/O, programming PID Loops, grammatical rules, and special "perform" functions.

Program Structure

There is a hierarchy in the structure of an ECLiPS program. Each program is divided into 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.



ECLiPS Program Hierarchy

This hierarchy of Tasks, States, and Statements are explained in sections 2 and 3 of this manual. A program is a collection of Tasks, and 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.

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.

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.

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.

There are two types of **expressions**: Conditional and Functional. Functional expressions describe some action that the controller executes. Conditional expressions describe a condition which must be true for the functional expression in the statement to be executed. Expressions consist of one or more Terms connected with logical (AND / OR) operators.

Terms express the fundamental actions and status tests of the program. Terms are built from words.

All words are classified into one of three categories: Keywords, Names, and Filler words. All filler words are ignored when the program is translated into a control program, therefore the keywords and names are the only important words in a Term. Words are separated by spaces.

ECLiPS comes with a set of keywords which are appropriate for many control applications. The programmer can define synonyms for the keywords and even change the default keywords. The set of filler words is similarly controlled by the programmer.

Language Structure Notational Conventions

In the following sections there are several examples of how to use the various State Logic expressions, plus a description of the different ways to use the expressions. There are so many different combinations possible that it is necessary to use some notational conventions to rigorously define all of the possible structures.

<u>Underline</u>	– 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

Language Structure Notational Conventions

The State Logic hierarchy terms described in the previous section are now specified using these notational conventions.

Program = { <Task> }

A program is one or more Tasks.

Task = Task: <Task Name> [Start_In_Last_State] { <State> }

A Task is the keyword Task followed by a colon and then optionally the keyword, Start_In_Last_State, then a Task name and one or more States.

State = State: <State Name> [{ <Statement> }]

A State is the keyword State followed by a colon, then a State name and optionally one or more Statements. A State does not require any Statements.

Statement = ([<Conditional Expression>] <Functional Expression>) |
(<Functional Expression> [<Conditional Expression>])

Every Statement must have a Functional Expression and may or may not have a Conditional Expression. The Conditional Expression may come before or after the Functional Expression.

Functional Expressions

Functional Expressions are composed of one or more Functional Terms. There is no limit to the number of Functional Terms used in a functional expression.

Functional Expression = { Functional Term }

Functional Terms describe any action or changes in the control system; turning on outputs, sending messages out a serial port, making a new State the active State of a Task, setting analog output values, and changing variable values, setting serial port parameters, starting and stopping PID loop execution, and doing specialized Perform functions are all performed by Functional Terms.

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 >

Turning ON Discretes (Actuate Term)

The Actuate Term is used to turn on Digital I/O Points and Internal Flags. This Term starts with the keyword Actuate followed by one or more discrete names.

Turn On Discrete Term = Actuate	**{ <Digital I/O Name> |** **< Internal Flag Name > }**

- Actuate the Ready_Light.
- Start Pump_1 and Pump2.
- Energize Clamp_1, Clamp_2, Clamp_3 and Clamp_Flag.

Assigning Values (Make, Math–Assignment, Set_Bit/Clear_Bit)

To assign values use the Make Term, Math–Assignment Terms,Set_Bit/Clear_BitTerms. These Terms assign values to variables and analog I/O points.

Assign Values Term =	**< Make Term > |** **< Math–Assignment Term > |** **< Set_Bit/Clear_Bit Term >**

Make Term

The Make Term is used to assign a value to a variable or analog I/O point. The Term starts with the keyword Make an is followed by a variable or analog name, the keyword equal, then a number or a calculated value.

Make Term = Make	**< Numeric Assignment Term > |** **< Character Assignment Term > |** **< String Assignment Term>**

Numeric Assignment Term =	**(<Numeric Variable Name> |** **<Analog I/O Name> |** **<PID Value>) equal < Numeric Value >**

Character Assignment Term =	**<Character Variable Name> equal** **<Character Value>**

String Assignment Term =	**<String Variable Name> equal <String 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 how to do mathematical calculations.

Math–Assignment Term

This Term does a simple arithmetic operation on a variable value then assigns the new value to the variable. The four terms are add, subtract, multiply, and divide.

Math–Assignment Term = < Add Term > |
< Subtract Term > |
< Multiply Term > |
< Divide Term >

The Add Term is the keyword Add followed by a number or variable name then a variable name.

Add Term = Add (< Numeric Constant > | < Variable Name>)
< Variable Name >

- Add 1 to Parts_Count
- Add Second_Shift_Parts_Count to Total_Parts_Count

The Subtract Term is the keyword Subtract followed by a number or numeric variable name then a variable name.

Subtract Term = Subtract (< Numeric Constant > | < Variable Name >)
< Variable Name >

- Subtract 2.78 from Starting_Value
- Subtract Tare_Weight from Test_Weight.

The Multiply Term is the keyword Multiply followed by a variable name then a number or variable name.

Multiply Term = Multiply < Variable Name >
(< Numeric Constant > | < Variable Name >)

- Multiply Parts_Lost by 2
- Multiply Machine_Strokes by Strokes_Per_Cycle

The Divide Term is the keyword Divide followed by a variable name then a number or variable name.

Divide Term = Divide	**< Variable Name >**
	(< Numeric Constant > | < Variable Name >)

- Divide Right_Side_Length by 4.5
- Divide Box_Volumn by Volumn_Adjustment

Set_Bit/Clear_Bit Term

The Set_Bit / Clear_Bit Term is used to set a High Speed Counter Command bit or assign individual bits of a variable. First the Set_Bit or Clear_Bit keyword is used. If a variable bit is being set the variable name is followed by the zero based bit number.

Set_Bit/Clear_Bit Term =	**(Set_Bit | Clear_Bit)**
	(<Integer Variable Name> <Integer Number>)

- Set_Bit Transfer_Status 2
- Clear_Bit Tac_Register

Changing Active States Term

There are three ways of changing the active State of a Task. A Task can change its own State, change the active State of another Task, or cause any Task to become inactive or resume its previous active State.

Change State Term =	**(Go <State Name>) |**
	(Make <Task Name> equal <State Name>) |
	((Suspend_Task | Resume_Task) <Task Name>

The GO term is the means by which a Task transitions to another State. Only the Go and the State name are mandatory. All other words are optional as in the following examples. The Go may appear in any Statement but there may only be one Go per Statement.

- Go to the Forward_Motion State.

Tasks control other Tasks by merely setting the Task to a new State value.

- Put the Assembly_Control Task into the Emergency_Stop State.

In this example, put is a synonym for make and into is a synonym for equal.

The Suspend_Task and Resume_Task Keywords are also used to change the current State of a Task. The Suspend_Task keyword puts the named Task into the Inactive State. The operating system gives every Task a State named Inactive. When a Task is in the Inactive State, this Task performs no activity and the only way to exit this State is for another Task to change its current State. The inactive State can also be used with the GO keyword.

The Resume_Task keyword causes the named Task to go to the State that was active before being suspended.

- If Water_Level is above 45.5 feet then Suspend_Task Fill_Tank.
- If Water_Level is below 43.8 feet Resume_Task Fill_Tank.

Sending Serial Data (Write Term)

The Write Term is the keyword Write followed by data to send inside double quotes. Optionally a communications port name or R_Register numbers may be specified following the data to be sent. If no port name or R_Register is specified the data is sent to the programming port.

Send Serial Data Term = Write " <Serial Data> " [<Port Name> | R_Register <Number>]

- Write "Push Start Button" to Operator_Control.
- Write "ERROR Number 16" to R_Register 10.

The data is directed to either serial port which is named or to a series of R_Registers. If the data is directed to R_Registers the string of characters are stored in the Series 90 register reference table beginning at the %R register number which follows the R_Register keyword. In the example above the characters in the string, "ERROR Number 16", are stored in successive bytes starting at %R10.

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 Term. Multiple line messages can be formatted in the program exactly as they appear on an terminal 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 from this Write Term 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 7 full (80 character) lines of text.

Variable values are sent out the port by preceding a variable name with a "%".

- Write "Current parts count is %Part_Count." to Operator_Terminal.

If Part_Count is 10 at the time the Write Term is sent, this Statement displays the following line on the screen connected to the port named Operator_Terminal.

Current parts count is 10.

Formatting Characters that are used with the Write Term follow:

%NOCRLF – Write Terms always send a carriage return line feed pair following each message. Use this formatting feature suppresses these terminating 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
%CHR(X) – The ASCII character for the value in the "()" is sent
%#X – The ASCII character for the hexadecimal value X is sent
• The "|" character is used to place two words together without any spaces in between them. For example, "%Pressurepsi" would look like one long variable name to ECLiPS and would yield an error message. But "%Pressure|psi" would yield the desired result of the value 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.

PID Loops Control Terms (Start_PID, Stop_PID)

PID Loop control Statements start with the keywords StartPID or StopPID, 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.

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>]

- Start_PID Oven_1.
- Stop_PID KILN5 with 456.29.

Change Serial Port Configuration Term

The Term 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.

Port Configuration Term = Set_Commport <Port Name>
<Parameter Value List>

This Functional Term 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, select the "Insert Reconfiguration Data for the Port" option from the next menu. The entire Term is entered into the program at the current cursor location.

Perform Function Term

This Term is the Perform keyword followed by the function name, the keyword with and then a list of parameters and values.

Perform Function Term = Perform	**<Function Name>**
	with <Parameter Value List>

This Term is entered automatically by ECLiPS at the end of the State where the cursor is located. First select the "Add" option from the menu, then the "Add a Perform Function" option. Fill in the blanks that are displayed after selecting the function desired.

Conditional Expressions

Conditional Expressions are used for receiving information in a serial port and for testing a value or status. The Functional Expression in a Statement is executed only if the Conditional Expression in the Statement is true. The usual action that is dependent on a conditional is a transition to another State.

Conditional Expression =	**< Test Conditional > |**
	< Character Input Conditional >

Test Conditional

The test conditional starts with the keyword, If, and is followed by one of the four types of Test Conditional Terms. The keyword NOT can precede the conditional term. Several Conditional Terms can be joined by the AND and OR keywords. The AND Terms have lower precedence and are therefore executed first. The order of execution can be changed by use of parenthesis and parenthesis can be nested.

Test Conditional = If	**[NOT] <Conditional Term>**
	[{ (OR | AND) [NOT] <Conditional Term> }]

The type of Conditional Test Terms are Digital, Timer, Relational, and Current State.

Conditional Term =	**<Digital Test Conditional> |**
	<Timer Test Conditional> |
	<Relational Test Conditional> |
	<Current State Test Conditional>

Digital Conditional

The Digital Conditional Term tests the status of digital I/O circuits and internal flags. This term is a discrete name followed by the keyword ON or OFF.

Digital Test Conditional =	**(<Digital I/O Name> | <Flag Name>)**
	[{ (AND | OR)
	(<Digital I/O Name> | <Flag Name>) }]
	(ON | OFF)

- The following notational conventions are used throughout this section to rigorously define the required structure:
- 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 have a lower precedence and are therefore executed first.

Timer Conditional Term

The Timer Conditional is a number or variable followed by the keyword SECONDS. The timer has a resolution of 1/100 of a second and the number used to indicate the number of seconds can be a floating point number.

Timer Conditional = (<Numeric Constant> | <Integer Variable Name>) seconds

- 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 value of 100 would indicate a time 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 it is scanned.

- If Track_Monitor is ON and 5.3 seconds have passed . . .

The timer above refers to the time that the State has been active and is not influenced by the condition of the Track_Monitor.

The timer number must be in the range of 0.01 to 600.00 seconds or 10 minutes. When using an integer variable, the variable value must be between 0 and 32767.

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.
Wait 600 seconds then go to Timer_Counter State.

State: Timer_Counter
Add 1 to Ten_Minute_Counter and go to Heater_On_One_Hour State.

Also note that there is a perform function that can be used to measure elapsed time.

Relational Conditional Term

Relational Terms test a variable or analog values. The Term is a value, followed by a relational operator, then another value. The values tested can be numbers, calculations, variable names, and analog names.

Relational Test Conditional = <Numeric Relational Term> |
<Character Relational Term> |
<String Relational Term>

Numeric Relational Term = <Numeric Value> <Relational Operator>
<Numeric Value>

- If Parts_Count = 500 . . .
- If Flow_Meter_Input is above Flow_High_Limit . . .
- If Canister_Pressure – Atmosphere <= Pressure_Limit – Safety_Margin . . .

See the section, Mathematical Calculations, for a discussion on how to use calculations.

Character Relational Term = <Character Value> **(equal | not_equal)**
<Character Value>

- If Test_Char not_equal '@' . . .

String Relational Term = <String Value> (**equal** | **not_equal**) <String Value>

- If String_Entry equal "Formula 1" . . .

String Control Characters

Any ASCII character can be imbedded in a String constant by using the % character. The % is followed by the # and then the two digit hexadecimal number for the ASCII character.

If String Input equals "%#01Start " . . .

The previous conditional expression checks the string variable String Input to see if the characters of the string are ASCII 1 followed by the word Start.

Current State Conditional Term

The Current State Conditional is a Task Name followed by the keywords EQUAL or Not_Equal and then a State Name. This conditional is used to test the current State of another Task.

Current State Conditional =	**<Task Name>**
	(equal | not_equal)
	<State Name>

- If Pump_Monitor Task is in the Backwash State . . .

Complex Conditionals

Conditionals can be preceded by the NOT keyword and several can be joined by the AND or OR keywords and parenthesis can be used to change order of evaluation.

- If Hydraulic_Pump_Control Task is in the Over_Pressure State or Hydraulic_Pressure 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 not_in Pouring State or not Mold_Number above 67

Character 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.

Another option is to receive the input from R Registers of CPU. The syntax for this option is again to use the keyword from followed by the keyword R_Register then the starting register number, the keyword for and then the number of registers to read. When using this option, the variable receiving the data must be a string variable. Each byte of the R_Registers read is stored in the string variable.

Character Input = Read	**<Variable Name>**
	[from <Communications Port Name> |
	(R_Register <Register Number> for
	<Integer Number>)]

- **Read Menu_Choice from Operator_Station . . .**
- **Read Error_String from R_Register 46 for 20 R_Registers . . .**

The first example reads data into the variable named, Menu_Choice, from the communications port named Operator_Station. The second example reads 40 characters into the string variable, Error_String, from the 20 CPU R Registers starting at %R46.

A GO Functional Term must always follow the character input conditional and there cannot be any other Terms in the Statement. If two character input conditionals for the same port are written in the same State it is unknown which conditional receives a message from the port.

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 conditional is true, when an end of message character is received at the port. The default end of message character is the carriage return, so that normally the input is completed when the <Enter> key is pressed. The end of message character may be changed by the Set_Commport Functional Expression.

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.

IMPORTANT: 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 Terms as in this assignment Term:

Make Pointer_Position = Last_Position * (Forward_Pressure + 345.8)

and in Conditional Terms as in this comparison term:

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 the precedence listed in the operator keyword table. 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.

Variables

Variables are used to store some information in memory. All variables are identified by a unique name that is assigned when the undefined words in the program are defined. Each variable can be configured to save the value over a power cycle or be initialized when the program is started. There are two main categories of variables, ASCII and numeric. Calculations may refer to the value stored in any of the numeric variables.

ASCII Variables

The ASCII variable types are Character, String and R Register. Character Variables store one character and use one byte of memory. String Variables store up to 80 characters and use 80 bytes of memory. 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 equals “abc%#1 Bxyz”.

Is an assignment to a String Variable to store the characters abc the escape character and then xyz.

The R Register Variables can also be used to store character information in the CPU %R registers. Use the Write term to send characters to the R Register variables and the Read term to retrieve characters from the Register locations. Two characters are stored in every %R location. Use the R Registers when the stored information needs to be used by either the CPU ladder program or some other device that can access the CPU memory locations.

Numeric Variables

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

Analog Variable are the analog l/O connected to the system. Analog values are floating point if they are scaled and integer if left unscaled. The scaling option is selected when the analog channel is defined.

Time Variables store the current time information from the SLP system clock. The reserved time variables are second, minute, hour, day, and month. These variables are integer values and should not be changed. To set the clock use Logicmaster 90 to set the CPU clock and then cycle power to the system. On power up the SLP clock is initialized. To keep the SLP clock in synch with the CPU, it is reset to the CPU clock once every day at midnight.

R_RegisterVariables are stored in the %R registers in the CPU of the system. Use R Registers to store information that needs to be used by the CPU ladder program or some other device that accesses the CPU memory. The R Register variables can be designated to be either integer or floating point type. An integer R_Register variable uses one %R register and a floating point R Register variable uses two %R registers.

Floating point and Integer Variables are stored in SLP memory locations. Use these variables when the values are used only in the SLP program.

Numerical Data Types

There are two numerical data types integer and floating point. Integer data is limited to the range of –32768 to 32767. Floating point data is limited to the range of +/–1.2E–38 to +/–3.4E+38 with an effective precision of seven decimal digits.

Integer constants are whole numbers in the range –32768 to 32767. Integer constants can be specified in hexadecimal format by preceding the number with '#', ie, #77FF. Floating point constants are numbers using decimal points, numbers outside the range for integers, or numbers using scientific notation.

Data types maybe 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. If a floating point value is assigned to a variable of integer variable 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.

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.

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

Language Structure Summary

Notational Conventions

The following notational conventions are used throughout this section to rigorously define the required structure:

Underline	– 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

Language Structure Notational Conventions

Program Hierarchy

Term	Syntax
Program	{ <Task> }
Task	Task: <Task Name> { <State> }
State	State: <State Name> [{ <Statement> }]
Statement	([<Conditional Expression>] <Functional Expression>) (<Functional Expression> [<Conditional Expression>])

Functional Structures

Term	Syntax
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** < Variable Name > (< Numeric Constant > | < Variable Name >)
Divide Term	**Divide** < Variable Name > (< Numeric Constant > | < 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> | **R_Register** <Number>]
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	Syntax
Conditional Expression	< Test Conditional > | < Character Input Conditional >
Character Input	**Read** <Variable Name> [**from** <Communications Port Name> | (**R_Register** <Register Number> **for** Integer Number>)]
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	Syntax
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

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, but there may be several in a State.
- If a "Read" Term is used in a Statement, it must be accompanied by a "Go" in the same Statement. There may be no other Terms in the Statement.
- Only one "Read" is allowed per State.
- A "Perform" function must be the only Term in a Statement.

Using the System Clock

The CPU has a clock that maintains the current month, day, day of the week, hour, minute, and second. These values are always available to the State Logic Processor through the Time Variables Month, Day, Day_of_Week, Hour, Minute, and Second.

These variables represent an integer values that may be used by program statements or accessed using the DISPLAY debug mode menu option. These are READ–ONLY values that cannot be change from the State Logic Processor. To change the clock settings use Logicmaster to change the time values in the CPU.

CAUTION: Any changes to the Time Variables while the program is running, may affect the execution of any timing Statements currently active in the program.

The current time values may also be saved using the Time Variables:

Make Start_Hour = Hour, Start_Minute = Minute, and Start_Second = Second.

Use this flexible method of saving the current time to create any type of elapsed time timer needed. Subtract the saved variables from the current Time Variables to get the elapsed time.

Number_Of_Hours = Hour – Start_Hour.

This Statement gives the number of hours that the clock has changed since the Start_Hour variable has been set. Be careful to account effects of other Time variable rollover, such as minutes going from 59 to 00.

Standard Predefined Keyword Set

This is the set of Keywords supplied with ECLiPS. Up to 10 synonyms may be added for each Keyword. The Default Keyword may also be changed. 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 have passed . . .
For, Until	Test conditions and values, actions are executed when the test returns a False condition. • Actuate Stop_Light **until** Parts_Count = 23. • Turn on Alarm_Light **for** 2 seconds then go to Reset.
Read	Get input from comm port or CPU Registers, must be used with GO term. • **Read** Name from Port_1 then go Display_Name. • **Read** Status_String from R_Register 69 for 23 R_Registers, then go to Display_Status State.
Wait	Time Delay, must be used with GO term. • **Wait** 2.34 seconds then go to Restart State.

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 = 56.
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.
State	Identifies the Name of State Logic States • **State**: PowerUp • go to the Motion **State.**
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.
Task	Identifies the Name of State Logic Tasks • **Task:** Main • If the Main **Task** is in the PowerUp State, go...
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_10.
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_215.
Suspend_Task	Stop the execution of a Task – no State is active. • If Level > alarm, **Suspend_Task** Automatic.
Resume_Task	Restart the execution of a task – the State that was active when the Task was suspended is made the active State. • If Level < alarm, **Resume_Task** Automatic.
Write	Send data out the comm port • **Write** "Error Message" to Operator_Console.

Mathematical 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: where –65535 <= exp <= 65535 • Make Overlay = **ARCTAN**(Hyp * 2).	2
COS(exp)	Cosine: 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 Simulated_Inp = Set_Time *** Random**	2
SIN(exp)	Sine: where –65535 <= exp <= 65535 • Make Vector1 = **SIN**(Strain_Gauge).	2
SQRT(exp)	Square Root: • Make Out_Pot = **SQRT**(Flow_Meter).	2
TAN(exp)	Tangent: where –65535 <= exp <= 65535 • Make Slope = **TAN**(TRIM * In_Flow).	2
^	Exponential Operator • Make Count = Amount ^ 2.	2
%, Modulus	Modulus Operator – integer operands only • If Count **%** 5 = 0, go...	
*, Times	Multiplication Operator • Make Count = Amount ***** 2.	3
/,Divided_By	Division Operator • Make Count = Amount **/** 2.	3
+, Plus	Addition Operator • Make Count = Amount + 2.	4
–, Minus	Subtraction Operator • Make Count = Amount **–** 2.	4
Bitwise_And	AND bits operator • Value = Code Bitwise_And Mask.	4
Bitwise_Or	OR bits operator • Setup = Code Bitwise_Or Mask.	4
=, Is, Equal, Equals, Into	Assignment Operator • Make Count **=** 5,	5

Relational Operators

Keyword, Synonyms	Meaning / • Examples	Precedence
= , Is, Equal, Equals, Into	Comparison Operator • If Count = 5, go...	5
< , Less, Under	Less than Operator • If Count < 5, go...	5
<= , =<	Less then or equal to 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**
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 to identify a port name. • Read Name **from Port**_1 then go to the Stretch State.
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 Time Term • Wait 3.2 **seconds**, go... • 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 = **Friday**, go...
With	Prefix for data that is needed by a function • Stop_PID Kiln_Temperature **with** 45.679 • Perform Get_User_Input **with** Clear_Screen = Y...
R_Register	Used to indicate %R registers in the Read or Write Term • Read Status_String from R_Register 69 for 23 R_Registers
Start_In_Last_STATE	Indicates that the Task should start in the state that was active when the program was stopped. • Task: Compressor_Control Start_In_Last_State

Filler Words

Filler words have no functionality, ie. they do not change the meaning of any of the statements in which they appear. Filler words should 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.

PID Loops

The State Engine controller provides the capabilities of modulating control through the use of the PID algorithm. Each State Engine provides the User with up to ten PID algorithms that are continuously executed at User selected time intervals. These PID algorithms can be connected to field inputs and outputs, or interconnected in cascaded and other fashions to implement the User's desired control strategy.

Those who are already familiar with traditional PID control, might want to skip the **PID Algorithm Philosophy** section and go directly to the **PID Summary** section. The **PID Summary** section is a summary of the features available with the PID controllers and is probably all those familiar with PID loops need read. For those Users needing or wanting a more detailed discussion, the **PID Algorithm Philosophy** section provides the details.

PID Algorithm Philosophy

The State Engine PID employs a traditional 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, and the resulting output is the action that should be taken by the process actuator.

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 part or term uses the Gain tuning constant. Its 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 product of the error signal times the Reset over time. 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's result immediately allows an error signal to have its full effect, then returns the term's value to zero as time goes on. The amount of the Derivative term output for a given error and the rate it decays 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. 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.

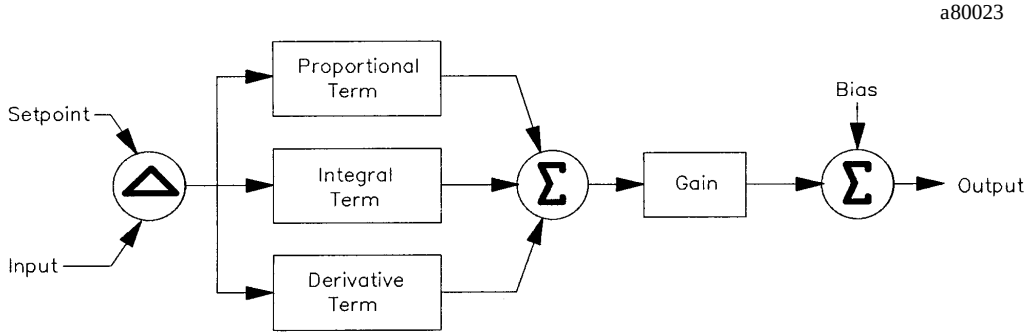


Figure 8-1. PID Algorithms

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 equals 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.

Simple PID Loop

Many applications only require a simple PID loop to achieve the desired control results. A process variable represented by an analog input is compared with a setpoint with the output of the PID controller being directly sent to a field actuator by means of an analog output.

To set up this simple loop, the User needs only choose the “Define PID” function from the “Define” menu in ECLiPS. The User then enters the name of the analog input, the name of the analog output, and the various initial settings for the tuning constants.

The setpoint and tuning constants can be changed while the process is on line by using the “Tuning” function in either the Debug mode of ECLiPS or OnTop.

Complex PID Control

In some cases a more 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.

a80024

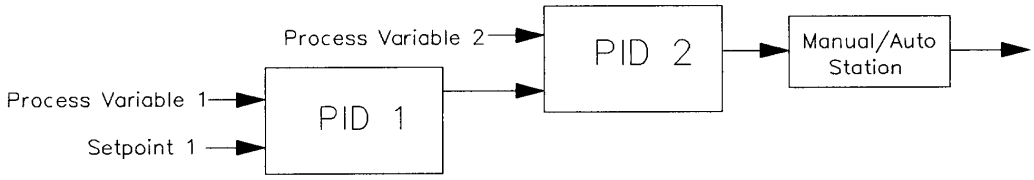


Figure 8-2. Cascaded PID Loops

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

Figure 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, and then by means of Raise and Lower pushbuttons, change the actual value of the Manual / Auto output. In the Auto mode, the value of the Manual / Auto station is equal to the output of the PID.

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 ten 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–R eset 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 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 PIDs can be accomplished without reset windup occurring and with bumpless transfers from one mode to another.

Tuning and Scaling

The Gain, Reset, and Rate constants can be adjusted to obtain the desired PID performance, such as speed of response and over shoot. In addition, each PID has a high limit and low limit that can be set to limit the output to less than its full range when desired. These limits automatically employ anti–reset windup.

Within the PID algorithm, all signals are treated as being 0 to 100%. Each input and the output and the high low limits, have scaling constants associated with them. Values given for this parameter are converted to a percentage of the range specified by the scaling constants.

If these constants are left blank at programming time, that input is assumed to be already 0 to 100%. Values given for this parameter are assumed to be a percentage which has already been scaled.

By using the scaling factors, the output can be scaled to have a live 0, that is go from –100% to +100%. This is a valuable tool at times when cascading PIDs and the upstream PID needs the ability to overcome and move the downstream PID across its full range.

PID Summary

The state engine provides up to ten PID algorithms. Each algorithm is identical and can be executed at User selected time intervals with the minimum interval being 1 second. The PID algorithms can each be independent or can be cascaded together with the output of one PID becoming the input to another. Each PID features built in bumpless transfer and anti–reset windup features.

Each PID is initialized in the Program mode of ECLiPS using either the LIST or DEFINE option from the program mode menu. PID Loops are tuned on–line in the Debug mode or using OnTOP. Each PID loop is given an alpha numeric name which can be up to twenty characters in length. All references to the PID loop use this name.

To start a PID Loop use the Start_PID keyword. For example the statement:

Start_PID Tank_Level.

starts the PID Loop name Tank_Level.

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 output to 10.

PID Parameters

Each PID loop is defined by the values of several parameters. These parameters are initialized when the PID loop is first defined, and can be changed by program statements during program execution or through the tuning forms provided in debug mode and OnTOP. The program refers to these parameters by specifying the name of the PID loop followed by the parameter keyword. For example, the ECLiPS program statement:

Make Tank_Level Setpoint = 45.

sets the setpoint of PID loop named Tank_Level to 45.

The following table lists each of the parameters with a description and keyword to identify the parameter in an ECLiPS program statement.

PID Loop Parameters

Parameter Action	Keyword	Description
Action Direct or Inverse	Loop_Action	Set to D or I to make PID integrate from 0 toward 100% if the setpoint> process variable (direct acting) or process variable > setpoint (inverse acting.)
Update Time	Update	Time interval between updates for this PID.
Gain	Gain	Gain for the PID
Reset	Reset	Reset constant for this PID
Rate	Rate	Rate constant for this PID
Setpoint	Setpoint	Name of the variable acting as setpoint or a value to use as the initial value.
Setpoint Max Scale	SP_Max	Engineering unit value for 100% scale or blank to use 100%
Setpoint Min Scale	SP_Min	Engineering unit value for 0% scale or blank to use 0%
Process Variable	Process_Var	Name of the process variable
Process Var Max Scale	PV_Max	Engineering unit value for 100% scale or blank to use 100%
Process Var Min Scale	PV_Min	Engineering unit value for 0% scale or blank to use 0%
Control Variable	Control_Var	The output of the PID loop – analog output channel or floating point variable.
Control Var Max Scale	CV_Max	Engineering unit value for 100% scale or blank to use 100%
Control Var Min Scale	CV_Min	Engineering unit value for 0% scale or blank to use 0%
Bias	Bias	Amount to be added to the Output
High Limit	High_Limit	Maximum allowable value for the Output
High Limit Max Scale	HL_Max	Engineering unit value for 100% scale or blank to use 100%
High Limit Min Scale	HL_Min	Engineering unit value for 0% scale or blank to use 0%
Low Limit	Low_Limit	Minimum allowable value for the Output.

Low Limit Max Scale	LL_Max	Engineering unit value for 100% scale or blank to use 100%
Low Limit Min Scale	LL_Min	Engineering unit value for 0% scale or blank to use 0%

There are also PID status and command bits associated with each PID loop. 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 an ECLiPS program, the PID name is used followed by the keyword for the bit.

The ECLiPS 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 ECLiPS statement:

Set_Bit Tank_Level Track_Mode.

makes the Tank_Level PID loop tack the output so that there is no error signal.

PID Command and Status Bits

Status or Command Bit	Keyword	Description
Block Up	Block_Up	When this bit is set, the PID does not integrate up for a positive error signal. Used for Anti_Reset Windup.
Block Down	Block_Down	When this bit is set, the PID does not integrate down for a negative error signal. Used for Anti_Reset Windup.
Track Mode	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 Status	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 Status	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.

PID Inputs

There are 5 inputs to the PID algorithm. Two of the inputs are the process variable and setpoint. These are either analog inputs or calculated real variables such as the output of another PID. The setpoint can also be a constant which can be adjusted on line from the PID tuning menu available in OnTOP or the ECLiPS Debug Mode.

The two analog inputs, process variable and setpoint, are treated as 0 to 100% signals inside the algorithm for mathematical purposes. Each input and output has a maximum

and minimum scale parameter associated with it. When the User programs the PID, he sets the minimum engineering unit which will correspond to 0% and the maximum engineering unit that will correspond to 100% for that input. The PID algorithm will internally scale the inputs, generating a percentage of the scaling range before they are used. If the minimum and maximum scale parameters are left blank, the algorithm assumes that input is a percentage already scaled 0 to 100%.

The other three inputs are digital values and can be used to implement complex control strategies. They can be actuated by any active State of a Task. They have been assigned names based upon the PID Loop with which they are associated. These names are Track_Mode, Block_Up and Block_Down. These inputs are controlled with the Set_Bit and Clear_Bit keywords.

If a PID Loop is named Tank_Level then the ECLiPS program statement:

Set_Bit Tank_Level Track_Mode.

puts PID loop Tank_Level into tracking mode.

The Track_Mode input when set, puts the PID loop into a track mode where the output is not changed by the PID, and the algorithm is set up to perform a bumpless transfer when the Track_Mode input goes back to being false. This signal would be 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 integrate down, towards zero, when the error signal, the difference between setpoint and process variable, is negative. If the error signal is positive however, the output will not change and the integral portion will not be allowed to wind up. 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.

Using the Track, Block_Up and Block_Down signals allow the User to build very sophisticated controls. If all that is needed is a simple single loop controller, these inputs can be ignored. A more detailed discussion of these inputs and their use is in the **PID Algorithm Philosophy** Section.

PID Output

The PID outputs consist of a floating point value output that can either be assigned directly to an analog output, or to a floating point variable for use such as the input to another PID or in some calculation. Internal to the PID, the output's range is 0 to 100%. Like the inputs, the output also has a scaling constant that allows the User to set the maximum and minimum scale. This allows the User to make the output be a –100% to +100% controller, or any other engineering units desired. If the maximum and minimum scale constants are left blank, the PID output is 0 to 100%.

The other three outputs are digital variables that indicate the status of the PID algorithm. They are Track_Mode, HL_Status and LL_Status. The two limit outputs are true when either the PID has reached its limit or the Block_Up or Block_Down inputs are true.

The outputs may be used in the ECLiPS program by specifying the PID name followed by the appropriate keyword. For example, the ECLiPS program statement:

If Tank_Level LL_Status is true, go to the Reset State.

makes Reset the active State of this task.

Tuning Constants

The PID algorithm also has several adjustable tuning constants. These include the Gain, Reset and Rate, the high limit and low limit, and a bias value. The bias is a value that is added to the output at all times and can be used to insure a minimum output from the PID.

The high and low limits set maximum and minimum values, within the scale maximum and minimum, that the controller output will not exceed. When these limits are reached, the appropriate status output is set, and anti–reset windup techniques automatically go into affect for that PID.

Every PID also can be selected to be either a direct acting or inverse acting controller. This is a parameter selected at programming time. A direct acting controller will integrate from 0 to 100% if the setpoint is greater than the process variable. A reverse acting controller will integrate from 0 to 100% when the process variable is greater than the setpoint. All other features are exactly the same whether the PID is in the direct or inverse mode.

The following is a summary of User selected values. These values are all entered in the PID Loop Configuration form displayed for initializing a PID loop. PID loops are initialized in program mode using the LIST and DEFINE options from the program mode menu.

Perform Functions

The Perform functions implement operations which are more complicated than the common State Language Terms. ECLiPS presents a form which has a template of information that is entered by the programmer. When this form is completed ECLiPS enters the English text for the proper execution of the function.

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, ie. each time the State is scanned.

Perform functions are Functional Terms and must be the only Terms in the Statement.

Perform Function Forms

ECLiPS presents a form to enter the parameters for the Perform functions. Except for the specialized Perform functions described in the next section, most of the forms are tables providing information about each of the parameters. The column headings of the table follow:

Parameter Name	Type	Use	Required	Actual

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 Functions

The set of table functions (individual descriptions to follow) allow the User to perform several functions using data in a table or array type fashion.

Tables in General

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). 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. There is 20K bytes of memory reserved for use with table functions.

A Table must first be defined before it can be used. This is done using the four Table Define functions. The Swap_Table_Value functions named Swap_Table_Value_Int, Swap_Table_Value_Float, Swap_Table_Value_Dig, and Swap_Table_Str allow the User to either write a value from a variable of the same type into a Table element, or read a value from a Table element into a variable of the same type.

The Init_Table functions named Init_Table_Integer, Init_Table_Float, and Init_Table_Digital allow the user to store multiple values into a Table at one time. There is no initialization function for a string table. The Copy_Table_To_Table function allows the User to copy one Table's value into another Table.

All Table functions check to make certain that when a Table is selected for use, it fits the definition that has been previously creates. 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

The User must define every Table he Uses with this function. When this function is selected from the Perform menu, a menu will display the following for the User to select:

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 (Y or N)

At State Engine run time, a non–critical error with a message will be generated if the Table number specified has already been defined. The Table will not be re–defined and the original definition will be retained. To avoid this error, the User should place this function in a State that only executes once.

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 on the three different types of Tables.

Swap_Table_Value_Int
Swap_Table_Value_Flt
Swap_Table_Value_Dig
Swap_Table_Value_Str

These functions allow the User to write a value from a variable of the same type into a Table element or to read a value from a Table element to a variable. There are four distinct functions of different types so that ECLiPS can check to make sure the variable named is of the same type as the Table specified.

When the User selects one of these functions from the Perform menu the following information will be requested:

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

The State Engine 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.

Initializing Tables

The three Init_Table functions allow the User to set values for multiple elements in a Table at one time. There are three distinct functions of different types so that ECLiPS can check to make sure the variable named is of the same type as the Table specified. There is no initialization function for String Tables.

Init_Table_Int
Init_Table_Flt
Init_Table_Dig

When the User selects one of these functions from the Perform menu the following information will be requested:

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

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 Engine 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.

The State Engine 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.

BCD I/O Representation

General

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 Engine 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 will be 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.

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.

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.

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.

If more than 28 integers are to be connected, two shift registers can be cascaded by having variable_28 of the first shift be the same as variable_1 of the next shift.

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.

Note: only Variable_1 must exist, all others are optional.

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 Engine 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.

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 integer 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 will convert the value stored at the float 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.

C for Concatenate

If the Operation character is a C, the function will concatenate or add the string of characters named in Reference_variable to the main string. The resulting main string can not exceed 80 characters, so the addition of the Reference_variable characters will be truncated at that time.

L for string Length

If the Operation character is an L, the function will calculate the number of characters in the string, and put 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 will result and produce errors if they do not.

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 10 of the hour counters and 30 of each of the other counters active at a time.

The Time Counter form that ECLiPS displays when this Perform Function is selected, has three parameters that define what 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'.
- **'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.

Note

Each of the time counters has a maximum value of 32,767. If you expect the value is going to exceed the maximum, use the next larger counter type. For example, use a minute counter instead of a second counter. After reaching the maximum, the counter still continues to increment, becoming a negative number.

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 have unique ways that the operations are specified.

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.

User Menu

This function will display a menu of up to 10 items and then wait 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.

Keyboard Definitions

Function Key Definitions

Key	Functions			
	<Key>	<Alt + Key>	<Ctrl + Key>	<Shift + Key>
<F1>	Help	Toggle Insert Overtype Mode	Project Error Help	
<F2>	Save File to Disk	Send to Controller	Retrieve from Disk	Check for Errors
<F3>	Menu			
<F4>	Define / View Current Word	Define All 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	Change Frame / Loop in Form		
<F10>	Toggle Program / Debug Modes			

Hot Key Definitions

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, ie. Paste.	Program
<Ctrl + R>	Communication Port Reset in Debugger	Debug
<Ctrl + V>	View English in Debugger	Debug
<Ctrl + E>	Enable Monitor Display in Debugger	Debug
<Ctrl + F>	Force Table	Debug
<Ctrl + T>	Trace Upload / Display	Debug

Miscellaneous Key Definitions

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 a List/ Menu	
<Enter>	Carriage Return / Select Item	
<Esc>	Cancel Operation	

Series 90–70 State Logic Control System

This section describes the Series 90–70 State Logic control system using the State Logic Processor(SLP). Descriptions include the SLP / CPU interface, how to setup your system, serial port setup, and scanning considerations.

SLP/CPU Interface

ECLiPS programs execute in the State Logic Processor (SLP). The SLP is a module with the State Engine operating system imbedded that plugs into one of the slots of a Series 90–70 PLC chassis. The CPU is the direct interface to the I / O and other VME products used in the control system. The SLP accesses the CPU memory to use the system I / O and communicate to other VME modules. Communication between the SLP and CPU is across the Series 90–70 backplane.

This section describes the CPU memory accessed by the SLP, the different communication methods, and how to set up your system.

CPU Memory Accessed by the SLP

ECLiPS programs access discrete and register types of memory stored in the CPU. Discrete types are bit oriented being either ON or OFF (1 or 0). The discrete types accessed by the SLP are %I, %Q, %M, %T, %G, %S, %SA, %SB, %SC. The following table describes each of the discrete memory types.

Memory Reference	Reference Description
%I	Real world discrete inputs. These locations store the status of the inputs following the last scan of the inputs by the CPU of the system.
%Q	Real world discrete outputs. These locations store the status of the outputs as last set by the control programs. The CPU updates the real world outputs during its normal execution cycle.
%M	Internal discrete references. These references are used to store intermediate conditions used in the logic decisions in the program.
%T	Temporary internal references.
%G	Global Data references. This data can be shared between multiple devices using the Genius Communications Module to communicate over a Genius communications bus.
%S, %SA, %SB, %SC	System Status references. References system fault information.

Discrete Memory Types

Register memory types store a value using sixteen bits of memory space. The types of register storage are R–Registers(%R), analog inputs (%AI), and analog outputs (%AQ). These register types are described in the following table.

%R	Stores a value in sixteen bits. State Logic Program Register Variables access these memory locations. A Register Variable can be either an integer or floating point value. Floating point values use 2 %R registers. The write and read terms can also access these locations for character values.
%AI	Real world analog inputs. These locations store the value of the inputs following the last scan of analog values.
%AQ	Real world analog outputs. The CPU transfers these values to the analog outputs during the normal execution cycle.

Register Memory Types

State Logic program references to CPU memory locations are user selected names that are identified by the memory type and a number representing the storage location, ie. Q14.

Input/Output Memory Designation

Every CPU memory location used in the SLP program is specified to be either an input or an output to the State Logic program. Output memory locations can be changed (WRITTEN) or used (READ) by a State Logic program, but inputs can only be read. Therefore, if the State Logic program is going to change a memory location, it must be defined as an output, but if the Ladder Logic program in the CPU is going to change the memory location, it must be defined as an input for the State Logic program. The CPU and SLP cannot both change the same memory locations.

Any communication between a State Logic program and a Ladder Logic program requires that an output to one program must be an input to the other. Therefore %Q's which are outputs to the CPU may be used as inputs to the State Logic program logic and %I's, normally inputs to the CPU may be changed and thus are actually outputs to the State Logic program.

Specific situations where these types of definitions are used are for communications between the two programs, when the SLP is used to provide diagnostics for an existing ladder program, and when the SLP is used to simulate the real world response to a ladder program.

State Logic Processor Memory Capacities

The CPU selection determines the I/O and register ranges that can be accessed by the control program. ECLiPS generates different control programs depending on the type of CPU selected (731/732, 771/772, or 781/782).

The State Logic program can access a set number of points for each of the discrete and register memory types. These points must form a continuous block anywhere within the range provided by the selected CPU. The SLP obviously cannot access any more discrete or register memory locations than those provided by the CPU in the system.

Processor	731/732	771/772	781/782	SLP
%I	512	2048	12288	1024
%Q	512	2048	12288	1024
%AI	8K	8K	8K	256
%AQ	8K	8K	8K	256
%T	256	256	256	256
%M	2048	4096	12288	2048*
%G	1280	1280	1280	1280
%S	128	128	128	128
%SA	128	128	128	128
%SB	128	128	128	128
%SC	128	128	128	128
%R	16K	16K	16K	2048**

90–70 CPU and SLP I/O and Register Capacity

* Programs with I/O transfer type Co_Processor are limited to 2000 %M I/O

** Programs with I/O transfer type Controller are limited to 600 %R registers.

CPU Selection

The CPU selection determines the discrete and register ranges that can be accessed by the control program. ECLiPS generates different control programs depending on the type of CPU selected (731/732, 771/772, or 781/782).

Whenever a new project is created, ECLiPS asks for the CPU type. The CPU type can be changed at any time with the "PLC Type" option from the DEFINE menu. The different programs depending on the selected type are created when the program is translated.

Programs configured for one CPU can be run in a system that has the selected CPU or more advanced CPU. A program configured for 781/782 CPU can run only on that CPU. A program configured as 771/772 can run on that CPU or the 781/782 CPU. Program configured for a 731/732 CPU can run in a system with any CPU.

The State Logic program can access a set number of points for each of the I/O types and registers. These points must form a continuous block anywhere within the range of the selected CPU. To assign a range for the discretes or registers, select the "I/O and Register Range" option from the Define menu. Fill in the form which specifies the base or lowest number for the SLP block.

For example, if the base number selected for %AI is 201, the 256 %AIs that the SLP can access start from 201 and end at 456 in the %AI storage tables of the CPU.

CPU/SLP Interface Methods

Each 90–70 program must be configured for the interface type which specifies how information is transferred between the SLP and the CPU. This selection is chosen from the "CPU Type" option on the Define menu and is also presented whenever a new project is created. The possible choices are Co–Processor, Controller (Small), and Controller (Large).

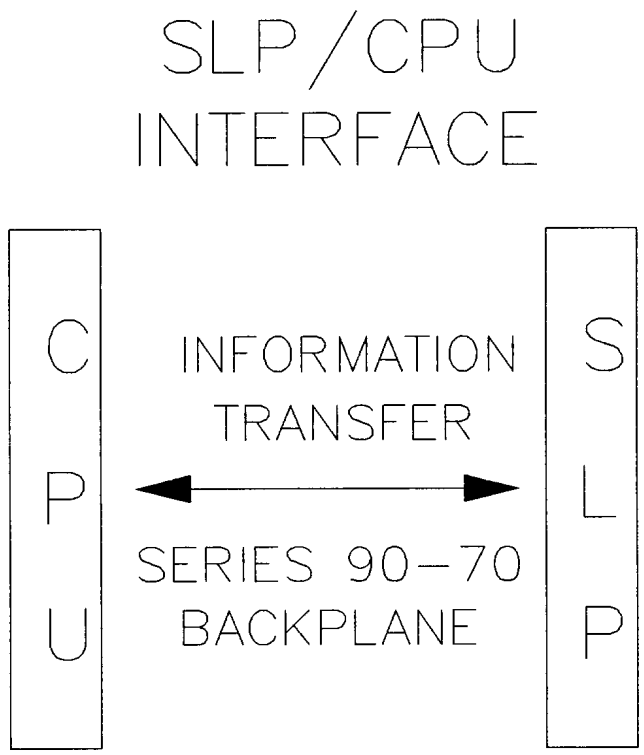


Figure 8-3. SLP/CPU Interfaces

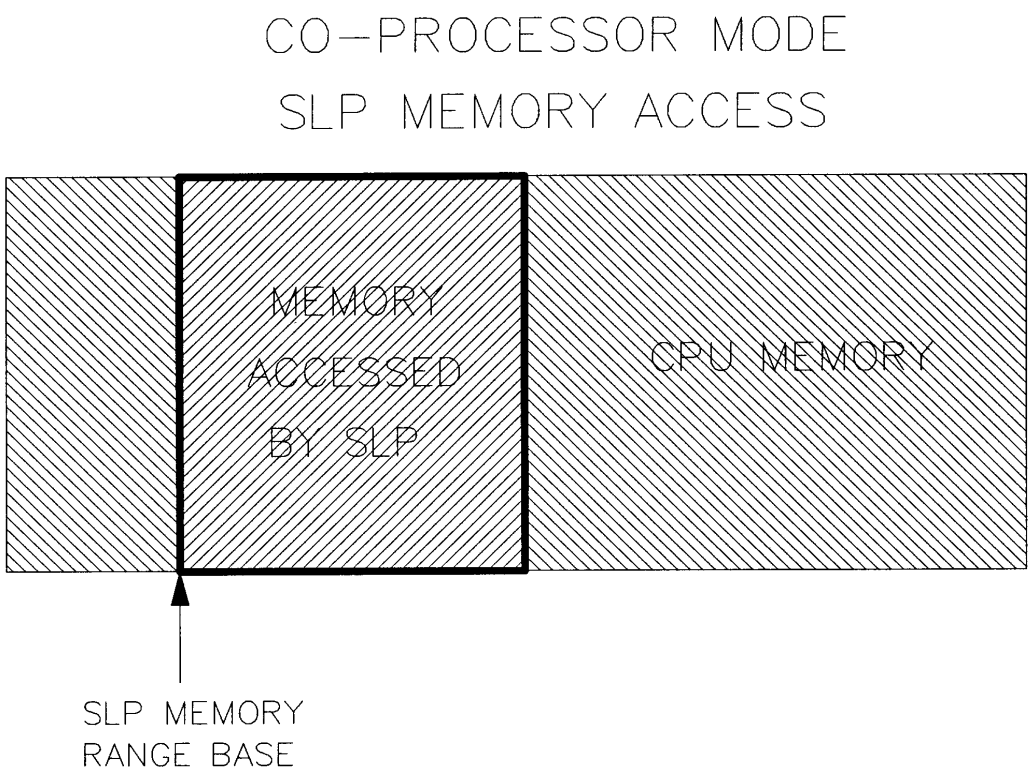
The Co–Processor method of transfer is easy to implement, since the communications between the SLP and the CPU are handled automatically by the SLP, and the input and output designations for each of the memory types can be mixed throughout the range for that type.

The Controller methods are faster, since the communications are handled by Ladder rungs of logic executing in the CPU. The ranges for each of the memory types is split so that the upper range is all outputs and the lower range is all inputs. These inputs and outputs cannot be mixed when the Controller interface methods are used.

Co–Processor Interface Option

The Co–Processor interface method allows the discrete and register values to be set up specifically for the needs of each application. Use the Co–Processor method for either State Logic only control or with hybrid State / Ladder Logic control systems.

a80014



The inputs and outputs can be mixed, over the range of each memory type. If program performance is an issue, it is important to keep the outputs contiguous. If the outputs are split up, there is an adverse effect on the system response time. The translator issues a warning if the outputs are not contiguous.

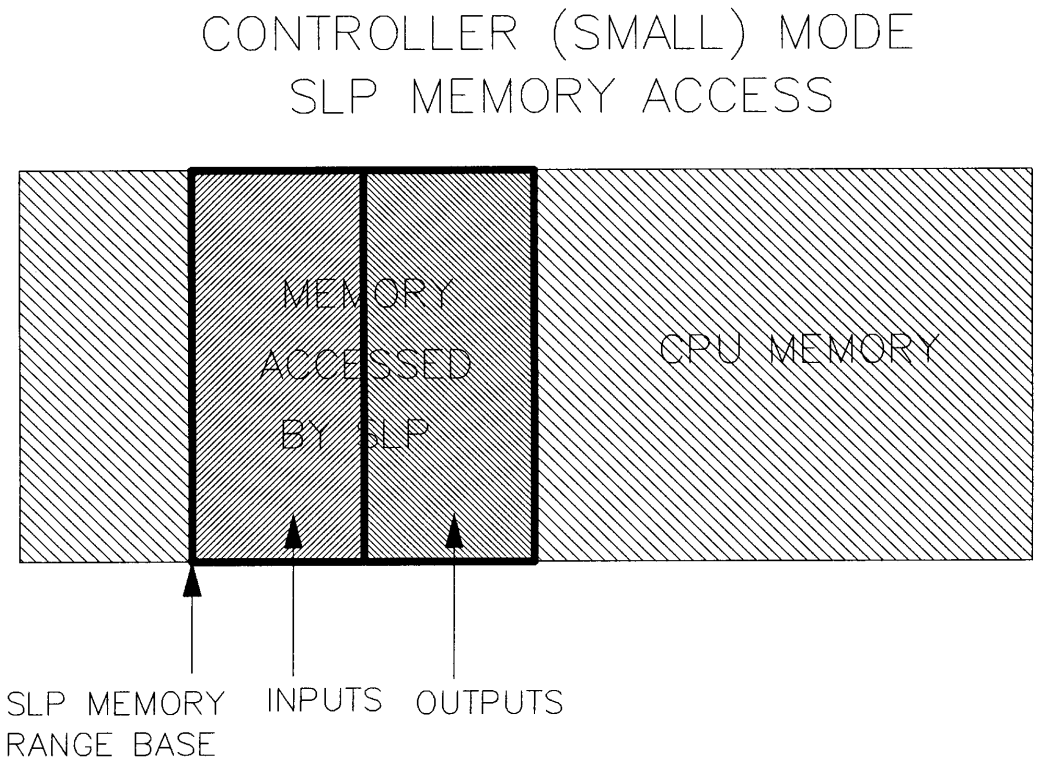
Whether the Co–Processor method or Controller methods are used the CPU must be placed in the run mode and I / O enabled for the State Logic program to access the real worldI / O.

Controller Interface Options

The Controller options provide higher system performance and require rungs of ladder logic in a program running in the CPU to handle the information transfer. The Small Controller option is used for State Logic only control systems. The Large Controller option for a dual processor, hybrid State / Relay Logic system with the CPU executing a control program concurrently with the SLP execution of a State Logic program.

For both of the Controller types, the ranges of each memory type are split into two sections, the upper part is all outputs and the lower part inputs. By default, each range of memory type is split in half. When an I / O or register number is selected, ECLiPS automatically selects the input / output option.

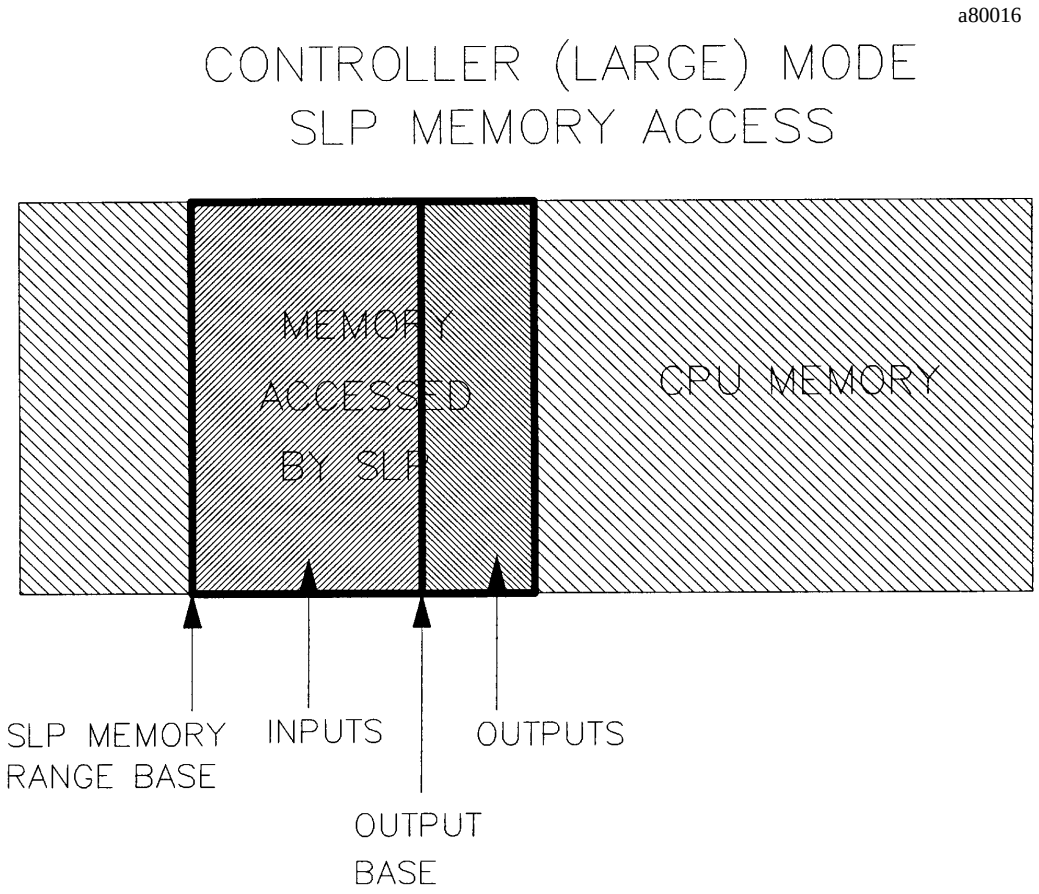
a80015



The exception to this rule is how the actual I / O points are configured when using the Small Controller option. All of the %I's and %AI's are inputs, and the %Q's and %AQ's are outputs. Since the Small Controller type is used only when there is no Ladder Logic used, there is no need for the inputs to the CPU to be anything but inputs to the SLP or outputs to the CPU to be anything but outputs to the CPU.

In the Large Controller option, the split point between inputs and outputs is modifiable, by changing the number where the outputs start. If the Large Controller option is selected, then the DEFINE menu displays the option "PLC I / O Output Base." The starting point of the outputs can be changed anywhere within the range for the memory type. If the output base is set to the start of the range, all locations are outputs. All locations are inputs when the output base is set to the maximum value.

a80016



The controller options require some memory transfer functions in a Relay Ladder Logic program executing in the CPU. The Large Controller option is used if a RLL application is being controlled at the same time the SLP is executing a State Logic program, otherwise the Small version is used.

Disk number 2 of the ECLiPS distribution disks contains several SDE files with ladder programs that manage the transfer of information between the SLP and the CPU. See the section titled “Setup Series 90–70 CPU” for a list of the transfer program files and how to incorporate them into the CPU program.

These transfer programs assume that the SLP is in the slot next to the CPU and that the SLP memory range setup is at default values. If either a different slot is used or different memory ranges selected, these transfer programs must be changed. Call one of the technical support hot lines for help making these changes.

Whether the Co–Processor method or Controller methods are used, the CPU must be placed in the run mode and I/O enabled for the State Logic program to access the real worldI/O.

Third Party VME Interface

To use information provided by third party VME products, transfer the information between the CPU and the VME product with logic in a Ladder Logic program executing in the CPU. Use the VME READs and VME WRITEs to accomplish this transfer to and from CPU memory locations accessed by the SLP. Refer to the Logicmaster 90 User's Manual for information on using these terms in a Ladder Logic program.

Clearing Outputs at Power Up

Much of the discrete memory in the CPU is retentive, meaning that its value is saved when power is lost, because the memory is battery backed. These discrete locations must be cleared when power is first applied to the system, otherwise outputs that were ON when power was lost may go ON when power is back on again..

To accomplish this initialization, a rung of ladder logic in the CPU program is used to clear the SLP outputs. This rung uses the First Scan contact and block clears to initialize the SLP outputs.

Watchdog Timer

The watchdog timer is designed to stop the control system if the SLP fails or stops communicating with the CPU. ECLiPS provides an option to include a watchdog Task into the State Logic program every time a new project is started. The watchdog Task can also be added to the program at any time by importing the watchdog project that is stored in the same directory as ECLiPS.

This Task is designed to work in conjunction with rungs of logic executing in the CPU program. The rungs of ladder logic are included in the file called WATCHDOG.SDE on disk 2 of the ECLiPS distribution disks. See the section on setting up the system CPU, for an explanation of how to incorporate this file with an existing ladder program. The watchdog rungs of ladder logic should not be used without the watchdog Task included in the State Logic program.

If the watchdog timer does shut down the system, the fault, "PLC CPU Software Fault", appears in the fault table.

The default watchdog system provided is a generic system that does add some time to the scan time of the system. To minimize the scan time impact, customize the watchdog Task and ladder logic by changing the Heartbeat and SLP Running outputs from %G memory type to %Q and assigning the %Qs to be contiguous with the %Qs in the system. Remember that this change must be made to the Ladder Logic program in the CPU as well as the State Logic program in the SLP.

How to Setup Your System

This section describes the step–by–step procedure for setting up your Series 90–70 State Logic control system. First the steps are listed, and then each step is described in detail.

1. Identify the CPU to be used in the Series 90–70 PLC.
2. Select the SLP / CPU communications mode.
3. Determine whether the system is State Logic only or a hybrid State / Ladder logic system.
4. Make memory range adjustments if necessary.
5. Make output memory base adjustments if necessary.
6. Transfer Ladder Logic rungs from the ECLiPS distribution disks into the CPU program, using Logicmaster 90.
7. Configure the Series 90–70 Control System.

Series 90–70 State Logic Control System Setup

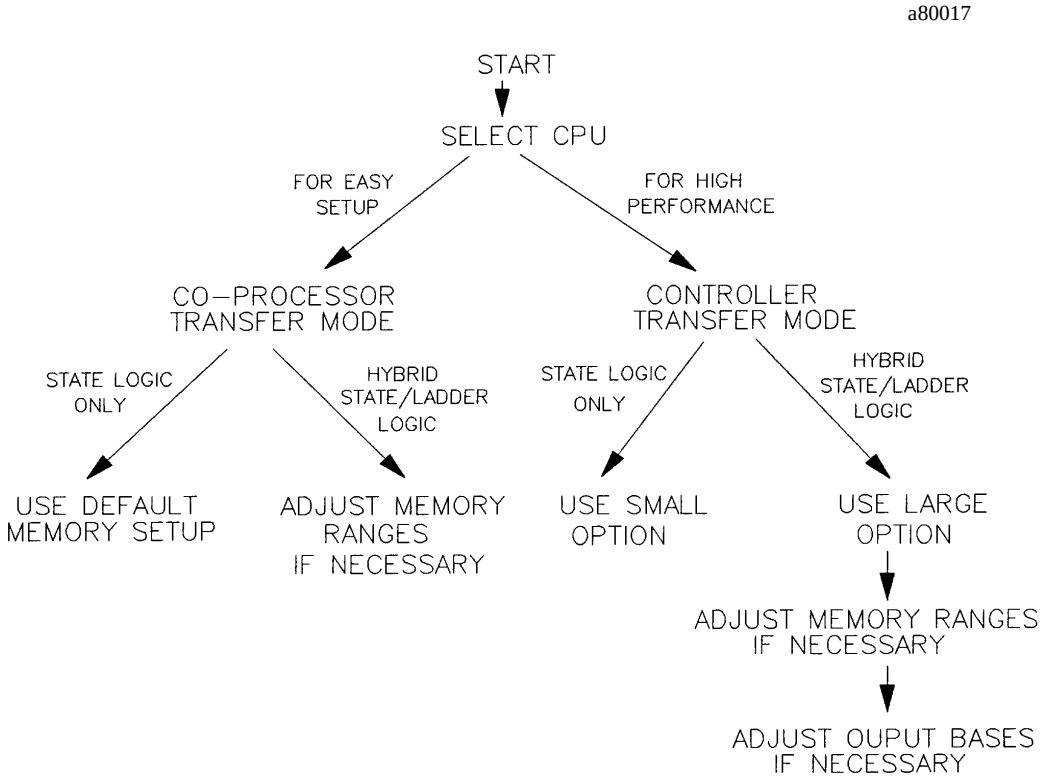


Figure 8-4. Series 90–70 State Logic Setup Decision Tree

Identify the CPU to be Used

Whenever you start a new project, ECLiPS asks you to identify the system CPU. This selection can also be made at any time from the DEFINE menu by selecting the "PLC Type" option.

The CPU selection affects the amount of memory available, see the section on SLP Memory Capacities for more information on CPU memory capacities. The different CPU's also affect the execution speed because of varying clock speeds and micro–processor types.

ECLiPS produces different programs for the different CPU types. The CPU type at translation time determines the type of control program that ECLiPS creates. Programs created for a larger CPU execute properly on a smaller CPU, but the reverse is not true.

Select SLP/CPU Communications Mode

For the easiest setup procedure and input/output assignments use the Co–Processor mode. If a high performance system is required, use one of the Controller modes. If a high performance system requires both State and Ladder Logic control, use the Controller (Large) mode, but for State Logic only high performance systems use the Controller (Small) mode.

The mode selection is made from the same screen that the CPU type is specified. Select the "PLC Type" option from the DEFINE menu.

State Logic Alone or Hybrid State/Ladder Logic System

The State Logic Control System can be a State Logic only system or a multi–processing system with State Logic executing in the SLP and Relay Ladder Logic executing in the CPU. There are also some systems which require more than one SLP.

For most systems that are State Logic only there is no need to change the default memory ranges. In a multiple SLP system, one of the SLP programs probably would access memory outside of the default ranges. State Logic only systems using the Controller interface mode should select the Controller (Small) option.

Systems that are hybrid State/Ladder Logic may need to change the default memory ranges although many can use the default ranges. For hybrid control systems, using the Controller interface mode must use the Controller (Large) option.

Make Memory Range Adjustments

If memory range adjustments are necessary, make these adjustments after selecting the CPU and the interface mode. It is also a good idea to make the range adjustments before assigning any State Logic names to memory locations. If the new range does not include an existing name, ECLiPS aborts the change.

Make changes to memory ranges by selecting the "PLC I/O Ranges" option from the DEFINE menu. To change the range, enter a new base or first location for the block of memory to be accessed by the State Logic program. ECLiPS displays the span of possible selections for the base of each memory type. If the State Logic program accesses all of the available CPU memory for a particular memory type, ECLiPS indicates that the range cannot be adjusted by giving a span of bases of (1..1).

Make Output Base Adjustments

When the Controller interface option is used, the ranges of all memory types are split in half. The highest numbers are all outputs and the lowest are inputs. For the Controller (Large) option, the smallest output reference or base number of the outputs can be adjusted, thus changing the number of inputs and outputs.

To change the output base for a memory type, select the "PLC I/O Output Base" option from the DEFINE menu. The current base and the span of choices are displayed in the form. Enter the new output base for any of the memory types listed.

Setup Series 90–70 CPU

There are two steps to setting up the CPU of the system. First, the CPU must be configured with Logicmaster for what ever hardware modules are installed in the Series 90 chassis. Second, rungs of Ladder Logic are entered into the CPU program area.

There are four reasons to have Ladder Logic running in the CPU:

1. To use the flexibility of two processors and do some of the control operations using the CPU. This hybrid State/Ladder Logic control system is used when some parts of an application are very time critical or when the control is performed by the CPU and the SLP is use for diagnostics or application simulation.
2. To manage the SLP/CPU communications when one of the Controller interface options (Small or Large) is selected for the project. For this option the ladder program uses VME reads and writes to move information between the SLP and CPU.
3. To do a powerup initialization of the CPU %Q outputs. Since the %Q data area is retentive, this rung of logic is necessary to turn OFF %Q outputs that might have been ON before power is lost to the system.
4. To perform the functions of a watchdog timer, which detects whether the SLP is functioning properly. If the SLP stops functioning, the Series 90 system is stopped. These rungs must be used in conjunction with the watchdog Task in the State Logic program.

When the "Make a New Project" option is selected from the Project menu, ECLiPS offers to enter the watchdog Task automatically. This Task may be added to a project at any time, by importing the Watchdog project located in the \ECLIPS\S90–70 directory.

It is strongly advised that this watchdog setup be used, because the system may not behave as expected if the SLP fails or stops communicating with the CPU. The watchdog system assures that the control system is stopped, if the SLP stops executing properly.

Ladder Logic program files for performing the necessary CPU functions have been included with the ECLiPS distribution disks. These files are on ECLiPS distribution disk 2. To select the right file for your application, determine the interface method and CPU for the system and then refer to the following table.

Interface Type	731 CPU	771 CPU	781 CPU
Co–P rocessor	CO–731.SDE	CO–771.SDE	CO–781.SDE
Controller (Small)	SML–731.SDE	SML–771.SDE	SML–781.SDE
Controller (Large)	LRG–731.SDE	LRG–771.SDE	LRG–781.SDE

Ladder Logic Program Files

These files contain Ladder Logic to initialize memory locations used be the SLP. The files for the controller interface methods also contain the necessary logic to perform the memory communications between the SLP and the CPU.

Use the Logicmaster 90 software package that comes with ECLiPS to configure the hardware of the system and to enter the Ladder Logic program file into the CPU. The following steps describe how to use Logicmaster 90 to perform these functions.

1. After starting Logicmaster, select the configuration package, key <F2>.
2. Create a program folder for your application by entering a folder name.
3. Setup the configuration of each chassis slot to match the hardware used in the system, key <F1>. Configure the SLP as a PCM with Configuration Mode as NONE. Press <Esc> when completed.
4. Change CPU memory allocation for %AI and %AQ. First press <F2> then <F4> and change the value for analog memory to 256 for both %AI and %AQ. Press <Esc> twice to return to configuration menu.
5. The configuration is now complete so press <Esc> again to exit the configuration package.
6. Enter the Programmer Package by pressing <F1>. Then select the folder that you just created in the Configuration Package.
7. Select Program Display / Edit <F1>
8. Select Region <F9>
9. Select Include <F4>
10. Now put ECLiPS disk 2 into drive A and enter the Ladder Logic program file name selected from the table above as follows: A:\<filename>. For the Coprocessor mode with the 731 CPU enter A:\CO–731. This operations loads the Ladder Logic program from the file and incorporates it with any existing Ladder program.
11. 11. Press <Enter> and the program file is incorporated into the program.
12. 12. Press <Esc> to return to the Programmer Package starting screen. Then press <F3> for PLC Control and Status.
13. 13. Press the <Tab> key until RUN / OUT EN is displayed in the block and press <Enter>.

The Ladder Logic program for the watchdog timer is called WATCHDOG.SDE and is also provided on disk number 2. Repeat this procedure to incorporate the watchdog file with the other rungs in the CPU. The same watchdog file is used for all CPU and interface types.

The CPU is now set up for State Logic program execution. The chassis slot I / O hardware is configured, the Ladder Logic program is loaded into the CPU, and the CPU is in RUN mode with outputs enabled.

Serial Ports

The State Logic Processor (SLP) has two serial ports. One of these ports may be designated as the Programming Port, the one which connects the SLP to ECLiPS or OnTOP. These serial ports can be RS232, RS–422/485 type ports. See the sections on Write and Read Terms in the Language Description section of this manual for a discussion of how to program communications through the serial ports.

Configuring Serial Ports

The attributes of each serial port may be changed by the State Logic Control program. The parameters are changed by the Set_Commport keyword. There is a discussion of using the Set_Commport keyword in the section that explains the LIST menu options.

Designating the Programming Port

The programming port is the State Processor serial port to which ECLiPS or OnTOP is connected. If ECLiPS or OnTOP is not connected to the programming port, the SLP cannot communicate to these software packages.

Port one is the default programming port. To change the programming port select the "State Engine Configuration" option from the Debug Mode PROJECT menu.

Selecting RS–422/485 Serial Port Option

By default both serial ports are RS–232 ports, communicating at 19200 baud. Either port can be changed to be an RS–422/485 port. See the Series 90–70 State Logic Processor User's Manual for more information the serial ports, especially the signal pin connections for the ports.

To select a serial port to be RS–422/485, select the "State Engine Configuration" option from the Debug Mode PROJECT menu.

CCM2 Protocol Serial Port

CCM2 protocol is a standard open communications protocol defined by GE. The State Engine acts as a slave in a master–to–slave architecture. The remote master computer must poll to retrieve data from the State Engine. All communications are initiated by the host.

The CCM2 protocol defines the message structure, framing, error checking and handling, and timing for all message types. At the lower physical level, the serial port of the State Engine is electrically RS–232 and can be configured for any Baud rate, parity, and stop bits desired via a command in the program executing in the State Engine. Use the Set_Commport keyword in the ECLiPS control program followed by the parameters, to configure the port so that CCM2 communications port electrically matches the serial port of the master computer.

Either of the 2 serial ports can be a CCM2 Communications port. The CCM port is always the port that is not the programming port.

Enabling CCM2 Communication

Both serial ports are normally normal serial ports using no communications protocol. To use the CCM2 communications protocol on the CCM port, the CCM2 Communications must be enabled. To enable CCM2 Communications select the "State Engine Configuration" option from the Debug Mode PROJECT menu.

There are other CCM2 options on this menu. The CCM2 options available are:

Enable CCM Protocol Port
Disable CCM Protocol Port
Set CCM Protocol Station Address

The State Engine using CCM2 only recognizes and responds to messages sent to its address, which may be any number from 1 to 89. The default address will be 1 and the User should assign a Station Address to match the number the Remote Master expects. The Station Address should be validated or assigned before CCM2 protocol communications is enabled.

If the "Disable" function is chosen, all CCM2 communications is immediately discontinued, even if in the middle of a message, and all inputs on that serial port are taken to be ASCII character inputs that will be handled as appropriate to the current State Engine program execution.

CCM Data Types

The CCM2, as well as many other standard protocols, were designed for host computer communication with PLCs. To access State Engine data, the CCM host computer must identify each data element with a CCM number and a CCM type. The types of information accessible from the State Engine plus the respective CCM information are listed in the following table.

Data Type	CCM Type	CCM Number
Task Current State	1	9501 – 9999
Analog I/O	1	0 – 2560
PID Loop Parameters	1	4001 – 5000
Internal Flag	1	5001 – 6000
Integer Variables	1	6001 – 7000
Floating Point Variables	1	7001 – 8000
String Variables	1	8001 – 9000
Character Variables	1	9001 – 9500
Discrete I/O	2	1 – 6912

Table 8-1. State Engine Data Types and CCM References

The State Engine assigns this CCM information to each data element in the program. To access this information use the "Print" option of the Program Mode Project menu. The CCM information about each data element of the State Logic program is displayed by the printout. The following is a sample of the CCM information print out:

Digital Point Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Forward_Limit_Switch	Discrete I/O	2	36
Analog Channel Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Lube_Oil_Pressure	Register	1	65
Oven_Temperature	Register	1	121
String Variable Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Bar_Code_1	Register	1	8001
Integer Variable Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Finished_Parts	Register	1	6001
Part_ID	Register	1	6002
Floating Point Variable Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Oven_Baking_Time	Register	1	7001
Internal Flag Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Part_ID_Read	Register	1	5001
PID Loop Names			
Name	**CCM type**	**CCM type #**	**CCM Number**
Tank_Level	Register	1	4001
Task and State Names			
Task Name	**CCM type**	**CCM type #**	**CCM Number**
Start_Sequence	Register	1	9501
	State Name	**State Number**	
	PowerUp	1	
	Check Conditions	2	
	Start Pump	3	

Part of the CCM protocol is for the message from the Master to define the type, starting register and total number of bytes to be read or written. The Master needs to specify the correct number of bytes depending upon the type of variable. The byte count and other considerations for each of the data types follow:

Task Current State The value stored in the designated register represents the current State of this Task. Current States can be changed and monitored using the CCM protocol. This value uses 1 register or 2 bytes in a CCM message. The inactive State is represented by a value of 0. Warning: If an invalid number is written to put a Task into a State that does not exist, the program continues to execute but that Task does nothing.

Integer Variables This data type uses 1 register or 2 bytes in a CCM message. Any Register Variables that are integers are in this class.

Floating Point Variables These values use 2 registers or 4 bytes of space and are stored in IEEE format. To use these values the host computer must be able to interpret the 4 bytes as an IEEE floating point number. Any Register Variables that are floating point values are in this class.

Digital Inputs/Outputs These points are represented by the right most bit of the transferred byte. A 1 represents ON and a 0 represents OFF. Note that since digital points are OFF by default, an output set ON with CCM will only be ON for one scan and then State Engine operating system turns it OFF.

Internal Flags Each flag is represented by a bit in the data transmitted. The first bit refers to the flag specified and the other bits represent other flags. At least one byte or 8 flags are represented in every transmission. Therefore, when writing to flags, at least 8 flags ar set at a time. State Logic treats flags as discrete l/O in that they are OFF by default. Therefore a flag set through CCM is ON for one scan only. The State Logic operating system turns OFF the flag in the next scan.

Analog Inputs/Outputs Analog I/O are floating point values and therefore require 2 registers or 4 bytes. The host must be able to interpret this value as an IEEE floating point number. Analog channels designated high speed channels are integer information using only 2 of the 4 bytes.

String Variables String variables can be up to 80 bytes or 40 registers long.

Character Variables Characters are 1 byte.

PID Loop Parameters There are 21 parameters associated with each PID loop. The register number displayed in the printout represents the number of the first parameter for the loop. The loop CCM numbers will be in multiples of 21 so that if the first loop is 4001 the second is 4022. The order of the parameters is listed in the following table.

Parameter Byte Count	Parameter Number	Byte Count
Status	1	2
Update Time	2	4
Gain	3	4
Reset	4	4
Rate	5	4
Setpoint	6	4
Setpoint Minimum	7	4
Setpoint Maximum	8	4
Process Variable (PV)	9	4
PV Minimum	10	4
PV Maximum	11	4
Control Variable (CV)	12	4
CV Minimum	13	4
CV Maximum	14	4
Bias	15	4
Lower Limit (LL)	16	4
LL Minimum	17	4
LL Maximum	18	4
High Limit (HL)	19	4
HL Minimum	20	4
HL Maximum	21	4

Table 8-2. PID Parameter Table

The Status parameter is the only one that is not a floating point value represented in IEEE format. This parameter has information about the PID loop operation contained in the lowest 4 bits of the value of this register as follows:

Bit 0 – Block Down
Bit 1 – Block Up
Bit 2 – Track Mode
Bit 3 – Inverse Mode

If the bits are set the corresponding status is true. The other bits are reserved and should not be changed.

If the minimum and maximum values of a parameter equal 0, the parameter is considered to be a percentage in the range of 0.0% to 100.0%. Otherwise, the minimum and maximum values are used with the parameter value to calculate of percentage used in the PID calculations.

Analog Scaling and Update Rates

Analog modules transfer raw numerical integer values to PLCs. Each increment represents a step on the full range for the module. Some normal ranges of raw analog values are 0 to 4095 or –32768 to 32767. These raw values represent some real world values.

For example, a scale might have a range of 0 to 50 pounds actually sending an analog signal of 0 – 5 volts to an analog module. The module converts the analog signal to an integer value in the range of 0 to 4095. If the scale detects 25 pounds, it sends a 2.5 volt signal to the analog module which converts the voltage to a number, 2047.

You can have ECLiPS do the conversions of raw analog values to real world engineering units for you. After each analog channel is defined, ECLiPS asks if you want to scale this channel. If the answer is YES, a form is displayed which accepts four values which specify how the scaling is done.

The D / A and A / D values are the raw values and the engineering units are the converted values. Two scaling points where the conversion of A / D or D / A value to engineering value is known are specified. In the example of the scale the low scaling point values are 0 for the A / D or D / A and 0 for the engineering units. The high point is 4095 for the A / D or D / A and 50 for the engineering units. After these points are specified, the State Engine can convert any other raw value to the correct real world value so that a raw value of 2047 is converted to 25 for use in the State Logic program.

When analog channels are scaled, the State Engine converts the raw analog values to floating point numbers. Since floating point operations use a lot of time, the analog values and their conversion to floating point values are not updated every scan. Conversely, unscaled analog channels are always updated every scan.

The State Engine uses a default scheme of scanning the analog channels at 1 / 10 second intervals. To inspect or change this scheme choose the “Series 90 Scan Rates” from the DEFINE menu.

Use this option to change ECLiPS’ default analog scanning scheme. A form is provided to show how the analog channels are scanned. There are 8 columns and 10 rows. The entry in each one of the 80 locations represents a block of 8 analog channels.

Each 1 / 10 of a second a collection of analog channels are scanned. The entries in each row represent the blocks of channels that are scanned for that 1 / 10 second time interval. There are 8 columns each representing a block of 8 channels, so that up to 84 analog channels can be scanned every 1 / 10 second. If a block is to be scanned every interval or 10 times a second, then that block must appear once in every row of the form.

There are 4 data entry locations at the top of the form. To enter values into the form, fill in these data entry locations for the Time Entry(Row), Block Entry(Column), Type and Number. Item is entered into the form when the Number value is entered.

State Engine Scan Considerations

The State Engine operating system is a scanning system. There are several steps that are executed in sequence to complete a scan. These steps are continuously repeated while the State Engine is in operation. One of the steps executed when the State Engine is running a program is the scan of the program. The first section below discusses all of the steps executed for each scan cycle and the next section discusses the program scan step.

Steps of the Scan Cycle

These are the steps that the State Engine executes each scan cycle:

1. Character Input Service

The State Engine retrieves messages that have completely received from the serial ports and dispatches them. These messages may be inputs to READ terms or ECLiPS or On-TOP commands.

2. Real Time Clock

The State Engine updates the current time registers from the Real Time Clock.

3. Transfer Digital I/O

Inputs are written from the CPU to the State Engine Input Image Table. Outputs written from State Engine Output Image Table to PCIM and then the output image table is cleared.

4. Transfer Analog I/O:

Analog output values are sent to the CPU and input values are read from the CPU when the update time for a block of analog channels has elapsed. High speed analog channels are updated every scan.

5. Program is Scanned

The active State of every Task is scanned. The State Engine looks to the Input Image Table for tests of digital inputs. Digital outputs that are to be ON are set in the output image table, outputs test ON only when verification that the output is ON is received back from the block. Variable values are updated immediately and flags set ON are not tested as true until the next scan.

6. Service CCM Messages

If there is a completed message received from the CCM serial port, that command is executed.

7. PID Update

PID Loops are updated when the update time period for the loop is completed.

Program Scan

The program scan 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 tens and maybe 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 (.).

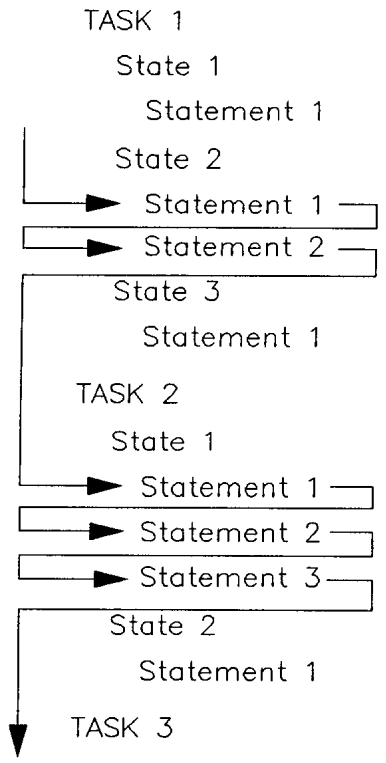


Figure 8-5. Program Scan

The actions specified by Functional Terms are executed when the 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.

a80003

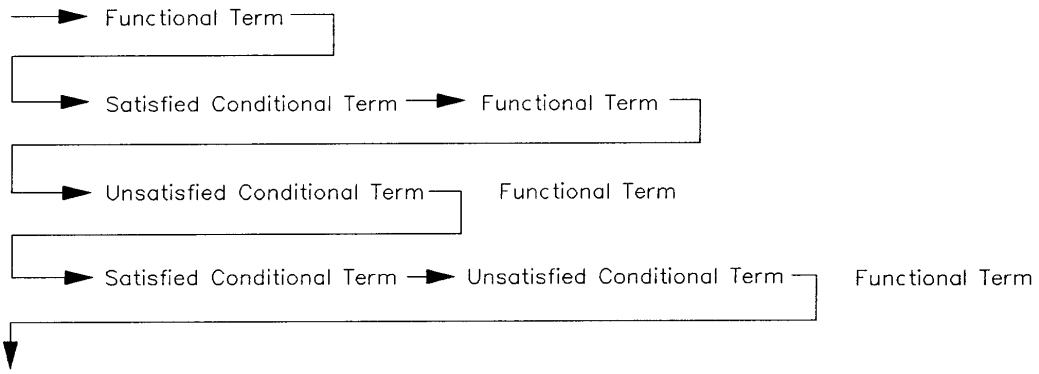


Figure 8-6. Statement Scan

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, none of the Functional Terms are executed.

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.

For example:

If Start_Pushbutton is pushed, go to Start_Up State.
G0 to Restart State.

causes the Start_Up State to become the active State when the process input represented by Start_Pushbutton is true. All Terms or Statements following this Statement are not executed and at the next controller cycle, the scan of this Task starts at the first Statement of the Start_Up State.

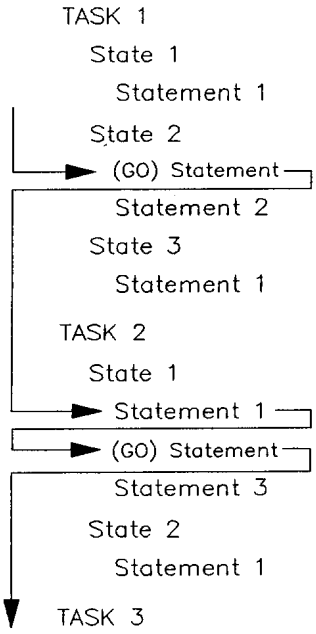


Figure 8-7. Program Scan with GO Terms

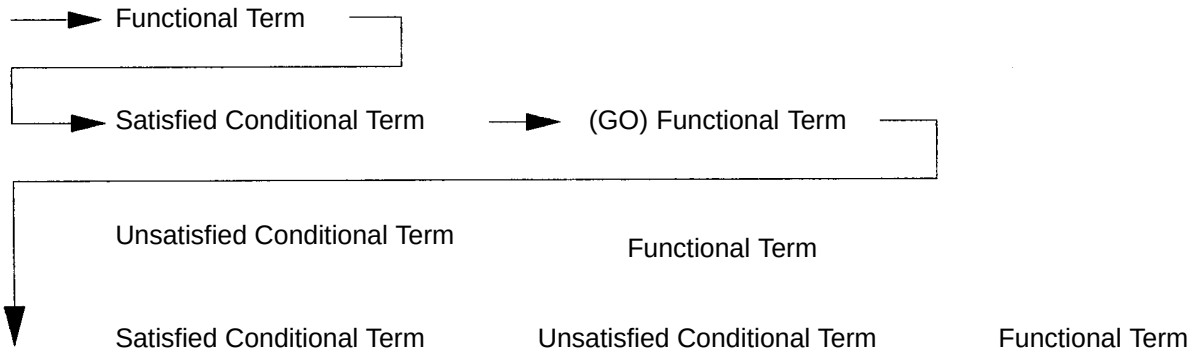


Figure 8-8. Statement Scan with GO Terms

During the program scan any changes to variable and analog values 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 conditions 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.

Other State Engine Setup Options

Run–time Error Setup

The are several errors which may occur during execution of the State Logic program such as divide by zero or integer overflow or perform function errors. Run–time errors are divided into critical and non–critical categories.

The default response of the State Engine is to halt the program when a critical error occurs and to continue program execution after a non–critical error. These responses may be changed through options on the Debug Mode PROJECT menu.

Automatically Start Program Execution

By default the State Engine powers up in the Halted mode. The State Engine can be set-up to automatically start running the program when power comes on.

To set up the State Engine to automatically execute the program on power up, select the "State Engine Configuration" option from the Debug Mode PROJECT menu. When set to automatically run the program, the previous settings of other State Engine setup options are also invoked on power up, including programming port, CCM Enables, CCM Station Number, Error Response Setup, serial port RS–422/485 status, and Simulation Mode Status.

Making a Permanent Copy of the Terminal Log

The Debug Mode main screen is called the Terminal Log. This log displays messages sent from the running control program, plus run–time error messages, and records of the use of the FORCE, DISPLAY, and CHANGE functions. A time stamp accompanies all of these messages except for the information received from the running program.

The Terminal Log stores these messages so that even those messages that have scrolled off the screen can be reviewed. To see information scrolled off the screen, press the up arrow key until to scroll the information back down into the screen. There is a limited amount of space available, so that when there is no more storage space, the oldest information is overwritten. When the Terminal Log storage space is full and the up arrow is continuously pressed, the information displayed eventually goes from the oldest back again to the newest information.

To make a permanent record of the Terminal Log, select one of the Log options from the Debug Mode PROJECT menu. Using these options all of the information displayed in the Terminal Log screen can be sent out the parallel printer port or entered into a disk file. When either log is activated, an appropriate indication is displayed in the top bar of the screen.

Simulation Mode

The State Engine can be put in Simulation Mode through the option on the Debug Mode PROJECT menu. When this mode is active the State Engine is completely disconnected from the real world I/O; the State Engine does not change any outputs and does not look at any of the inputs.

This mode is useful to debug a new program before connecting the I/O or for trouble-shooting a machine problem without interacting with I/O points. The State Engine set to this mode executes the program.

Make the program execute as desired by using the CHANGE option to change current States of the Tasks, variable values, and analog values. Use the FORCE option to change I/O or flag status. Changes can also be made by using the VIEW option to display the English program. Put the cursor on a name to display the current value then a new value may be entered.

Setting the System Clock

The CPU has a clock that maintains the current month, day, day of the week, hour, minute, and second. These values are always available through the System Variables Month, Day, Day_of_Week, Hour, Minute, and Second.

The clock cannot be changed from the State Logic Processor. The time must be changed in the CPU using Logicmaster 90. After the time has been changed in the CPU, and cycle power to the system. The CPU time is transferred to the SLP on power up and at midnight.

ECLiPS Menu System

ECLiPS is a menu driver program with the Main Menu displayed when the program is started. The main menu offers five options:

Program Mode
Debug Mode
Online Modifying
Setup
Quit

Program Mode

The ECLiPS Program Mode is a highly specialized editor and compiler designed to create State Logic control programs for one of the State Engine controllers. This section explains all of the menu selections available in the ECLiPS Program Mode.

The ECLiPS program includes the English text, the name definitions and system setup information. The English text is edited by the wordprocessing functions and other special functions are available to help enter some of the program text.

Another class functions define names used in the program to identify I/O points, variables, Tasks, States, and elements of the control program. Still more functions are used to specify system parameters such as security, scan control, and system setup options.

These functions are available through the Program Mode menu accessed by pressing <F3>. The following sections proceed through all of the menu options available in the Program Mode, describing how to use each one.

Add

All of the ADD options are used to add program lines to the English text through the use of fill–in–the–blanks forms. The ADD options always enter text into the program. This text is no different from text entered from the keyboard and may be edited as other text is edited.

Add a New State

This option prompts the user to enter a name for the new State followed by an optional Max Time diagnostic value. The Max Time diagnostic displays a run–time message if the State is active longer than the specified time. If no the diagnostic is required for this State, just hit <Enter> with no value specified in the Max Time entry window. The new State is inserted directly after the State where the cursor is located in the English text.

Add a New Task

This option adds a new Task to the current program. ECLiPS asks if this Task should start in the State that was active when the controller was stopped. A prompt is displayed for the Max Time value for the PowerUp State of this Task. The new Task name plus the PowerUp State name is inserted directly at the end of the Task where the cursor is located.

Add a State Time Out Diagnostic

This option adds the program text for a State Time Out Diagnostic. This option enters a Statement at the cursor location that transitions to another State in the event the State has been active for the specified amount of time. This time must be in the range of .01 to 600.00 seconds.

Add an Analog Range Checking Diagnostic

This procedure works on any analog channel. Select an analog channel from the list and enter the high and low limits along with the State to branch to should the limit be exceeded. When the form is completed, the ECLiPS enters the diagnostic instructions into the program at the current cursor position. The English program instructions can be edited just as any other program Statements are edited.

Add a Variable Range Checking Diagnostic

The variable range checking diagnostic is identical to the analog range checking diagnostic except it checks variable values instead of analog values.

Add a “Perform” Function

The Perform functions implement special purpose operations. ECLiPS automatically enters the English instructions for these functions from information entered in the provided form. Perform functions are Functional Terms and must be the only Terms in the Statement. There may be an unlimited number of Perform functions in a State.

For a more detailed description of the perform functions see the Language Description part of the Reference Section of this manual.

List

The List option is used to view and change lists of user defined names for I/O points, variables, PID Loops, Communication Ports, and Internal Flags. The predefined lists of Keywords, Filler Words, Math Functions, PID Parameters and System Time Variables are also available from the List option. All of the lists may be modified except the Math Functions, Time Variables, and PID Parameters. The List option is also useful for selecting a name and entering it into the program at the current cursor location.

When the list option is selected, the LIST menu is displayed showing all of the categories of names. After selecting a name category, the bottom bar displays the keys available to perform different functions with the LIST (<Ins> Add to List, <Del> Delete from List, <Enter> Write the Name into the Program, or <–>> Edit the Name and/or Configuration Data).

a80025

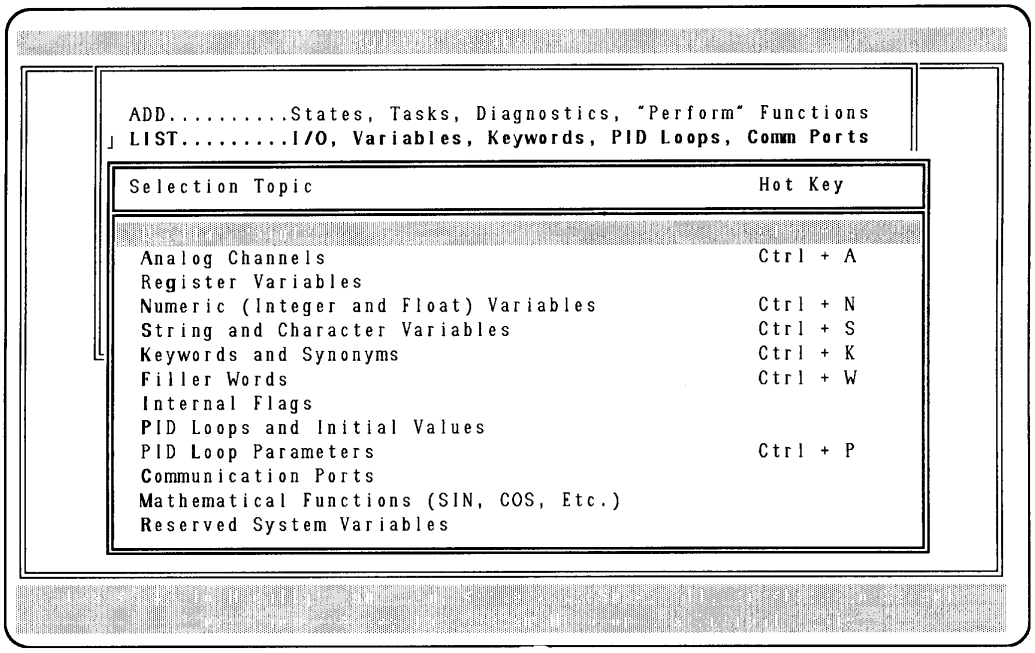


Figure 8-9. List Menu

Digital Points

This option lists in alphabetical order, all of the discrete points defined for the program. The list includes the name, the type (%I, %Q, %M, %G, %T, %S, %SA, %SB, %SC) and the number. This list may be modified by adding, deleting, or editing existing names.

Analog Channels

All of the analog channel names are listed along with the analog type (%AI, %AQ,) and the number that identifies each channel. This list may be modified by adding, deleting, or editing existing names. The block type and configuration for each channel may also

be edited here. When an analog channel is defined or changed, that channel can be scaled, changing the raw values to engineering unit values.

Register Variables

All of the Register variables are listed along with the %R register numbers and whether the type is integer or floating point. Integer values use 1 register and floating point values use 2 registers. All floating point numbers are represented in 32 bit IEEE format.

Numeric (Integer and Float) Variables

Variable names followed by the type are listed in alphabetical order. Integer variables store values in the range of –32,768 to +32,767. The range for floating point variables is +/– 1.175494E–38 to +/– 3.402823E+38.

String and Character Variables

String variables may be up to 100 characters in length and character variables store a single character. The names are listed in alphabetical order along with the type. See the section on the String Manipulation Perform Function for operations that can be performed on string variables.

Keywords and Synonyms

All of the words defined as keywords and their synonyms are listed with the default keyword first followed by all of the synonyms of that keyword. From this display use the following keys:

<Ins> – adds another synonym to the selected keyword
<Del> – deletes the selected keyword
<Enter> – writes the name into the program at the cursor location
<–>> – edits the word
<<–> – Makes the selected synonym the new default keyword

Filler Words

A Filler word is one which is ignored by the ECLiPS translator. Filler words clarify the meaning of the control program. This list is managed by using the same keys that are used with the Keyword list.

Internal Flags

The names of all of the flags are listed in alphabetical order. Flags are most often used to communicate between Tasks.

PID Loops and Initial Values

The names of all of the PID loops are listed in alphabetical order. Press <Enter> key to enter the PID name into the program. Other options also enter new PID loops or delete the selected loop.

The most common use of this option is to enter the initial values of the PID parameters. When the right arrow key is pressed a form is provided to enter these values. See the **Language Description** section for a discussion of using PID loops.

PID Loop Parameters

All of the PID loop parameters are listed in alphabetical order. To use these parameters in a program, the parameter name follows the PID loop name. The programming line:

Make Tank_Temperature Gain equal to Second_Stage_Gain.

assigns a new value to the gain parameter of the Tank_Temperature PID Loop.

Use this list to view the names of the PID Parameters or to enter the selected parameter into the program ar the cursor location.

Communication Ports

Use this option to change communication port names and parameters, to enter a Communications Port Name into the program at the current cursor location, or to enter a Statement into the program to change the serial port parameter settings.

The program Statement that changes the serial port parameters uses the Set_Commport keyword. In order for the communications port parameters to be changed, the Statement using the Set_Commport keyword must be executed by the program.

To enter the Statement to change the serial port parameters press the right arrow key to edit the parameters in a form. When finished press <Enter>, then select the "Insert Reconfiguration Data for Port" option. ECLiPS now enters the instructions into the program. Remember that just changing the parameter form does not change the port setup, but the Set_Commport Statement must be executed in the program.

See the Control System Configuration part of the Reference Section for more information about the serial ports.

Mathematical Functions

The advanced mathematical functions are listed. Press <Enter> to write the function name into the program at the cursor location. See the section on mathematical calculations in the Reference section of this manual.

Reserved System Variable Names

The time and date variables are listed. Use these time variables to access the system clock. The clock can only be read using these variables in the program. To change the clock settings, use Logicmaster 90 to change the clock in the CPU, then cycle power in the system.

Define

These options are used the meaning of words used in the ECLiPS program and to specify system configurations. Each of the options is described below.

The Current Word in the Text

This option displays the definition of the word at the cursor location. Use this choice is useful when you are not sure what of a word's definition. If the word is not defined, ECLiPS presents a list of possible definitions.

Undefined Words Throughout the Text

This option searches the entire program for undefined words. When one is found, the user is given a list of possible types for the word. When a definition has been made, the next undefined word is located and the process continues until the end of the text is reached.

PLC I/O, Analog, Digital, Register

Use this option to define data elements, I/O points or Register Variables, that are stored in the CPU memory. Select the type to define and fill in the displayed form.

Series 90 Scan Rates

This option allows the programmer to customize the scanning of his analog channels. When analog channels are scaled, ECLiPS converts the raw analog values to floating point numbers. Since floating point operations use a lot of time, the analog values and their conversion to floating point values are not updated every scan. Unscaled analog channels are updated ever scan.

Use this option to change ECLiPS' default analog scanning scheme. A form is provided to show how the analog channels are scanned. There are 8 columns and 10 rows. The entry in each one of the 80 locations represents a block of 8 analog channels.

Each 1/10 of a second a collection of analog channels are scanned. The entries in each row represent the blocks of channels that are scanned for that 1/10 second time interval. There are 8 columns each representing a block of 8 channels, so that up to 84 analog channels can be scanned every 1/10 second. If a block is to be scanned every interval or 10 times a second, then that block must appear once in every row of the form.

There are 4 data entry locations at the top of the form. To enter values into the form, fill in these data entry locations for the Time Entry(Row), Block Entry(Column), Type and Number. Item is entered into the form when the Number value is entered.

PID Loop Initial Values

The data entered in this form are the initial values given to the PID Loop when it is started under program control. See the section on PID Loops for more information on these values.

Communication Port Configuration

The values entered in this form may be used to enter a communications port name or configure a communications port. See the Communications Port part of the System Configuration Section and the LIST option for more information about serial port configuration and use.

PLC Type

Use this option to specify the CPU model number and the type of SLP to CPU interface used for this project. See the SLP to CPU Interface section of this manual for more information about these selections.

PLC I/O Ranges

The SLP uses a part of the available register and I/O memory locations available in the CPU. The part used by the SLP must form a contiguous block within the limits of the selected CPU and interface type.

Use this option to identify the lowest number of the block accessed by the SLP for each memory type. A form is displayed to identify the base or starting number of the block to be used by the SLP program.

PLC Output Base

This option appears on the menu only when the interface type is chosen to be Controller (Large). When either of the Controller types is chosen, each of the memory types is split in halves so that one half are all inputs and the other half all outputs. These designations cannot be changed when one of the data elements is defined.

This option though does allows the base or starting number for the outputs to be adjusted so that the numbers of inputs and outputs can be changed. All of the upper part are outputs and the lower part inputs. By adjusting the location of the base of the outputs, the size of each part is changed.

Find

These functions move the cursor around in the English text quickly and allow single and global replacements of text.

Find a Text String

This function allows a user to enter any string of characters that may be present in the English text. ECLiPS then begins searching forward through the English text from the current cursor location until the first occurrence is found. The cursor is then placed at that position. If it is desired to search the entire file, the user may first press <Ctrl + Home> to go to the start of the program before using the FIND function.

Replace a Text String with Another

This function allows a user to replace a single occurrence of a particular string with another. The string to be replaced is entered first, then the replacement string is entered. A confirmation box is displayed before the replacement takes place.

Replace All Occurrences of a Text String

This function replaces all occurrences of the entered string with the replacement string without prompting the user for confirmation.

Perform the Next Find or Replace

This function uses the previously entered find and replacement strings. If no previous find or replace has been initiated, this function executes as the Replace function described above.

Go to the Beginning of Another Task

This function allows the user to enter the name of any Task. The cursor is then placed at the start of that Task.

Go to the Last Project Error

This function places the cursor on the line where the last project error was found during the last Send or Check for Errors. After being placed at the occurrence of the error, the user may move freely about the file. The error message is also displayed once again on the bottom line of the screen.

Project

This menu contains project management functions needed to manipulate the files that make up a project. There are functions that act upon the entire project such as retrieve and save as well as functions to utilize portions of a project such as import.

Retrieve a Project from Disk

This function provides the user with a list of projects in the current working directory from which to choose. After the selection, the cursor is placed in the editor at the beginning of the program.

Save Current Project to Disk

This function saves the project in the current working directory.

Copy the Current Project to a New One

This function creates a copy of the current program including all name definitions and configurations. You give the new project a name and then there are two identical projects with different names.

Note

To copy the current project be sure to use the copy option. changing the path and saving the project, copies only the current Task Group.

New Path/Drive for Projects

This function changes the current drive and or directory where. ECLiPS uses this drive and directory when saving and retrieving programs. Any valid drive and path may be entered.

ECLiPS creates several files when it processes the ECLiPS program. The filenames of the created files use the project name with several different extensions. These files are stored in the current directory. To move a project to another directory or another computer simply copy all of these files to the new location using DOS copying facilities.

Make a New Project

This function starts a new program. If there are unsaved changes from the current program, a prompt to save the program is displayed.

Note

To copy the current project be sure to use the copy option. changing the path and saving the project, copies only the current Task Group.

Import Data from Another Project

This function adds data from another project into the current one. Any of the following categories of data may be imported.

- All Project Sections – The entire project
- Comm Port Configuration – This section contains Communication Port Configuration data
- English Text – The program statements
- PID Loop Configuration – This section contains the PID loop initial values and names.
- Variable Names – This includes all variable names.

Print Project Data

This option allows the user to print any of the sets of information listed below.

- English text
- I/O Map
- Data List
- Task / State List
- Cross Reference List
- CCM Protocol Listing

See the **Creating Program Documentation** section of this manual for more detailed information on these categories.

The Header and Footer entered on this form will be stored on disk so they can be used for all print jobs. The "Output to : " option directs the print data to the printer if the word printer is entered or to a file. To direct the output to a file just enter the file name for this option.

Translate and Download Project to the Controller

Use this option to load the program into the State Engine through the programming port. The program is saved to disk, checked for errors, translated and transmitted to the State Engine through the serial port.

To translate the program, ECLiPS creates a control program for the State Engine from the English program. This translated program is saved in a file in the current directory and has the project name with a .PSM extension. This option always causes the program to be translated. If there have been no changes to the program since the last translation, use the Download only option to eliminate the translation step.

If the program has been changed since the last time it was saved, ECLiPS asks whether to check for undefined words. Answer YES to have ECLiPS present all undefined words one at a time for definition. Answer NO to skip the undefined word search for a faster download.

When a program is downloaded, it is automatically saved to disk and a backup of the previous file with a .BK0 extension is created.

The download function always checks the program for errors. If an error is found, the cursor is placed at the line where the error occurred, and a message is provided with an explanation of the error.

The program is sent out the current serial port. If ECLiPS cannot communicate to a State Engine through the current serial port, an information panel is displayed presenting several options.

Select the "Try to Connect Again" option to attempt to connect to a State Engine again. Use this option to make another attempt to communicate with the State Engine, usually after checking cable connections or powering up the State Engine.

Use the "Change Host Comm Port Settings" option, to change the current port to COM1 or COM2 or to change the baud rate setting. After the change is saved, ECLiPS trys to complete the download operation.

Select the "Abandon Operation" option to stop the download operation and return to the editor.

Error Check the Current Project

This function checks the project for errors, issuing a statistics report if there are no errors. If an error occurs, the cursor is placed at the line of the error and an error message is displayed. If no errors are found, a statistics report including data such as project size and number of Tasks and States will be displayed on the screen.

The Error Check causes the project to be saved and asks whether to check for undefined words. Also a translation is made and saved to disk. All the operations of a download are made except actually sending the translated program to the controller.

Translate Project

The translation is the changing of the English program to a format executable by the State Engine. This translated program is saved in a file in the current directory and has the project name with a .PSM extension. This options does all of the functions of the Error Check but does not ask whether to check for undefined words.

Download Project to the Controller

This option sends the current project to the State Engine without any new translations. Use this option when there have been no changes to the program since the last translation. Eliminating the translation step saves time for the download. If there have been changes to the program since the last translation, an attempt to connect to the State Engine after the download will result in an error message from the State Engine saying that the program on disk is more recent than the one in the controller. At this point the program must be translated first then downloaded again.

Delete the Project from Disk

When this option is selected a list of all projects in the current directory except the current project are listed. Select the project to delete from the list and press <Enter>.

Make a New Task Group

Use this option to create a new Task Group. Initially the project is created all in one Task Group but for organizational reasons the project can be split into up to ten groups of Tasks.

Another reason to use more than one Task Group is that each Task Group is limited to 64K characters including all spaces and comments. Therefore, if the English program is to take up more than 64K Bytes of memory space, more than one Task Group is required.

Enter the name of the new Task Group and first Task of the group when prompted to do so. When a project exists in more than one Task Group another option is displayed on the Program Mode menu, “Task Group”, and this option no longer appears on the Project Menu.

Text

These are typical block manipulation functions found in most text editors. When this option is selected, the indicator word BLOCK is displayed in the top bar of the screen. ECLiPS is now ready for you to select a block of text for manipulation. Use the cursor keys to highlight the block of text to be manipulated. The <Enter> key is used to complete the operation. The operation may also be terminated at any time by pressing <Esc>.

Copy the Selected Block of Text

This function copies a selected block of text from one location to another point in the text.

Move the Selected Block of Text

This function moves a selected block of text from one location to another point in the text.

Remove the Selected Block of Text

This function removes a selected block of text permanently. It is possible, however to undo the last removal by pressing <Ctrl + U>.

Perform Multiple Copies of the Selected Block

This function copies a selected block of text from one location any number of other points in the text. Move the cursor to each location for another copy and press <Enter>. This operation is terminated by pressing <Esc>.

Task Group

This menu option is displayed only when the project is split into more than one Task Group. The options offered on the Task Group Menu provide tools for managing the various Task Groups.

Make a New Task Group

This option causes a new Task Group to be created. Enter the name of the new Task Group and the first Task name when instructed to do so.

Remove a Task Group

Use this option to delete a Task Group. Select the Task Group to delete from the list of Task Groups displayed. When there is only one Task Group remaining, the Program Mode menu no longer offers the Task Group option and the "Break Up the Project into Separate Task Groups" option is again displayed on the Project menu. The program text of the Task Group is erased but all of the definitions remain.

Give the Current Task Group a New Name

This option is used to change Task Group names. The name can be up to 20 characters long.

Change the Order of the Task Groups

Normally the order of Tasks does not have any affect on the execution of the program. Digital outputs are not set until the end of the scan but variable values are changed immediately, and changes to another Task's current State are made immediately so that the order that the Tasks are executed could make a difference to program results.

Use this option to change the order of execution of the Task Groups. Select the Task Group name from the list, and then enter the number to indicate the Task Group's new position in the list.

Join Two Task Groups Together

Use this option to bring separate Task Groups together to form a single Task Group. When there is only one Task Group remaining, the Program Mode menu no longer offers the Task Group option and the "Break Up the Project into Separate Task Groups" option is again displayed on the Project menu.

Select the name of the Task Group from the list. The selected Task Group is then appended to the end of the current Task Group.

Switch to Another Task Group

Use the option to change to another Task Group. Select the name of the Task Group from the list displayed.

Security

This function allows the user to fill in a security table containing passwords for ECLiPS and the 4 access levels of OnTOP.

The password for ECLiPS covers access to the entire program, while each of the levels for OnTOP has a separate password. Level 4 provides unlimited access to all OnTOP functions. For each of the other levels of OnTOP security, select the functions they may be accessed. The functions that may be selected are Force, Run, Halt, Change, Tune PID Loops and Modify Monitor Tables.

Note: Each level of OnTOP security, must have a specified password. If ANY of the levels does not have a password entered, then someone may gain access to OnTOP by just pressing <ENTER> when asked for a password.

Quit

Leave Program Mode and return to the Main Menu.

Debug Mode

In debug mode ECLiPS communicates to the State Engine through the serial ports of the host computer. All information sent to the controller is controlled and formatted by ECLiPS and all data coming in from the controller is analyzed and manipulated prior to being displayed on the screen.

To enter the Debug Mode, the .PRJ and .TG0 files of the project loaded in the State Engine must be in the current directory of the ECLiPS host computer. If these files have a different date / time stamp from the program in the State Engine, ECLiPS issues an error message and denies access to the Debug Mode.

Before starting the Debug Mode, ECLiPS compares the date and time of the last changes the program with the date and time of the program in the controller. If the program in the controller is not the same version as the one in ECLiPS, an error message is displayed. The version of the program used by ECLiPS must match the version of the one in the controller.

There is a menu system for interacting with the State Engine with the program running or halted. The following sections describe each of the menu options.

Project

These functions are used to manage the State Logic Processor, manage program execution (start or stop or simulation mode), manage serial ports and logs, change SLP configuration, and download a new program.

Run Program in the Controller

This command puts the controller into Run mode.

Halt Program in the Controller

This command halts the program in the controller immediately without any prompt.

Communication Port Reset

The communication port may become disabled by improper data transfer or noise on the serial cable so this function is available to reset the port and restore communications with the controller.

Start Printer Output

This function allows the user to send all of the data, that is displayed in the terminal log, out to a printer attached to the parallel port (LPT1). When the printer log is active, the word **PRINT** appears in the top bar of the screen. If there is a printer error, the printer log is no longer active and an error message is displayed. To stop the printer log select that option from the menu.

Activate Log File Output

This function allows the data sent to the terminal log to also be sent to a disk file with a user given name of up to 8 characters. The log file name is given an extension of .LOG. The log file is limited in size to 100K bytes. When the file size reaches 100k the logging of data is terminated.

State Engine Configuration

This option is used to setup the State Engine. Several setup options are displayed on the following form. This form shows the current condition and provides a space to make changes.

Toggle Simulation Mode

When the State Engine is in Simulation Mode, it does not communicate with the I/O. The State Engine does execute its program, but there is no transfer of inputs or outputs. To test a program in Simulation Mode, digital I/O and flags may be forced and analog variable values may be changed. The current State of a Task may also be changed.

To change to normal mode with the I/O System enabled, choose this option again. This option may not be chosen when the State Engine is executing a program.

Enable/Disable CCM Port

The CCM port is the one not designated to be the programming port. When enabled, the CCM port responds to commands using the CCM protocol. The default state of the CCM port is disabled. See the Series 90–70 State Logic Control System section for more information of the serial ports.

CCM Port Number

Selects which serial port is the CCM port. This port cannot also be the programming port.

Set CCM Station Address

Setting the CCM station number for the State Engine, gives the State Engine a number in the range 1 – 89. The State Engine then only responds to commands using the specified station address.

Enable/Disable Automatic Program Execution on Power Up

The default SLP response to power up is to be in halt mode. If automatic execution is enabled, the program starts running automatically when power is applied to the system. The SLP configurations active when power was lost are still active when automatic execution is enabled. These options include serial port setups, error handling configuration, CCM setup, and programming port designation.

Error Handling Setup

There are two classes of run–time errors generated by the State Engine; critical and non–critical. This option allows the State Engine to be configured to either halt the program or continue running when an error in either one of these two classes occurs.

When this option is chosen, the first the response to critical and next to non–critical errors is chosen. This setup may be changed at any time.

Change Programming Port

This option allows serial port chosen to be the programming port to be changed. The programming port is the one that connects the State Engine to ECLiPS. The default programming port is port 1. This port cannot be the same port as the CCM port.

Enter the number of the port that is the new programming port. If the programming port is changed, the current connection between the State Engine and ECLiPS is no longer valid. ECLiPS must now be connected to the new port.

Enable/Disable RS–422 Designation for the Ports

This option controls whether the designated port is an RS–422/485 port or an RS–232 port. Either port can be either designation.

Download Project to the Controller

Use this option to download a new program to the State Engine. First the option to select a new directory is presented and then a list of projects from the current directory. Select the project top be downloaded to the controller. A new project cannot be downloaded when the program is running. When the program is running, ECLiPS asks whether to stop the program or cancel the download.

Reset and Clear the State Engine

Use this option to clear the State Engine program memory and return all configuration parameters to the default condition. The State Engine setup options returned to their default state are CCM port setup, error handling setup, programming port selection, and automatic run setup.

Monitor

Monitor is an on–going display of the current values of all the selected elements. Up to 6 elements may be included in a monitor table at any time and up to 10 monitor tables may be defined. Monitor tables have user given names of up to 30 characters each. Spaces can be part of a Monitor Table name.

Enable Display

This function activates the display of the monitor tables. While a table is displayed, the user may press <Tab> to switch to the next table. If there are no tables defined, the user will be prompted to add a new one.

Disable Display

This function disables the monitor display but the tables are saved for the next time they are needed.

Add a New Monitor Table

This function allows the user to add a new monitor table as long as there are currently less than 10. It acts the same way as Monitor Setup. If no name is given to the table, it will default to “Number X” where X is a number between 1 and 10 corresponding with the position in the list of tables.

Select a Monitor Table

This function will display a list of the available monitor tables and allow the user to select one. The selected table will then become active in the monitor window.

Modify the Current Monitor Table

This allows a user to add and remove elements from the current monitor table. The list below shows the types of elements that may be monitored.

- Analog Channels
- String and Character Variables
- Digital Points
- Internal Flags
- Numeric (Integer and Floating Point) Variables
- Register Variables
- Reserved System Variables
- Current State of a Task

Remove Current Table

This function removes the current monitor table.

Clear All Monitor Tables

This function clears all of the defined monitor tables.

View

View English Text

This function allows the user to view the English text while debugging. There are several functions available while the text is being viewed.

Find a Text String

To find a string of text in the English Text, press <ALT + F>, then select Find a Text String. Enter the String of text to find and press <ENTER>.

Perform the Next Find

To find another occurrence of the string of text entered in the Find option, press <F5> or press <ALT + F> then select the “Perform the Next Find” of the “Entered Test String” option from the menu.

Select Another Task Group

This option displays a list of Task Groups so another Task Group may be viewed.

Go to the Beginning of Another Task

To go to the beginning of another task press <F7> and select the Task name from a list of the defined Task names in the project. The cursor will move to the beginning of the selected Task.

Look up the Current Word and Display its Value

This powerful function displays the definition of the word at the cursor. If the word has some value, then that value is displayed and an option presented to change that value.

If the cursor is located on a Task name, the current State of the Task is displayed, and an option to change the current State is offered. If the cursor is on an I/O name, the definition is displayed, and from the window the current value can be forced to another value. Similarly, variable definitions are displayed and options presented to change those values.

Current State of Each Task

This function displays the active State of every Task. The display is updated twice per second providing a realistic view of the current status of the process. Each Task name is displayed, followed by its current State.

View Event Queue

The Event Queue includes time stamped data about the Controller such as Run, Halt, Power Up, Power Down, Critical Run–T ime Errors, etc. The user may view up to the last 64 entries with 6 being the default number to view.

View System Status

This includes system date, time, memory usage and current scan rates in milliseconds. The scan rate information specifies the average scan rates for the last 1000 scans. The scan is also split into the time for State Logic to be scanned and the time for the transfer of I/O data between the SLP and CPU.

Trace

The Trace is a collection of the most recent program State transitions. This option allows the user to display the history of State transitions on the screen for all Tasks or selected Tasks.

Upload Trace from the Controller

This function will upload the Trace data from the controller and display it for the user to browse through. The Trace is also stored in a file on the logged disk and directory so that it may be viewed at a later time. Each upload of the Trace information, overwrites the previous upload saved to disk. The Trace is stored in a file with the project name and a .TRC extension, for example: CONVEYOR.TRC.

The display shows each State transition, listing the starting State, the resulting State and the time of the transition. The most recent transition is at the top of the list.

There are string search functions available to help find a specific State transition. Press <ALT + F> to access these search functions. Also the up and down arrows and <Pg Up> and <Pg Dn> keys are used to move through the listing.

Display the Previously Uploaded Trace

This function reads and displays the file of the previous Trace display which is stored on disk.

Setup Trace List

This option allows the user to setup the Tasks that will be traced (stored in the Trace buffer). Upon entering this function, a list of the currently traced Tasks will be displayed and the user may add or delete Tasks.

Clear Trace List

Reset the buffer to Trace all Tasks.

Force

These functions allow a user to force Digital points and Internal Flags and analog channels. To change variable values use the CHANGE option. Forced items remain in the forced state independent of real world conditions or program actions. The only way that a forced condition changes is for the force option to be used again to either change the forced value or clear the force. Once the force is cleared, outputs go to whatever conditions specified by the program and inputs go to whatever value is set by the CPU.

Set or Modify Force List

When invoked, this function displays a list of all currently forced I/O and internal flags and allows the user to add, remove or change any of the entries.

Clear All Forces

This function allows the user to completely clear the force table. Entries cleared are removed from the table and the values allowed to be changed by the State Logic program or by the CPU.

Display

The display function displays the current value of the selected element together with the current time in the Terminal Log. When the type of data to display is selected, a list of names of that type are displayed. Select the name to display from this list. The following data types can be displayed with the Display function:

Analog Channels
String and Character Variables
Digital Points
Internal Flags
Numeric (Integer and Floating Point) Variables
Register Variables
Reserved System Variables
Current State of a Task

Change

The change function allows the user to enter a new value for the selected item. After the new value is entered, it is sent to the controller immediately. Also the change is reported to the Terminal Log together with the current time. The following data types can be changed using the Change function:

String and Character Variables
Numeric (Integer and Floating Point) Variables
Register Variables
Current State of a Task

PID Loops

This option appears on the debug main menu only if the program has defined a PID loop. Choose this option to tune a PID loop while the process is running. To tune a PID loop, change the parameters displayed.

a80026

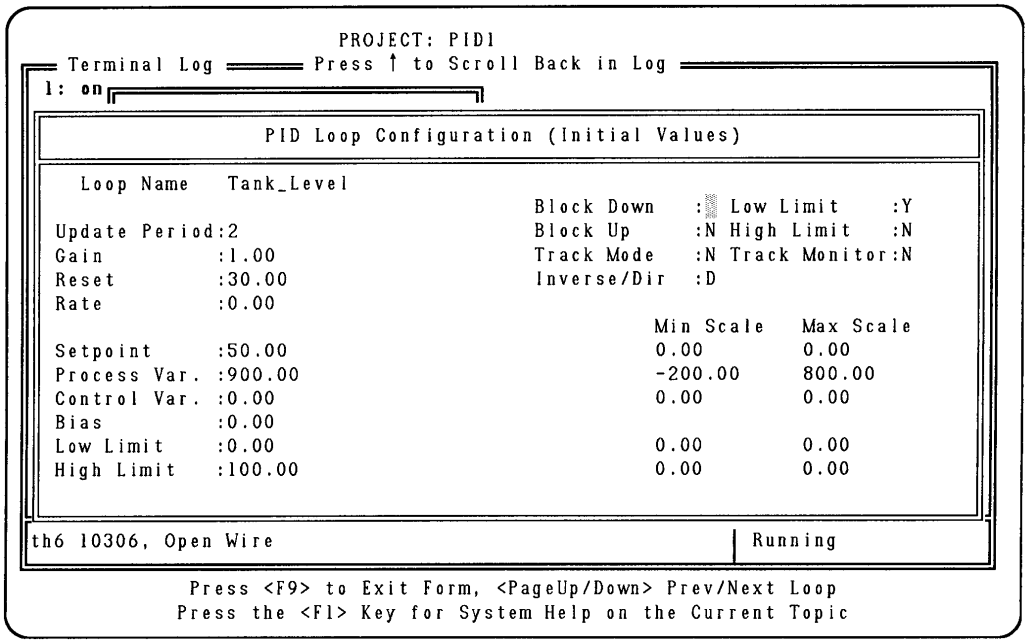


Figure 8-10. PID–Loop Tuning Screen

PLC I/O

This option is used to view definitions of program names referring to CPU memory locations. Analog point, discrete point, and register variable definitions are all displayed using this option.

After selecting the category of I/O to view, a list shows all of the names of that category plus the type and location number. Select the name from the list to see the same form used to define that I/O name. The input/output designation also appears in this form.

Faults

Use this option to display and clear both PLC and I/O faults. When the option to display a fault table is displayed that table is then displayed on the screen. To clear a fault table, just select that option from the menu.

The display lists the current time, the last time faults were cleared, total number of faults in the table, and faults since the last time faults were cleared.

The PLC Faults Table displays general system faults for different slots in the Series 90–70 chassis. The I/O Fault Table displays specific circuit faults. Both tables list the date and time of the fault.

Quit

Leave Debug Mode and return to Main Menu.

Online Modify

The Online Modify option is used to change the program without halting program execution. One Task of the program is changed at a time. When the Task changes are complete, ECLiPS translates the entire project, checking for any programming errors. The State that is to be active when the changed Task starts executing is specified and then the changed Task is downloaded. The changed Task replaces the old one at the start of the first scan after the Task is downloaded.

Modlfy Mode Menu Optlons

The programmer is can change the project while the program is running using most of the same menu options available in Program Mode. The differences in the menu options offered in Modify Mode reflect the editing limitations for this mode.

- One Task is modified at a time.
- No discrete, analog l/O, or Register Variables may be added or changed.
- States may be added or deleted, but the number of States in the modified Task is limited to 2 more than the number of States in the largest Task of the project.
- Variables may be added but not made retentive, and existing variables may not be changed or deleted.

Onllne Modify Restrictions

Online Modify Menu Differences from Program Mode

ADD – The changes in the ADD menu from Program Mode are that Tasks cannot be added and the State Timeout Diagnostic is not available.

LIST – The LIST menu is the same for both Program and Modify Modes, but there are some restrictions limiting changes and additions for I/O and variables.

DEFINE – The Modify Mode DEFINE menu does not allow new I/O to be defined and the PLC I/O Scan Rate option which specifies how often analog I/O are scanned is not available.

FIND – All of the options from the Program Mode FIND menu are offered in Modify Mode except the option to find the start of another Task.

PROJECT – The only 2 options on the Modify Mode PROJECT menu are Translate and Download and just Translate. The Translate Only option does an error check an then transfers to the Debug Mode. The Task changes made in Modify Mode are not loaded into the controller. These changes are not abandoned yet. When the QUIT option is selected from the DEBUG menu, ECLiPS asks whether to return to Online Editing or abandon the changes.

TEXT – The TEXT options are identical to the Program Mode options.

SECURITY – There is no SECURITY option on the Modify Mode menu.

QUIT – The QUIT option returns to the Main Menu.

Setup and ECLIPS Memory Usage

The setup option is used to configure the ECLiPS system. The two options that can be selected are the maximum number of defined data elements (Discrete I/O, Flags, Analog I/O and Variables) to be used in the project and the maximum number of States in the project. The number of data elements is may be from 500 to 3000 in increments of 500. The number of States is 500 to 3000 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 error message and allows the Setup limits to be reselected.

There are two reasons to select smaller than maximum size limits for the ECLiPS setup options. First ECLiPS requires less free conventional memory with lower defined limits. The second reason is that ECLiPS uses a page swapping system and therefore does some operations faster when using the smaller limits.

If there is not enough conventional 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. Another way to deal with the problem of not enough memory to run ECLiPS, is to free up conventional memory in your PC by eliminating device drivers and Terminate and Stay Resident (TSR) programs by changing your CONFIG.SYS and AUTOEXEC.BAT files. If you are using DOS 5.x, loading DOS HIGH also frees up additional conventional memory. To use the maximum settings for the number of States and Variables, DOS must be loaded into high memory area.

ECLiPS also 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 more States and Variables and also 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. One megabyte of added memory is enough to provide the maximum capacity and speed for ECLiPS operations.

Quit

Use this option to exit ECLiPS and return to DOS.

Specifications

These specifications are for ECLiPS when running under DOS 5.0 with DOS loaded in the Upper Memory Area. The only specifications that change when running under an older DOS is the number of Tasks, which becomes 100, and the number of Names, which becomes 700. DOS must always be loaded into the High Memory Area when using DOS 5.0 with ECLiPS.

Tasks	256
States Per Task	254
Total Number of States	3000
Statements per State	Unlimited
Integer Variables (range −32768 to +32767)	1000
Floating Point Variables (range +/− 1.175494E−38 to +/− 3.402823E+38)	1000
String Variables	100
String Variable Size acters	80 Characters
Character Variables	64
PID Loops	20
Internal Flags	1000
Number of Timers	Unlimited
Timer Resolution cond	1/100 second
Number of Characters per Write Term	512
State Changes Listed in Trace Display	80
Force Table Size	32
Monitor Table Size	6 entries
Monitor Tables	10
Total Variable Plus I/O	3000

A

Actuate, 8-21
Actuate Term, 8-4
Add, 8-17 , 8-21
Add Term, 8-5
AM, 8-23
Analog Inputs / Outputs, 8-61
Analog Scaling, 8-63
Analog Variables, 8-14
AND, 8-9 , 8-23
Anti–Reset Windup, 8-27
ARCTAN, 8-22
ASCII Variables, 8-14
AUTOEXEC.BAT, 8-89
Automatic Program Execution, 8-82
Automatically Start Program, 8-68

B

Base, 8-56 , 8-75
BCD, 8-38
BCD_In_Convert, 8-38
Bitwise_And, 8-22
Bitwise_Or, 8-22
Block_Down, 8-28
Block_Up, 8-28
Bumpless Transfer, 8-26

C

Calculations, 5-3 , 8-15
Cascaded PID Loops, 8-26
CCM Protocol Listing, 7-6
CCM2, 8-58 , 8-82
Change State, 8-17
Change State Term, 8-6
Character, 8-15 , 8-17 , 8-18
Character Variables, 3-8 , 8-13 , 8-14 , 8-61
Clear_Bit, 8-4 , 8-17 , 8-21
Clearing Outputs at Power Up, 8-53
Clock, 8-19 , 8-69
Co–P rocessor, 8-50
Comments, 5-7 , 7-8
Communication Functions, 2-8
Conditional Expressions, 8-2 , 8-9
Conditional Terms, 8-20
CONFIG.SYS, 8-89
Configuration, 8-17
Configuring, 6-1
Configuring Serial Ports, 8-58
Controller, 8-51
Copy, 8-76
Copy_Table_To_Table, 8-36
COS, 8-22
Counter, 4-20
CPU, 8-48
Cross Reference List, 7-6
Current State, 8-12 , 8-18 , 8-61 , 8-85

D

Data List, 7-4
Data Types, 8-15
Day, 8-19
Day_of_Week, 8-19
Debug Mode, 8-81
Define, 4-14 , 8-73
Define_Table, 8-34
Diagnostics, 2-11 , 4-20 , 8-70
DigitalI / O, 8-4 , 8-61
Discrete, 8-46
Display Date and Time, 8-44
Divide, 8-17 , 8-21
Divide Term, 8-6
Documentation, 7-1
DOS, 1-3
DOS 5.0, 8-90
Download, 6-3 , 8-77 , 8-83
Drive, 8-76

E

ECLiPS Memory, 8-89
Energize, 8-21
English Code, 7-2
Error, 6-5 , 8-82
Error Setup, 8-68
Event Queue, 8-85
EXP, 8-22
Expanded Memory, 1-3 , 8-90
Expressions, 8-2

F

Faults, 6-7 , 8-88
Files, 8-57 , 8-68 , 8-77
Filler Words, 3-6 , 8-24 , 8-72
Find, 8-75
Finite States, 2-1
Flags, 3-8 , 8-4 , 8-72
Floating Point, 8-14
Floating point, 8-15
Floating Point Variables, 3-8 , 8-61
For, 8-20
Force, 6-7 , 8-86
Formatting, 8-8
Friday, 8-23
FROM, 8-12 , 8-23
Function Key, 8-45
Functional Expressions, 8-2 , 8-3 , 8-17
Functional Terms, 8-21

G

Get User Input, 8-44
GO, 8-13 , 8-19 , 8-21
GO Term, 8-6

H

HALT, 6-5 , 8-21
Hardware Requirements, 1-3
Hot Keys, 5-10 , 8-45
Hour, 8-19

I

I/O Map, 7-3
If, 8-9 , 8-20
Import, 8-77
Inactive, 8-24
Inactive State, 8-6
Initialization, 6-2
Initializing Tables, 8-35
Input, 8-47
Installation, 1-3
Integer, 8-15
Integer Variables, 3-7 , 8-14 , 8-61
Internal Flags, 8-61

K

Keyboard Definitions, 8-45
Keywords, 3-6 , 8-1 , 8-20 , 8-72

L

Language Structure, 8-2
Lists, 5-11 , 8-71
LN, 8-22
Log, 6-4 , 8-68
Log File, 8-82
Logicmaster 90, 8-57

M

Make, 8-21
Make Term, 8-4 , 8-17
Mathematical Calculations, 8-13

Mathematical Operators, 8-22
Max_Time, 8-23
Memory Capacities, 8-48
Memory Types, 8-48
Menus, 5-10
Minute, 8-19
Modulus, 8-22
Monday, 8-23
Monitor, 6-5 , 8-83
Month, 8-19
Multiply, 8-17 , 8-21
Multiply Term, 8-5

N

Names, 3-4 , 5-6
NOT, 8-9 , 8-23
Numeric Variables, 8-14

O

Off, 8-23
On, 8-23
Online Modify, 8-88
OR, 8-9 , 8-23 , 8-24
Output, 8-47
Output_BCD_Convert, 8-39

P

Path, 8-76
Perform, 8-19 , 8-21 , 8-70
Perform Functions, 8-9 , 8-17 , 8-32
PID Command and Status Bits, 8-30
PID Inputs, 8-30
PID Loop Parameters, 8-61
PID Loops, 6-7 , 8-8 , 8-17 , 8-24 , 8-72 , 8-87
PID Output, 8-31
PID Parameters, 8-29
PID Value, 8-15 , 8-19
PLC Type, 8-74
PM, 8-24
Precedence, 8-13
Print, 7-1 , 8-77
Printer, 8-81
Program Mode, 8-69
Programming Port, 8-58 , 8-83

R

R_Register, 8-7 , 8-24
R_Register Variables, 8-14
RANDOM, 8-22
Ranges, 8-55 , 8-75
READ, 4-19 , 5-4 , 8-12 , 8-19 , 8-20
Register, 8-46
Register Variables, 3-8 , 8-72
Relational Operators, 8-23
Relational Terms, 8-11
Replace, 8-75
Restart_In_Last_STATE, 8-24
Restart_In_Last_State, 8-3
Resume_Task, 8-6 , 8-21
Retrieve, 8-76
RS–232, 8-58 , 8-83
RS–422 / 485, 8-58 , 8-83
RUN, 6-5

S

Saturday, 8-24
Save, 8-76
Scan, 2-9 , 3-9 , 5-3 , 5-7 , 8-63
Second, 8-19
Seconds, 8-24
Security, 8-80
Serial Data, 8-17
Serial Ports, 8-7 , 8-58 , 8-78
Series 90 Scan Rates, 8-74

Set_Bit, 8-4 , 8-17 , 8-21

Set_Commport, 8-8 , 8-13 , 8-17 , 8-21 , 8-58

Setup, 8-89

Shift_Register, 8-40

Simulation Mode, 6-5 , 8-68 , 8-82

SIN, 8-22

SLP / CPUInterface, 8-46 , 8-49

Specifications, 8-90

SQRT, 8-22

Start_PID, 8-17 , 8-21

State Engine Configuration, 8-58 , 8-68 , 8-82

State Logic Control Theory, 2-1

STATEMENTS, 2-5 , 2-7 , 3-3 , 3-5 , 8-1 , 8-2 , 8-3

STATES, 2-1 , 2-5 , 2-6 , 3-2 , 4-3 , 8-1 , 8-3 , 8-6 , 8-12 , 8-19 , 8-21 , 8-70

Stop_PID, 8-17 , 8-21

String, 8-15 , 8-17 , 8-18

String Manipulation, 8-40

String Variables, 3-8 , 8-14 , 8-61

Subtract, 8-17 , 8-21

Subtract Term, 8-5

Sunday, 8-24

Suspend_Task, 8-6 , 8-21

Swap_Table_Value, 8-34

Synonyms, 3-6

System Clock, 8-19 , 8-69

T

Table, 8-33

TAN, 8-22

Task Group, 8-79

Task / State List, 7-5

TASKS, 2-3 , 2-4 , 2-5 , 2-6 , 2-9 , 3-1 , 4-3 , 5-3 , 8-1 , 8-3 , 8-6 , 8-12 , 8-19 , 8-21 , 8-61 , 8-70

Technical Support Line, 1-4

Terminal Log, 8-68

Text String, 8-75 , 8-84

Third Party VME Interface, 8-53

Thursday, 8-24

Time Counter, 8-43

Time Variables, 3-8 , 8-14 , 8-19

Timer, 5-5 , 8-10 , 8-18

Trace, 6-7 , 8-85

Track_Mode, 8-27

Translate, 8-77

Tuesday, 8-24

Tuning, 8-28

U

Underscores, 5-6

UNINSTALL, 1-3

User Menu, 8-44

V

Variables, 3-7 , 8-72

View, 6-6 , 8-84

W

Wait, 8-20

Wednesday, 8-24

With, 8-24

Write, 4-19 , 5-2 , 8-21

Write Term, 8-7