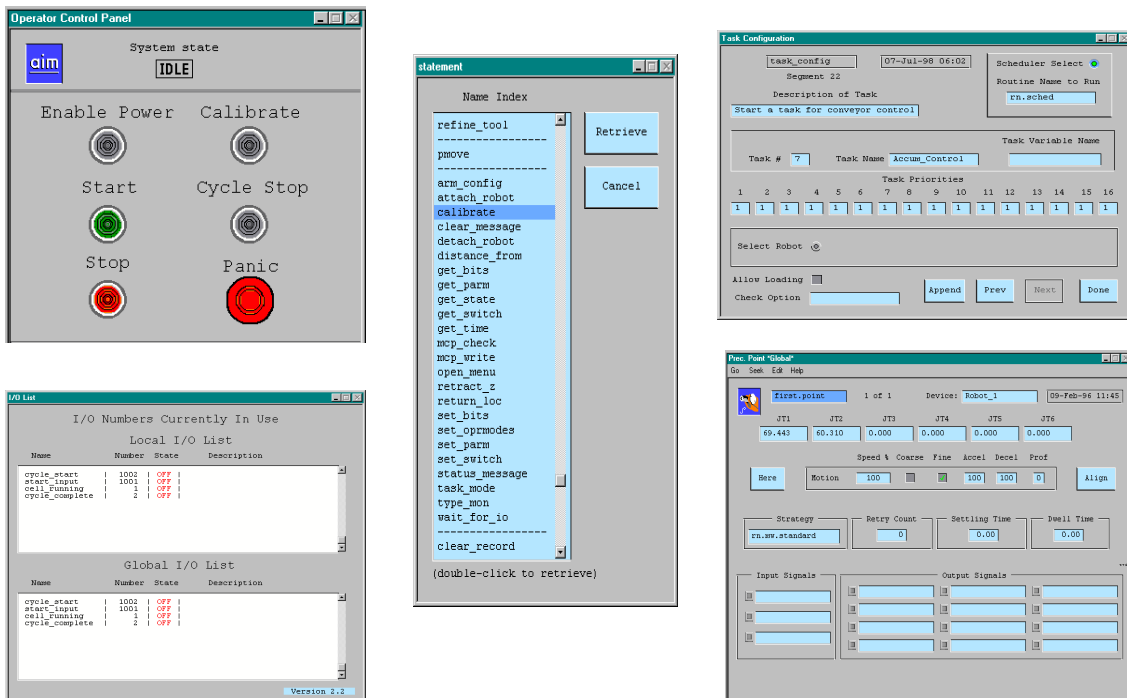


AIM Utilities

User's & Reference Guide

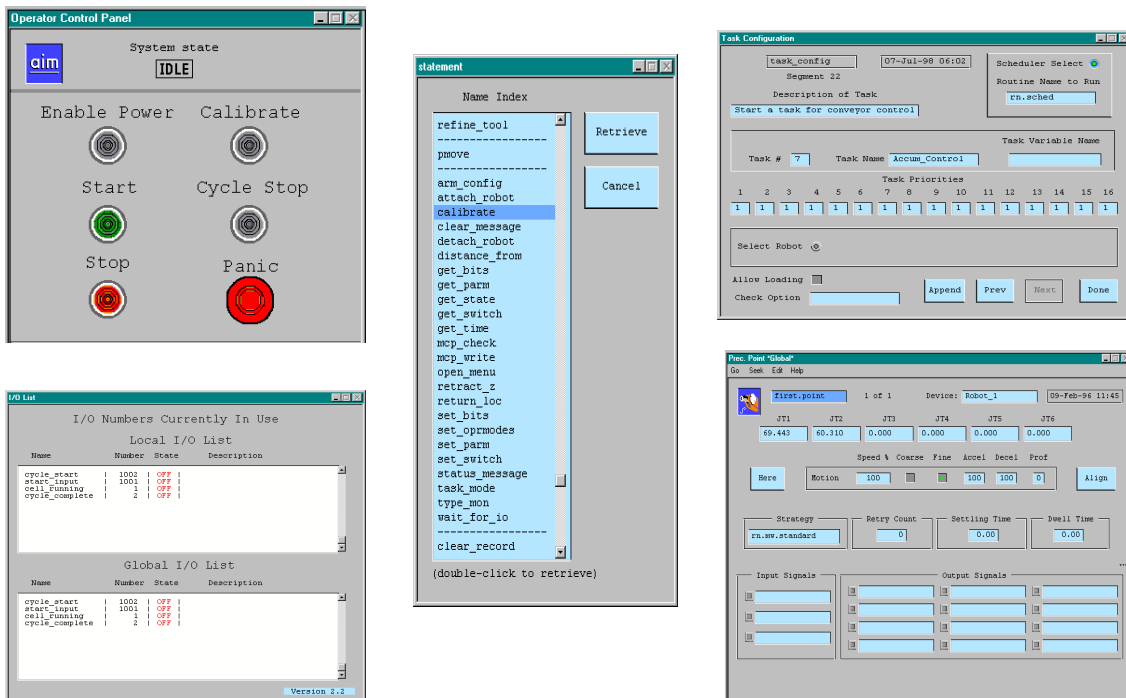


Version 3.2 B



AIM Utilities

User's & Reference Guide



Part Number RDASG-C0004 Rev. 3.2B

March 1999



150 Rose Orchard Way • San Jose, CA 95134 • USA • Phone (408) 432-0888 • Fax (408) 432-8707

Otto-Hahn-Strasse 23 • 44227 Dortmund • Germany • Phone (49) 231.75.89.40 • Fax(49) 231.75.89.450

41, rue du Saule Trapu • 91300 • Massy • France • Phone (33) 1.69.19.16.16 • Fax (33) 1.69.32.04.62

1-2, Aza Nakahara Mitsuya-Cho • Toyohashi, Aichi-Ken • 441-31 • Japan • (81) 532.65.2391 • Fax (81) 532.65.2390

The information contained herein is the property of Adept Technology, Inc. and shall not be reproduced in whole or in part without prior written approval of Adept Technology, Inc. The information herein is subject to change without notice and should not be construed as a commitment by Adept Technology, Inc. This manual is periodically reviewed and revised.

Adept Technology, Inc. assumes no responsibility for any errors or omissions in this document. Critical evaluation of this manual by the user is welcomed. Your comments assist us in preparation of future documentation. A form is provided at the back of the book for submitting your comments.

Copyright © 1992, 1997 by Adept Technology, Inc. All rights reserved.

The Adept logo is a registered trademark of Adept Technology, Inc.

Adept, AdeptOne, AdeptOne-MV, AdeptThree, AdeptThree-MV, PackOne, PackOne-MV, HyperDrive, Adept 550, Adept 550 CleanRoom, Adept 1850, Adept 1850XP, A-Series, S-Series, Adept MC, Adept CC, Adept IC, Adept OC, Adept MV, AdeptVision, AIM, VisionWare, AdeptMotion, MotionWare, PalletWare, FlexFeedWare, AdeptNet, AdeptFTP, AdeptNFS, AdeptTCP/IP, AdeptForce, AdeptModules, AdeptWindows, AdeptWindows PC, AdeptWindows DDE, AdeptWindows Offline Editor, and V⁺ are trademarks of Adept Technology, Inc.

Any trademarks from other companies used in this publication are the property of those respective companies.

Printed in the United States of America

Table Of Contents

1	Introduction	13
1.1	What is The AIM Utilities Module?	14
1.2	Using This Manual	14
1.3	Software Overview	15
	Manual Control Pendant Statements	15
	Statements for Auto-Startup and Robot Calibration	15
	Example Sequences and Databases	15
2	Installation	17
2.1	Before You Begin	18
2.2	Installation Procedure	24
2.3	Verifying Installation	25
2.4	Installing Utilities to existing Applications	25
2.5	Installing AIM Dispense Module or AIM PalletWare	25
2.6	Selecting Default AIM Modules	26
2.7	AUTO File Installation	26
3	AIM Utilities Menus	27
3.1	Software Module Loading	29
3.2	Software Loader Record Menu	31
3.3	Task Configurator	33
	Task Configurator Record Menu	35
	Task Priorities Display Page	37
3.4	Software Plug-In Utilities	38
	Uninstall Software Module	39
	Create the Software Module Install File	41
3.5	Operator Control Panels	43
3.6	Precision Point Database	44
3.7	MCP Emulator	46
3.8	NFS Disk Drive Mounting Utilities	48
	NFS Mount Status Menu Page	49
3.9	Tool Database	50
3.10	IO Status Display Page	52
3.11	Cell Status Display Page	53
3.12	MCP Teach Menu Changes	54

4	Using AIM Utilities	55
4.1	AIM Utilities Statements	56
	AIO	56
	ARM_CONFIG	56
	ATTACH_ROBOT	56
	CALIBRATE	57
	CLEAR_MESSAGE	57
	DETACH	57
	DISTANCE_FROM	57
	GET_BITS	58
	GET_PARM	58
	GET_STATE	59
	GET_SWITCH	59
	GET_TIME	60
	J4_INERTIA	61
	MCP_CHECK	61
	MCP_WRITE	61
	MOUNT_NFS	62
	MOVE_UT	63
	OPEN_MENU	64
	PMOVE	64
	RETRACT_Z	64
	RETURN_LOC	65
	RETURN_JTS	65
	SET_BITS	65
	SET_OPRMODES	66
	SET_PARM	66
	SET_PATH	66
	SET_SWITCH	67
	SHIFT_MOVE	67
	STATUS_MESSAGE	68
	TASK_MODE	68
	WAIT_FOR_IO	69
	TYPE_MON	69
4.2	Calibration from Sequences	70
4.3	Using the Message Statements	71
4.4	MCP example	71
4.5	Tool Offset Teaching	72
	Testing the Tool Offset	74
4.6	Paths Relative to Frames	75
4.7	Precision Point Module	75
4.8	Software Plug-Ins	75
4.9	Example Sequences	76

A	Sequence Examples	77
A.1	Introduction and Overview	78
	Start-Up and Calibrate Sequences	78
	Calibrate Sequence	79
	Start-Up Sequence Example from PalletWare	80
	Start Button Control Sequence	81
	Cell Control Sequence Example	82
	Second Cell Control Example	86
	Robot Main Sequence	91
	Main Sequence Example from PalletWare	92
	MCP Sequence Example	101
B	Quick-Change Documentation	103
B.1	Introduction and Overview	104
	The CHANGE_HAND Statement	104
	Installation Procedure	104
B.2	The Quick Change Database Menu	107
	Quick-Change Database	107
	Quick-Change IO Debug	108
B.3	Quick-Change Operation	112
	The CHANGE_HAND Statement	112
B.4	Quick-Change Example	113
	Quick-Change Database	113
B.5	Quick-Change Routines	117
	ch.set.io()	119
	rn.ch.get()	120
	rn.ch.put()	121
C	Event I/O Manager	123
C.1	Introduction and Overview	124
C.2	Event Monitor Editing	124
C.3	Event Initialization Database	126
C.4	Running the Event Monitor	126
D	HPGL Graphics Module	127
D.1	Introduction and Overview	128
E	ConnectWare Module	129
E.1	Introduction and Overview	130
E.2	Sequence Statements	130
	DDE_CONNECT	130
	DDE_DISCONNECT	130
	DDE_READ	131
	DDE_WRITE	131

F	TCP-IP V+ Server	
F.1	Introduction and Overview	134
	TCP Message Format.	134
F.2	TCP Debug Menu	135
F.3	TCP Initialization Database	136
	Index	137

List of Figures

Figure 3-1	Software Module Loader	29
Figure 3-2	Software Loader Record	31
Figure 3-3	Task Configurator Main Menu	33
Figure 3-4	Task Configurator Record Menu	35
Figure 3-5	Task Priorities Display	37
Figure 3-6	Update Software Module	38
Figure 3-7	Uninstall Software Module	39
Figure 3-8	Create Software INF File	41
Figure 3-9	Standard Operator Control Panel	43
Figure 3-10	Precision Point Menu	44
Figure 3-11	MCP Emulator (Step Mode).	46
Figure 3-12	NFS Mount Menu Page	48
Figure 3-13	Tool Offset Menu	50
Figure 3-14	Cell Status Menu	53
Figure 3-15	MCP Teach Buttons.	54
Figure 4-1	Calibration Sequence.	70
Figure 4-2	MCP Sequence	71
Figure 4-3	Cartesian Tool Offset.	72
Figure 4-4	Tool Offset Utility Menu Display.	72
Figure 4-5	Calculating XYZ Offsets.	73
Figure 4-6	General Tool Offset	74
Figure 4-7	Available Software Plug-Ins.	75
Figure A-1	Auto Start-Up Sequence	78
Figure A-2	Calibrate Sequence Example	79
Figure A-3	PalletWare Start-Up.	80
Figure A-4	Start Button Sequence Example	81
Figure A-5	Cell Control, (First Example Page 1)	82
Figure A-14	Dispense Robot Main Sequence	91
Figure A-15	PalletWare Main Sequence (Page 1).	92
Figure A-24	MCP Example	101
Figure B-1	Quick-Change Database Record Form	107
Figure B-2	Quick-Change Database Record End-Effector 1	115
Figure B-3	Quick-Change Database Record End-Effector 2	116
Figure C-1	Event Monitor Database	124
Figure D-1	HPGL Instructions	128
Figure F-1	TCP Debug Window	135

List of Tables

Table 2-1	AIM Utilities Files	19
Table B-1	Quick-Change Module Files.	105
Table B-2	Quick-Change Database Fields.	108
Table B-3	QUICK-CHANGE Database Record Definition	110
Table B-4	Quick-Change Database Variable Name.	111
Table B-5	Change_Hand Example Tool I/O.	113
Table B-6	Documented Routines	117

Introduction

1

1.1 What is The AIM Utility Module?	14
1.2 Using This Manual	14
1.3 Software Overview	15
Manual Control Pendant Statements	15
Statements for Auto-Startup and Robot Calibration	15
Example Sequences and Databases	15
Global Variables included	15

1.1 What is The AIM Utilities Module?

The AIM Utilities Module is an AIM application module designed specifically with the implementation of AIM in mind. The module simplifies the process of programming AIM by providing more AIM sequence statements as well as other high end features. The AIM Utilities Module includes Operator Control panels and additional icons.

Unlike other AIM modules, the AIM Utilities Module must be used in conjunction with the MotionWare, PCB, or VisionWare application module.

1.2 Using This Manual

This manual presents a detailed description of the AIM Utilities Module. We assume you are familiar with AIM, and the basic operation of Adept robot systems. See the *Adept MV Controller User's Guide* and *Adept Robot User's Guide* for details on the robot and controller. See the *MotionWare User's Guide* for details on using the basic AIM system. If you will be customizing the AIM Utilities Module, we assume you are familiar with the V⁺ operating system and language. See the AIM reference manuals and the *V+ Language Reference Guide* for information on customizing and writing V+.

Chapter 1 presents an overview of the software features of the AIM Utilities Module. This chapter should be read by all AIM Utilities Module users.

Chapter 2 describes the software installation and startup procedures for the AIM Utilities Module. This chapter is intended for system customizers who will install the software and users interested in the AIM Utilities Module software organization.

Chapter 3 describes the menus and databases used with the AIM Utilities Module. All users should review this chapter and become familiar with these databases.

Chapter 4 details the uses of the AIM Utilities Module. The AIM statements and other features are discussed here. Example applications are reviewed. This chapter should be read by all AIM Utility Module users.

Appendix A contains example sequences from the database examples disk.

Appendix B contains information about the Quick-Change Module. It describes the databases, and operation of the software package.

1.3 Software Overview

AIM Utilities is an application module of the AIM software system. It was developed to speed implementation of Adept's AIM Systems. To achieve these goals, AIM Utilities adds 30+ new task-level statements. These statements allow the user to create calibration sequences, easier use of IO, access to internal AIM settings, statements to create MCP pendant displays from a sequence and many other features.

AIM Utilities provides two software utilities. One allows easy installation and selection of software modules. The other utility allows the ability to add and select executing tasks. These items used to require customization of AIM software. These utilities allow the customizer to create software libraries that can be easier to install by the user.

AIM Utilities also adds the ability to program precision point motions. This allows the user to program robots with locations that have multiple arm configurations.

AIM Utilities provides the ability to program paths relative to frames of references. This is needed when user wants to use the path features for process type applications.

AIM Utilities requires AIM version 3.2+ and the Adept V⁺ operating system version 11.4 or higher. Some features of the AIM Utilities are described below. Details are presented in subsequent chapters.

Manual Control Pendant Statements

These statements allow the user to write sequences to display menu's and provide an operator interface using Adept's MCP.

Statements for Auto-Startup and Robot Calibration

Several statements have been added to aid in typical startups of Adept's equipment. An example sequence is provided to better show the uses of these instructions.

Example Sequences and Databases

The AIM Utilities software package includes example databases and sequences that are provided to aid in the development of your applications.

1.4 Hardware Requirements

The PalletWare Module requires the following Adept controller options:

- Adept A-series controller or MV controller with AdeptWindows
- Adept Teach Pendant (MCP)
- Mouse, Monitor, and Keyboard (not required with AdeptWindows)
- 68030 / 68040 / 68060 system processor (with 8 Mb's of RAM) The 68060 processor is recommended for high end applications that demand more processing speed.

1.5 How Can I Get Help?

Service Calls

Adept Technology maintains a fully staffed Customer Service Center at its headquarters in San Jose, CA.

(800) 232-3378

When calling Customer Service, please have the serial number of the robot (if you have one), the serial number of the controller, and the system software version. The system software version is available by issuing the command "ID" at the system prompt.

Training Information

For Adept Training Course information, please call (408) 434-5024.

Application Information

There is also a dedicated phone line for assistance with robot applications. The AIM Utilities Software Package is considered an Adept RDA Services product. Please call the Adept Cincinnati Office with software related questions at 513-792-0266.

International Customer Assistance

For information on training, service, or applications, Adept also has a Customer Service Center in Dortmund, Germany. The phone number is: (49) 231/75 89 40.

Outside of Europe or the USA, call (408) 434-5000.

Installation **2**

2.1 Before You Begin	18
2.2 Installation Procedure	24
2.3 Verifying Installation	25
2.4 Installing Dispense Module or AIM PalletWare	25
2.5 Selecting Default AIM Modules	25

2.1 Before You Begin

This chapter describes the AIM Utilities installation procedure. To simplify the installation, have the **Adept Utility Disk** handy. The Utility Disk is supplied with all Adept controllers. Complete instructions for using the diskcopy utility are in the *Instructions for Adept Utility Programs*.

The AIM 3.2 software files, and the MotionWare application module files must already be installed. See the *MotionWare User's Guide*

NOTE: The procedures outlined in this chapter will replace MotionWare files. Any previous AIM customizations may be replaced. If you have customized AIM, check below to assure your files will not be overwritten before installing the AIM Utilities Module.

Table 2-1 lists the files that are included with AIM Utilities. Three diskettes are provided in the AIM Utilities package. They are titled AIM Utilities Software, AIM Utilities Example Databases, and AIM Utilities Software Plug-Ins. The AIM Utilities Software disk contains all of the software for AIM Utilities, this includes the runtime software, the database definition files, and the database files. The files with *.SQU* filename extensions are *squeezed*, i.e. all comments have been removed from the files. Squeezed files require less memory when they are loaded than their commented non-squeezed *.V2* counterparts. The squeezed files and their *.V2* counterparts are otherwise identical. (See the *MotionWare User's Guide* for a description of all AIM file types.).

The disk labeled Example Databases contains AIM database modules of examples for different applications. These modules can be loaded by the user to show how to program different applications.

The disk labeled Software Plug-Ins contains additional software utilities that can be loaded into the system via the Software Loader Menu and the Install button option. These plug-ins are generally nice features that not all customers will need to use.

To install the AIM Utilities, follow the steps in section 2.2. The installation procedure copies all the files listed in Table 2-1 onto the hard disk drive. Files marked with a P are protected files and cannot be read by the user. Files marked with a ✓ replace existing AIM files.

Table 2-1 AIM Utilities Files

Disk Files		Contents
AIM Utilities Software Disk		
AIMUTIL.V2		AIM Utilities Runtime Software routines
INFMOD.OVR	P	Plug-In Utilities Menu routines
INFMOD.MNU		Plug-In Utilities AIM Menu
IOL.MNU		IO List Menu file
IOLMOD.OVR		IO List menu routines
IOLMOD.INF		IO List Plug-In Configuration File
IOLRUN.V2		IO List Runtime Routines
LAIM.V2		Loader routines for generalized AIM
MENU_UT.V2	P	Menu routines for AIM Routines
MNT.MNU		NFS AIM Menu files
MNTINI.DB		NFS Initialization Database
MNTMOD.INF		NFS Mounting Plug-In file
MNTMOD.OVR		NFS initialization overlay file
MNTRUN.V2		NFS runtime routines
MOW.MNU	✓	Location Record Menu
PATH.SQU	P ✓	Modified Path routines
PLGMAIN.ADV		Plug-In software advisor
PLGMAIN.MNU		Plug-In menus
PLGMOD.OVR		Plug-In menu routines
PPOINT.V2		Precision Point runtime software
PPT.MNU		Precision Point AIM Menus

PPT.RFD		Precision Point RFD file
PPT.DB		Precision Point database
PPTICON.DAT		Precision Point Icon data file
PPTMOD.OV2		Precision Point Initialization commented file
PPTMOD.INF		Precision Point Plug-In file
PPTMOD.OVR		Precision Point Initialization file
STATPPT.DB		Precision Point Sequence statement file
STATUT.DB		AIM Utilities Sequence statement file
STS.MNU		Cell Status Menu
STSINI.DB		Cell Status Initialization Database
STSMOD.OVR		Cell Status Initialization file
STSMOD.INF		Cell Status Plug-In file
STSRUN.V2		Cell Status Runtime software
TCHTL.SQU	P	Tool Offset teaching software
TOOL.MNU	✓	Tool Offset Menu
UTICON.DAT		AIM Utilities Icon Data file
UTIL.MNU		Station RFD file
UTINI.MNU		AIM Utilities Software Loader Menu File
UTINI.RFD		AIM Utilities Software Loader RFD file
UTINI.DB		AIM Utilities Software Loader Database
UTLINI.DB		AIM Utilities Initialization database
UTMOD.INF		AIM Utilities Plug-In file
UTMOD.OVR		AIM Utilities Initialization Loader File

UTTSK.RFD		Task Configurator RFD file
UTTSK.DB		Task Configurator Database
UTTSK.MNU		Task Configurator Menu
AIM Utilities Plug-Ins		
<u>Quick Change Plug-In</u>		
QCMOD.OVR		Initialization routine for loading Quick Change
QUICKCH.DB		Quick Change database
QUICKCH.MNU		Quick Change menu database
QUICKCH.RFD		Quick Change RFD file
QC_IO.MNU		Quick Change IO display menu
QCINI.DB		Quick Change Initialization database
QCICON.DAT		Icons for Quick Change
QCMOD.INF		Quick Change Plug-in loader file
RUN_CH.SQU		Quick Change squeezed runtime software
RUN_CH.V2		Quick Change commended runtime software
ERRORQC.DB		Quick Change error database
MENU_CH.SQU		Quick Change Squeezed Menu routines
MENU_CH.V2		Quick Change Commented Menu routines
STATQC.DB		Quick change sequence statements
<u>ConnectWare Plug-In</u>		
CNWINI.DB		ConnectWare initialization Database
CNWMOD.INF		ConnectWare Plug-In Loader file
CNWMOD.OVR		ConnectWare Software initialization file

CNWRUN.SQU		ConnectWare runtime software
STATCNW.DB		ConnectWare sequence statements
<u>Event IO Manager</u>		
EVENT.DB		Event Manager database
EVT.MNU		Event Manager menu file
EVT.RFD		Event Manager RFD file
EVTINI.DB		Event Manager Initialization database
EVTMOD.INF		Event Manager Plug-In file
EVTMOD.OVR		Event Manager software initialization file
EVTRUN.V2		Event Manager runtime software
STATEVT.DB		Event Manager Statements
<u>HPGL Graphics Module</u>		
HPGINI.DB		Graphics Module Initialization database
HPGL.MNU		Graphics Module menu database
HPGMOD.INF		Graphics Module Plug-In File
HPGMOD.OVR		Graphics Module Software Initialization file
HPGRUN.V2		Graphics Module runtime software
<u>TCP Server</u>		
TCPMOD.INF		Server Plug-In file
TCPINI.DB		Server Initialization database
TCPMOD.OVR		Server Software initialization programs
TCPRUN.V2		Server Runtime Software
TCP.MNU		Server Menu Database
STATTCP.DB		Server Statements

AIM Utilities Examples		
EX_DISP.MOD		Dispense Module Example Databases
EX_MCP.MOD		MCP Example
EX_PALWR.MOD		PalletWare Example
EX_START.MOD		Startup Example

2.2 Installation Procedure

You can use the V⁺ utility program DISKCOPY to copy the AIM Utilities floppy disks to your hard disk. Complete instructions for using the utility program are contained in the *Instructions for Adept Utility Programs* supplied with your V⁺ reference manuals. The following section details the steps necessary to copy the AIM Utilities files. Copy the AIM Utilities files to the same directory as *AIM MotionWare*.

Note: For AIM versions before AIM 3.2, Do Not install the file *PATH.SQU*. Aim versions before 3.2 will not work with frame relative paths.

1. Turn on your controller.
2. When the start up procedures have completed, place the *Adept Utility Programs* Disk in the A: drive and enter the command. Most newer Adept systems have the utility programs in the \util\ subdirectory on the hard drive. (The hard drive can either be drive C or D depending if an AWC processor is used.)

LOAD A:DISKCOPY or (C:\UTIL\DISKCOPY) or (D:\UTIL\DISKCOPY)
3. When the program files have completed loading, enter the command:

EXECUTE a.diskcopy
4. Remove the utility disk, and insert the AIM Utilities Software disk in the A: drive or if your are running an AWC processor another disk drive will have to be mounted using NFS.
5. From the menu that is presented, select:

4. => Copy multiple files
6. Answer **A** or **NFS>mydrive** (when using an AWC processor) when prompted:

What is the INPUT disk (e.g., A/B/C/D)?
7. Answer **C** or **D** (if using an AWC processor) when prompted:

What is the OUTPUT disk (e.g., A/B/C/D)?
8. Answer **.*** when prompted:

Enter spec of file(s) to copy (blank to exit):
9. Press the **ENTER** key when prompted (a sub-directory name may be specified):

Enter output subdirectory (blank for default):
10. Answer **Y** when prompted:

Do you want existing files automatically superseded (Y/N)?
11. Answer **g** when prompted:

Copy this file...
12. After the last disk has been copied, press the ENTER key when prompted:

Enter spec of file(s) to copy (blank to exit):
13. Select exit from the main menu.

2.3 Verifying Installation

To verify the correct installation of the AIM Utilities Module, follow the steps below.

1. Set the default disk to the AIM Utilities subdirectory and enter the following commands to load the AIM Utilities Module:

LOAD LAIM.V2

COMMAND LAIM
2. AIM should now load and boot up properly. If you have not installed the Dispense Module or PalletWare software, the system will default to loading the MotionWare application software.

After completion of the steps above, the installation is complete. If you experience problems, check that the installation procedures were followed exactly. If problems persist, contact Adept Applications Engineering at the number listed in Chapter 1.

2.4 Installing Utilities to existing Applications

The AIM Utilities software can be installed on existing applications provided any customizations followed Adept recommended procedures. Generally, customizers modify the routines provided in the custom.squ file which is the proper place for customizations. The AIM Utilities package provides a different AIM loader, which means any changes to the program *ai.module.init* will be lost.

2.5 Installing AIM Dispense Module or AIM PalletWare

The Dispense 3.2E+ and PalletWare AIM modules are provided with a software plug-in configuration file that will allow you to load these applications either as a plug-in or by the standard way of using the diskcopy utility. If you are installing both AIM Utilities and either the Dispense Module or PalletWare, we recommend using the standard method as shown in installing the AIM utility package. In both cases, you first install AIM Utilities and then install either the Dispense Module or PalletWare. These packages are loaded last because they will overwrite the software module loader database file so that all of the Dispense or PalletWare software is loaded when starting LAIM.

If you are adding either of the two packages above to an existing application, Adept recommends installing them as a plug-in. This is done by starting AIM, and selecting the Software Loader option from the Setup pulldown. Then select the Install button. You need to have the Dispense or PalletWare disk in a disk drive and select this drive. The Install file will either be *DMMOD.INF* or *PWMOD.INF* for Dispense or PalletWare, By selecting the file to install, all of the required files will be loaded.

2.6 Selecting Default AIM Modules

The Software Loader option in AIM Utilities allows the user to select one of the following AIM Applications to load:

- PCB Module
- MotionWare
- VisionWare
- Dispense Module
- PalletWare

Once selecting one of these options, the software will select the proper software loading and task configurations for this module. The user now only needs to shutdown AIM by either selecting from the Software Module loader or from the "Special" AIM pulldown. Then restart the system by executing the following line from the V+ monitor.

Execute 1 a.aim

2.7 AUTO File Installation

Provided with the AIM Utilities software package is a file on the AIM Utilities Software disk called AUTO.V2. This file can be copied to the root directory so that when the system is booted, AIM will load and start automatically. The Adept controller must be configured so that the Auto file will load and execute automatically. This can be accomplished by setting dip switches on the SIO board if an Adept MV controller is purchased without the AWC controller. If the system has an AWC processor, then the auto startup would be configured in the Config_C program. Please refer to the *Adept MV Controller User's Guide* for more details.

The AUTO.V2 file can be copied by typing in the following V+ Monitor instruction.

```
FCOPY C:\AUTO.V2 = A:AUTO.V2
```

This instruction assumes the AIM UTILITIES Software disk is in drive A. If using an AWC processor NFS will have to be used instead of the A drive.

AIM Utilities Menus

3

3.1 Software Module Loading	29
3.2 Software Loader Record Menu	31
3.3 Task Configurator	33
Task Configurator Record Menu	35
Task Priorities Display Page	37
3.4 Software Plug-In Utilities	38
Uninstall Software Module	39
Create the Software Module Install File	41
3.5 Operator Control Panels	43
3.6 Precision Point Database	44
3.7 MCP Emulator	46
3.8 NFS Disk Drive Mounting Utilities	48
NFS Mount Status Menu Page	49
3.9 Tool Database	50
3.10 IO Status Display Page	52
3.11 Cell Status Display Page	53
3.12 MCP Teach Menu Changes	54

AIM Utilities Menu Pages

This chapter describes the menu pages used by the AIM Utilities software. Several new menu pages are added to the AIM software package when AIM Utilities software is installed. These include menus for software and task initialization of AIM, Operator control panel, I/O menu, NFS mounting utilities, Tool Offset Teaching and others.

This chapter will provide a general overview of the menus and describe the features by a graphic numeric bubble with text following the figure.

3.1 Software Module Loading

The Software Module Loading database is provided to allow the user to load custom software modules or other software files into the AIM system. This provides the user an easy method to load new software into the AIM system without customization. Please note that the order of which the software is loaded is based on the position in the database. In some cases, software must be loaded in a correct order for proper functionality.

To review or edit the current software being loaded select the Setup pulldown and the Software Module loader option.

Setup ➡ Software Loader

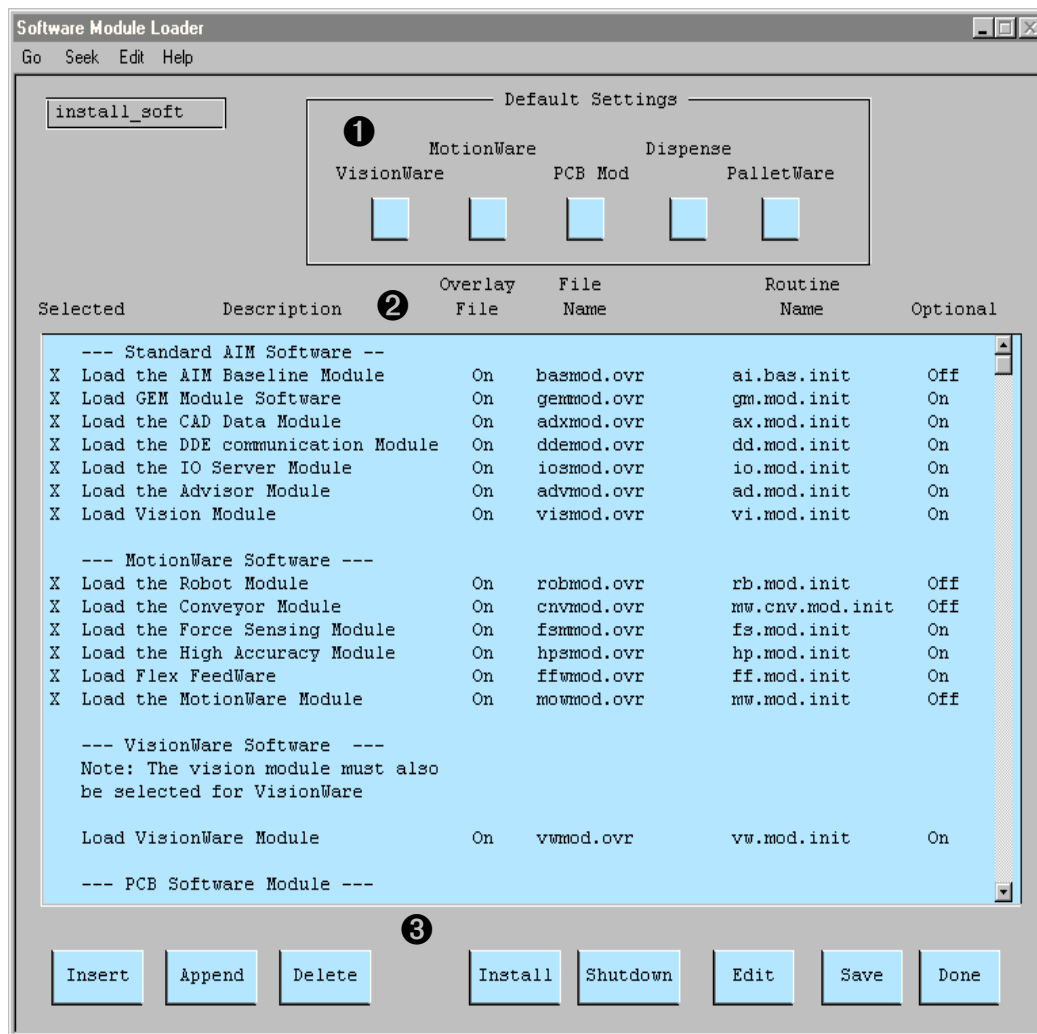


Figure 3-1 Software Module Loader

① Default AIM Software Setting:

This allows the user to load any of the current Adept Software modules as long as they have been loaded into the system. The Task Configuration database will be modified after the selection. After selection of the software you must restart AIM for the software to be reconfigured. Be sure to save the databases before restarting AIM.

VisionWare Selection - Pressing on this button will select the standard VisionWare software modules to load as well as the task selections. The Task Configurator database will be changed so that the standard VisionWare tasks will be selected and running the next time AIM is restarted.

MotionWare Selection - Pressing on this button will select the standard MotionWare software modules to load as well as the task selections.

PCB Software Selection - Pressing on this button will select the standard PCB software modules to load as well as the task selections.

Dispense Selection - Pressing on this button will select the standard Dispense module software modules to load as well as the task configurations.

PalletWare Selection - Pressing on this button will select the standard PalletWare software modules to load as well as the task configuration.

② Software Modules Display:

This display area shows what is currently selected by the system. The menu displays an **'X'** if the software module is selected to load during startup. It displays **On** if the software loaded is an overlay file and displays **Off** under the Optional heading if an error message will occur during loading if the module cannot be loaded.

If you select a row on this display page and double click you will be able to edit the software loader record. Please refer to the Software Loader Record Menu on page 31 for more information on editing records.

③ Editing Button Area - These buttons allow you to perform functions on the software loader database.

Insert Button - This button allows you to insert a record before the highlighted area on the display page. This is useful because many software modules require other modules to be loaded before they will load properly.

Append Button - This button will create a new record at the end of the database. After selection the record will appear.

Delete Button - This button will delete the currently highlighted record on the display page.

Install Button - This button will pop-up another display page to allow you to install software modules.

Shutdown Button - This button will shutdown the AIM software package. After selection of new software modules it is required to restart the AIM system for the software modules to load.

Edit Button - This button will pop-up the highlighted record on the display page to allow changes to be made. Please refer to the Software Loader Record Menu on page 31 for more information on editing records.

Save Button - This button will save the current changes to the Software Loader database.

Done Button - This button will exit the current menu display page.

3.2 Software Loader Record Menu

The software loader record page allows you to edit the software modules to be loaded.

To review or edit a record, highlight and double click on the record from the display page you wish to edit. Additionally, you can highlight a record and press the edit button.

Load Software Module

install_soft 18-Feb-99 14:54

Segment 37 Description of Task ①

Load AIM Utilities software package. This software includes the following:
 Paths relative to Frames, Operator Panels, Software Module and Task Loader
 MCP location programs, Tool Offset Teaching, 30 + task statements for other operations

② On to Load software during startup
 Off Disables software loading ④

② On if loading a overlay file File Name to Load
 Off if loading a V+ file ⑤ utmod.ovr

③ On if file is optional Routine Name
 Off displays a error if failure ut.mod.init

⑥ Append Prev Next Done

Software Files ⑦

Be sure to include the name of the Plug-In (INF).
 This will be used during copying operations.

laim.v2	mcw.mnu	
utini.db	utmod.ovr	
uttsk.db	tool.mnu	
menu_ut.squ	util.mnu	
uttsk.mnu	plgmod.ovr	
utini.mnu	infmod.mnu	
tchtl.squ	utlini.db	
path.squ	plgmain.mnu	
uticon.dat	uttsk.rfd	
statut.db	utmod.inf	
infmod.ovr	utini.rfd	
aimutil.v2	plgmain.adv	

Figure 3-2 Software Loader Record

- ① The description of task is displayed on the main software loader menu page. This is provided so you can add comments or a description of the operation.
- ② The On/Off selection of loading an overlay file performs the following operations.

(ON) - The On state will load an AIM Overlay file and execute the specified program based on the supplied fields. After the program has been executed, the overlay software will be deleted from memory.

(OFF) - The Off state will load a V+ file if the specified routine does not exist in memory. The software will remain in memory after completion.

- ③ The On/Off selection of the optional file performs the following operations. If selected, AIM will try to load the software module, if the file does not load no error condition will result and AIM will continue loading the system. If this button is not selected AIM will try to load the software, if there are problems, AIM will display an error message.
- ④ The On/Off selection of loading the software will try to load the software module if selected. If turned off AIM will not try to load the software. This allows a record of the available software, without trying to load the software.
- ⑤ The File Name field allows the user to enter in the name of the software file to load.

If the user should enter in **None**, the field description will be displayed on the main menu page without the rest of the data on the line. This is useful for adding comments to the main menu page. Be sure that the file is not selected to load.

The Routine Name field is provided for overlay files and for quicker loading of V+ files. If the overlay file is selected, this routine will be executed and deleted when complete. If the overlay file is not selected, this routine name is used to check to see if the software is already loaded.

⑥ The Function Button Area

Append Button - This button will allow you to add a new record to the end of the software module loader database. This record will appear after the append button is pressed.

Prev Button - This button allows the user to move between records in the database. This button will allow you to move to the previously displayed record as long as one exists.

Next Button - This button will allow the user to move to the next record in the database.

Done Button - This button will exit the current menu and return to the main menu.

- ⑦ The Software Files selection is a list of all of the files contained in the software module. When creating a software install file this list is placed in the file. During the Install, Update and Uninstall operations this list is used for transporting or deleting files.

3.3 Task Configurator

The task configurator database is provided to allow the user to add more user tasks into the system. This provides the user an easy method to configure tasks into the AIM system without customization.

To review or edit the current software being loaded select the Setup pulldown and the Task Configurator option.

Setup ➡ Task Configurator

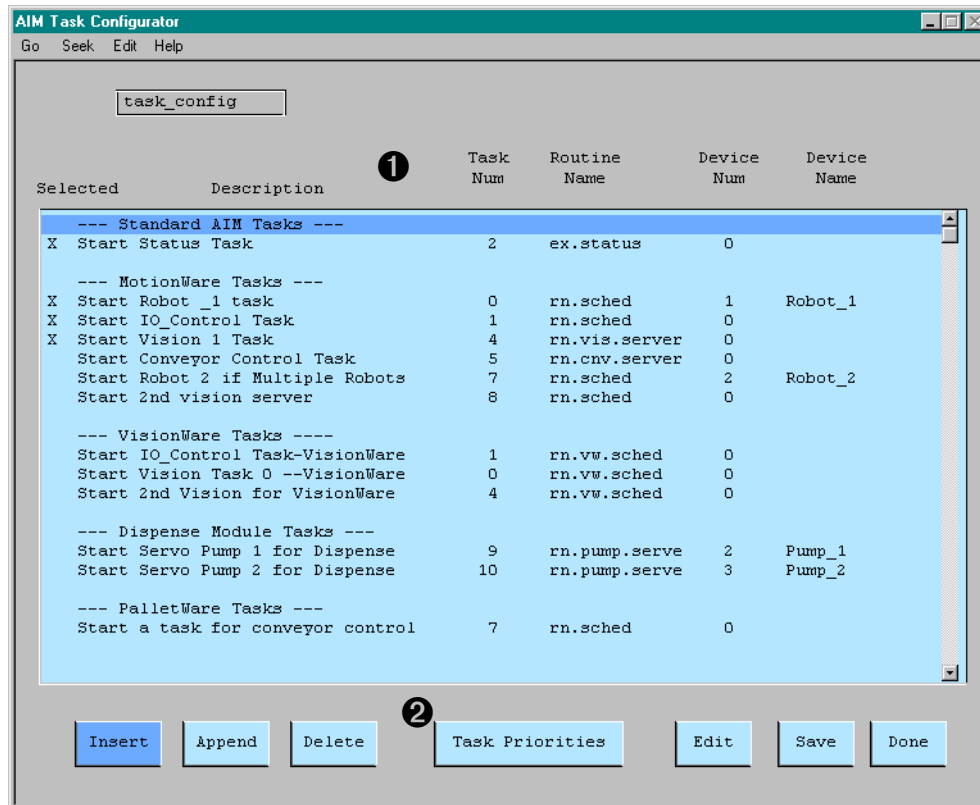


Figure 3-3 Task Configurator Main Menu

① Main Menu Display Area

Selected - The selected area indicates if this task is to be started during AIM bootup.

Description - This area displays the text description specified in the task record.

Task Num - This area displays the task number that is selected.

Routine Name - This area displays the routine, server or scheduler running in this task.

Device Number - This area displays the robot device that is running in this task. There will be no display if the task is not running a robot.

Device Name - This area displays the robot device name running in this task.

② Button Function Area:

Insert Button - This button allows you to insert a record before the highlighted area on the display page.

Append Button - This button will create a new record at the end of the database. After selection the record will appear.

Delete Button - This button will delete the currently highlighted record on the display page.

Task Priorities Button - This button will pop-up a menu display that shows the selected tasks in a list with all of the task priorities. Refer to *Task Priorities Display Page on page 37* for more details on this menu.

Edit Button - This button will pop-up the highlighted record on the display page to allow changes to be made. Please refer to the Task Configurator Record Menu on page 35 for more information on editing records.

Save Button - This button will save the current changes to the Software Loader database.

Done Button - This button will exit the current menu display page.

Task Configurator Record Menu

The task configurator record page allows you to edit the user tasks to be executed.

To review or edit a record, highlight and double click on the record from the display page you wish to edit. Additionally, you can highlight a record and press the edit button.

The screenshot shows the 'Task Configuration' window with the following elements:

- task_config** (text field)
- 11-Aug-98 07:38** (text field)
- Segment 7** (text field)
- Start Task** (checkbox, checked)
- Description of Task** (text field containing 'Start Conveyor Control Task')
- Scheduler Select** (radio button, selected)
- Routine Name to Run** (text field containing 'rn.cnv.server')
- Task #** (text field containing '5')
- Task Name** (text field containing 'Conveyor')
- Task Variable Name** (text field containing 'ti.cnv')
- Task Priorities** (table with 16 columns, values: 0, 0, 0, 0, 0, 0, 0, 22, 22, 0, 0, 0, 0, 0, 0, 0)
- Select Robot** (radio button, selected)
- Check Option** (text field containing 'ai.opt.convey')
- Append** (button)
- Prev** (button)
- Next** (button)
- Done** (button)

Figure 3-4 Task Configurator Record Menu

- 1 The description of the task is displayed on the main software loader menu page. This is provided so you can add comments or a description of the operation.

Start Task - This check box will allow AIM to start this task during boot up of the AIM system. Many tasks are specified in this database to perform different tasks depending on what AIM modules are selected.

2 Scheduler Selection

On/ Off button to select whether the task is running a scheduler or a server. The task is run differently based on this selection. If specified as a scheduler a sequence will be executed in the task when it is running. If a server is selected a V+ routine will run in this task and not be linked with a sequence.

The routine name to run specifies the routine that is running in the task. During startup AIM will execute this routine name in the specified task.

③ **Task Specification**

Task # - This selection specifies the task number your sequence or scheduler will run in.

Task Name - This field requires the user to enter a name for the task to be displayed in the AIM menus. This field is required.

Task Variable Name - This entry field will define a V+ variable which will be set to the task number if the task is running. This is used with most servers, but also can be used with other V+ routines. This field is optional.

④ **Task Priorities** - This allows the user to specify the priorities used by V+ when running software. V+ has 16 different time slice periods during one V+ tick. For more information please see the *V+ Operating System User's Guide*.

⑤ **Robot Selections**

Select Robot - This radio button will specify that a robot or servo device is going to run in this task. After selection of this button, other menu fields will appear to allow you to specify other needed information.

Robot # - This entry field specifies the robot number to associate with this task. This field is a required field.

Robot Name - This entry field specifies the name of the robot which will be used by AIM. This field is required if a robot is selected.

⑥ **Check Option** - This entry field allows the user to check if AIM options have been loaded before allowing a task to start during the AIM boot up.

Append - Allows you to add another record to the database without going back to the main menu.

Prev - Allows the user to move to previous records in this database.

Next - Allows the user to move to the following records in this database.

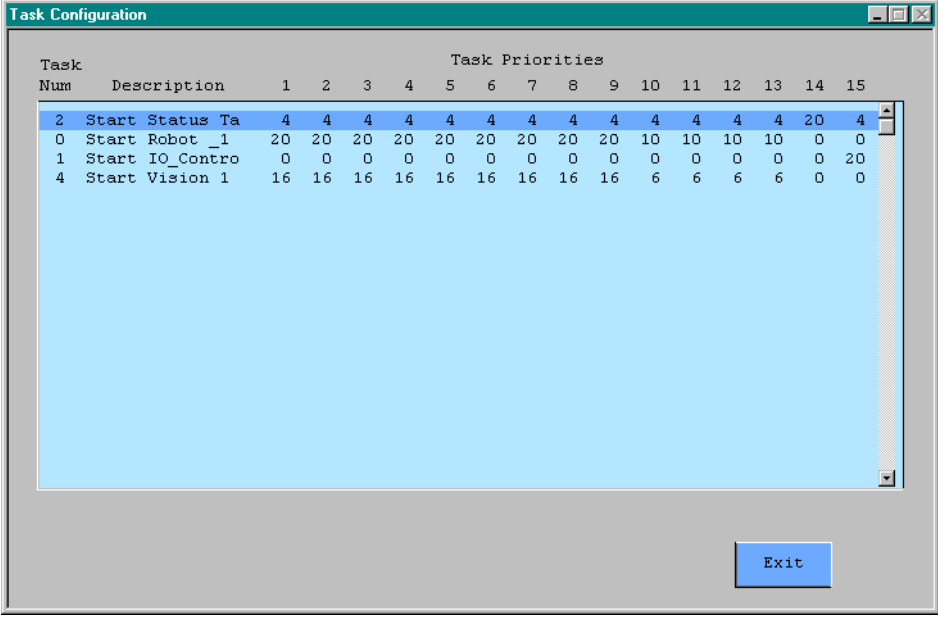
Done - Allows the user to exit this menu.

Task Priorities Display Page

The Task Priorities Display Page allows the user to view all of the tasks and their priorities and edit the priority values. This allows an easy way to compare what has already been setup in other tasks. The menu is placed on the monitor so the *Task Profiler* can be viewed while the menu is being displayed.

To get to this menu page simply select the Task Priorities button on the Task Configurator menu page. See Task Configurator on page 33 for more details.

The user can edit the task priorities by double clicking on the selected task on this menu.



Task Num	Description	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	Start Status Ta	4	4	4	4	4	4	4	4	4	4	4	4	4	20	4
0	Start Robot _1	20	20	20	20	20	20	20	20	20	10	10	10	10	0	0
1	Start IO_Contro	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
4	Start Vision 1	16	16	16	16	16	16	16	16	16	6	6	6	6	0	0

Figure 3-5 Task Priorities Display

3.4 Software Plug-In Utilities

The Aim Utilities package contains software to allow the user to create their own software module install file for easy installation into other systems as well as an easy upgrade path into future revisions of AIM. This utility software allows you to update software module backups, create a software module and uninstall an existing software module.

To get to the software plug-in utilities select the Utilities pulldown and the software install utilities option.

Utilities ➡ Software Plug-In Utilities

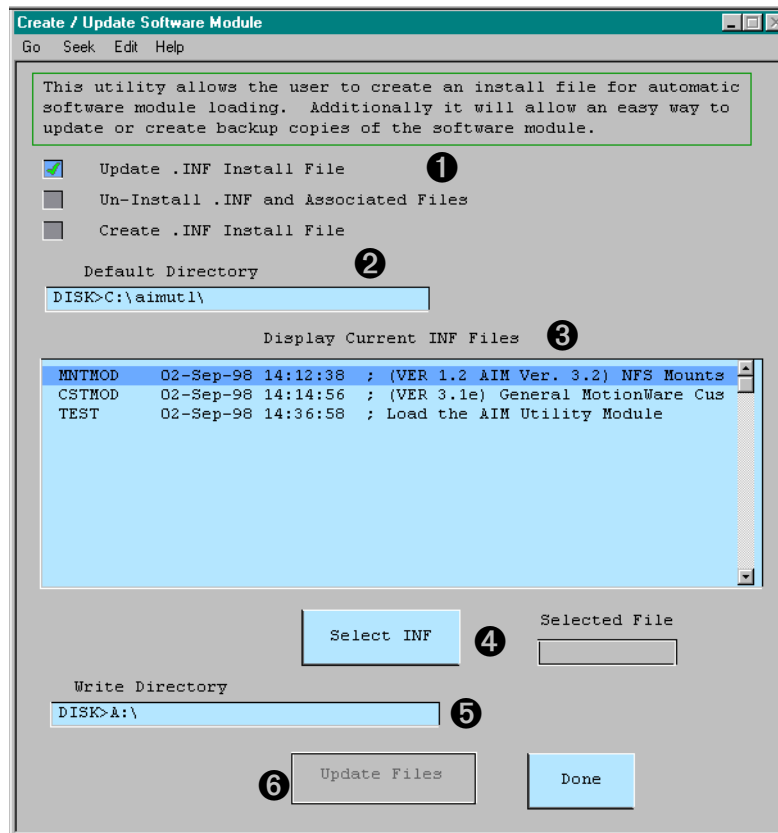


Figure 3-6
Update Software Module

- 1 Update INF Install File**
This selection will copy the files listed in the install (INF) file to the directory selected as the write directory. This allows the user to either overwrite and update a backup disk or create a copy of the module on a disk.
- 2 Default Directory**
This is the disk directory to search for the install (INF) files.
- 3 Display Current INF Files**
This is a pick list for selecting the software module you wish to update.

- ④ **Select INF**
This allows you to select the current highlighted file. This can also be done by double clicking with the pointing device.
- ⑤ **Write Directory**
This is the directory where the software module will be copied to.
- ⑥ **Update Button**
This button will begin the update operation.

Uninstall Software Module

The Software Plug-in Utilities allow the user to uninstall a software module that has been loaded previously. This can be done to reduce the number of task statements or the memory used by the software module in the system. This operation will delete the software module on the main directory as well as in the software and task loader databases.

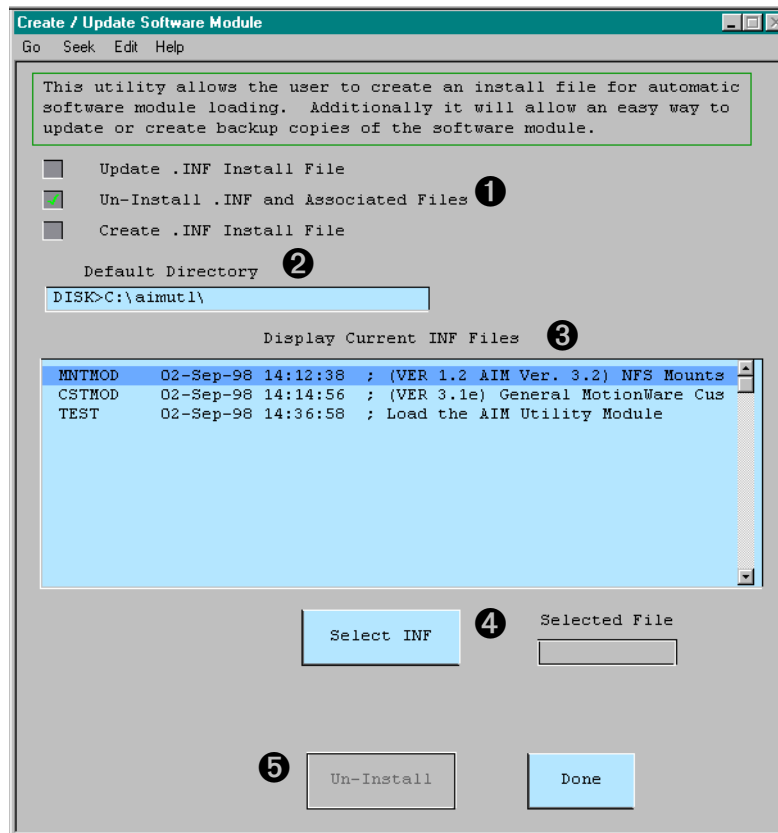


Figure 3-7 Uninstall Software Module

- ① **Uninstall Software module**
This selection will delete the files listed in the install (INF) file in the directory selected as the default directory. This allows the user to delete all the software files as well as the records defined in the software loader and task configurator databases.

② Default Directory

This is the disk directory to search for the install (INF) files and delete from.

③ Display Current INF Files

This is a pick list for selecting the software plug-in you wish to delete.

④ Select INF

This allows you to select the current highlighted file. This can also be done by double clicking with the pointing device.

⑤ Uninstall Button

Pressing this button will begin the uninstall process.

Create the Software Module Install File

This option allows the user to create an install plug-in file for easy loading of the customizations that the user may have made. This creates a file with the INF extension for easy listing of all software modules. A total of 3 software loader records and 3 task configuration records may be used for the software module. This allows the user more flexibility in the creation of their module. The four lines of description written in the first selected software loader records will be used in the description of the software INF file when displayed during the install process. The 20 fields available for software files are placed in this file for the operations listed in this section of the manual. The menus used for this operation are described below.

The screenshot shows a window titled "Create / Update Software Module" with a menu bar (Go, Seek, Edit, Help). Inside the window, there is a text box with the following text: "This utility allows the user to create an install file for automatic software module loading. Additionally it will allow an easy way to update or create backup copies of the software module." Below this text box are three radio buttons: "Update .INF Install File", "Un-Install .INF and Associated Files", and "Create .INF Install File" (which is selected and marked with a circled 1). Below the radio buttons are two sections: "Select Software Loader Record" and "Select Task Configurator Records". Each section has a "Select" button (marked with a circled 2 and 3 respectively) and three empty text input fields. Below these sections are two more input fields: "INF File Name" (marked with a circled 4) and "Write Directory" (marked with a circled 5), which contains the text "DISK>A:\". At the bottom, there are two buttons: "Create Files" (marked with a circled 6) and "Done".

Figure 3-8 Create Software INF File

① Create Software module

This selection will create the INF file specified by the entry data on this page.

② Select Software Loader Record Button

By pressing this button the Software Loader menu will pop-up to allow the user to select which records will be used with the software module. It also allows the user to make changes to the records to either add more software files or more descriptions which will be added to the INF file.

③ Select Task Configurator Record Button

By pressing this button the Task Configurator menu will pop-up to allow the user to select which records will be used with the software module. It also allows the user to make changes to the records before the software module is created.

④ Name for the INF File

This name will be used for the file that will be created. This name should be less than 8 letters and begin with an Alpha character. The software will check the name entered and reduce the size or delete a numeric character if one is entered.

⑤ Write Directory

This field allows the user to select the directory the file will be created in.

⑥ Create Button

This button will begin the creation process for the file. The process is complete when the create button is not highlighted anymore.

3.5 Operator Control Panels

The AIM Utilities package contains three different operator control panels. The dispense module, palletware and a standard operator control panel are provided. Depending on the software options that are installed, the AIM Utilities loader will select the operator control panel to install.

To display the operator menu select the Execute pulldown and the Operator option.

Execute ➡ Operator

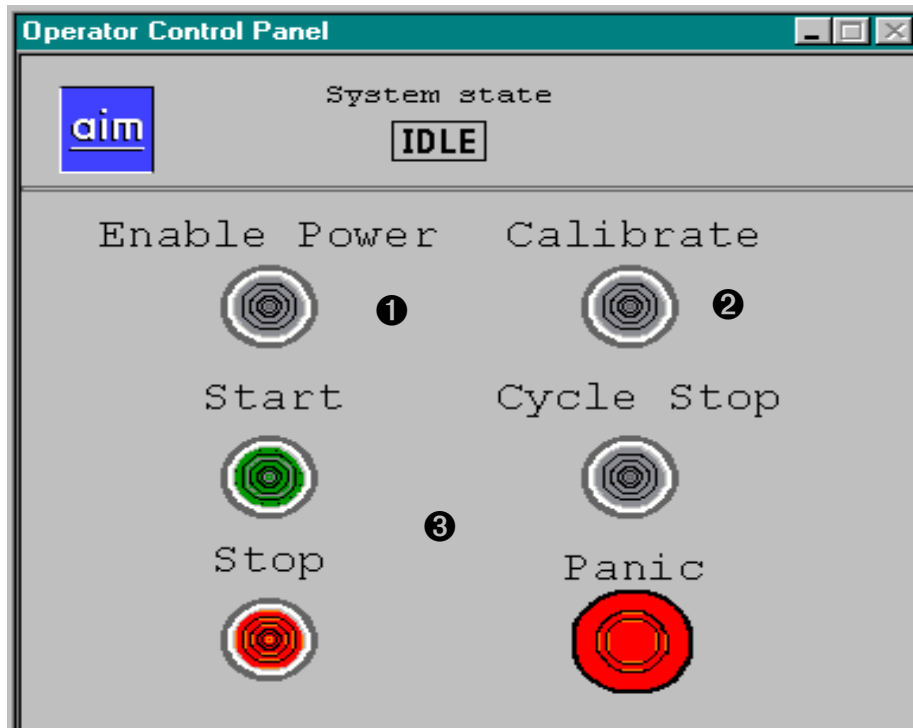


Figure 3-9 Standard Operator Control Panel

- ❶ The **Enable Power** button will try to turn power on to the robot system. Some systems may require the user to press an external button to enable. (This is in the case of a CE certified Cat 3 robot) If an error occurs during this operation an error message will be displayed.
- ❷ The **Calibrate** robot will display the current status of the robot calibration as well as allowing the user to calibrate the robot.
- ❸ The **Start** button will execute the standard control sequence called start. This has the same function as in the Master Control Panel.

The **Stop** button executes the stop sequence. This will generally cause a pause state to the sequence.

The **Cycle Stop** button executes the cycle stop sequence.

The **Panic Button** will halt the robot motion and disable power to the system.

3.6 Precision Point Database

The AIM Utilities package contains software to allow motion to a joint defined position rather than a world coordinate defined position. This allows the user to work better with positions that require a specified arm joint position. This is very useful with jointed arm applications. This will also be useful with applications that require full rotation of Adept arms with Joint 4.

To display the Precision Point menu select the Edit pulldown and the Precision Point option.

Edit ➔ Precision Point

The screenshot shows the 'prec.point *Global*' window. At the top, there's a menu bar with 'Go', 'Seek', 'Edit', and 'Help'. Below the menu bar, there's a status bar showing '1 of 1' and 'Device: Robot_1'. The main area contains several input fields and buttons. A red circle with the number 1 points to the 'first.point' field. A red circle with the number 2 points to the 'Here' button. A red circle with the number 3 points to the 'Motion' button. A red circle with the number 4 points to the 'Align' button. A red circle with the number 5 points to the 'Strategy' field. The window also displays joint positions for JT1 through JT6, a 'Device' field set to 'Robot_1', a date/time field, and sections for 'Input Signals' and 'Output Signals'.

Figure 3-10Precision Point Menu

- ① Record Name field requires a name for the jointed position.

Device: Requires the proper robot to be selected for the position to be defined.

JT1-JT6 are six fields that define the position of most robot arms supported by the Adept controller. If the robot is only a 4 axis arm then only 4 fields will be used for the position. These fields can be edited by the user or taught by using the Here or Teach buttons.

- ② The **HERE** button will teach the current location in the six JT fields provided.

The **TEACH** button will allow the user to teach the current position via Adept's manual control pendant. A pendant display will appear with selections and the speed pots can be used to move the robot to the current taught location if needed.

③ Motion Parameters Section

Line / Joint radio button selection allows the user to define a straight line or joint interpolated motion to the defined position.

Speed selection is a percentage of maximum speed. This parameter is not limited to 100%. This value is dependent on the type of robot being used in the application.

Coarse / Fine nulling tolerance selection: This determines how close the robot must be to the taught location before the next position or process is executed. Fine is typically ± 0.001 inches with Adept robots.

Accel / Decel specification: These parameters control the acceleration and deceleration during the robot motion to the specified location.

Prof specification: This is the acceleration and deceleration profile during a motion. This is based on the robot that is running. There are several preprogrammed profiles that will allow S curve acceleration.

④ **Align** Feature: This feature will square up the wrist joints on a jointed arm (5 or 6 axis arms). This is a good aid with robot teaching.

⑤ Strategy Parameter Section

Strategy Routine: This is a V+ routine that will be called once the robot reaches the specified position. This routine will handle the specified inputs and outputs as well as the settling and dwell delays that can be specified.

Retry Count: This allows the user to specify a number of retries if a failure occurs. In this case the parameter is not used.

Settling Time: This parameter allows a period of time after the robot reaches the position before any outputs occur.

Dwell Time: This parameter allows the user to specify a period of time to dwell after the outputs have been asserted before allowing the process to continue.

3.7 MCP Emulator

Provided with the AIM Utilities software is a utility to allow the user to move the robot without the use of the MCP. This menu has 2 different modes; Step and Jog mode. The keyboard arrow keys are provided as hot keys to allow easy stepping of the X/Y axis. Many default parameters are provided in a initialization database. To view the MCP Emulator menu pages, select the Show pulldown and the MCP Emulator option.

Show ➡ MCP Emulator

The figure below is the MCP Emulator in Step Mode.

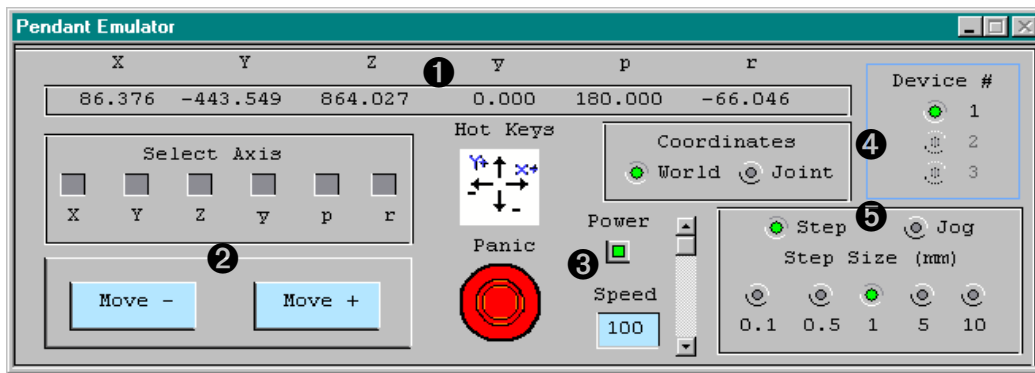


Figure 3-11 MCP Emulator (Step Mode)

① Robot Position Display:

This area displays the current location of the robot. This display changes based on the coordinate selection. If Joint Coordinates is selected then joint positions will be displayed.

② Select Axis:

The select axis area allows the user to select the robot axis that will move when the move buttons are pressed. The axis selections will change based on the Coordinate selection. If joint is selected then the joint numbers will be selected.

Move Buttons:

The Move +/- buttons will move the robot in the positive or negative direction.

③ Hot Keys:

When the arrow icons appears in a bold state the Hot Keys are active. This will allow the user to step the robot by pressing the keyboard arrow keys. Hot Keys are only active in the step mode function.

Panic Button:

Pressing the Panic button will immediately stop the robot and disable robot power.

Power Button:

The power push button will enable or disable power depending on the current state. Power can only be turned on if all error conditions have been cleared.

Speed:

The speed field and scroll bar allow the user to adjust the speed of the motion. This speed is based on the default values specified in the initialization database.

④ Coordinates:

The MCP Emulator software allows operation in world or joint coordinates. This allows the user to move the axis along the axis or in a individual joint fashion.

Device:

The MCP Emulator software allows a maximum of 3 robots to be moved by this menu package. The user selects the desired robot by pressing on the radio button.

⑤ Motion Mode:

The MCP Emulator software allows motion in either a Step or Jog fashion. The radio button allows the user to select the operation. In Step mode, the robot will move the selected distance every move button press or Hot Key press. In Jog mode, the robot will move continuously while the Jog +/- button is pressed.

Step Distance:

The desired step distance is selected by pressing the radio button by the distance amount. Units are specified in millimeters. These buttons are only active when the software is in Step Mode.

3.8 NFS Disk Drive Mounting Utilities

Provided with the AIM Utilities software is a utility to allow the user to mount other disk drive devices thru a network. Before this software will function the user must install a NFS server on the other computer system that contains the drives. Additionally, Ethernet must function properly between both systems. To view the mount menu pages, select the Setup pulldown and the NFS Mount option.

Setup ➔ NFS Mount

The figure below is the menu that allows the mounting of other disk drives thru NFS.

Mount a new NFS Volume

Example

Server Name: Server1
 Server Address: 192 168 144 101
 Mount Name: mydrive
 Mount Path: /c

Note: The server name is defined by the user and can be anything. The server address is the IP address of the system that is the NFS server. The mount name is also user defined and can be anything. The mount path is the path to the desired NFS shared drive on the NFS server.

Server Name: ①
 Server Address: ② ☐ Report Error
 Mount Name: ③
 Mount Path: ④

⑤ Version 1.2

Figure 3-12 NFS Mount Menu Page

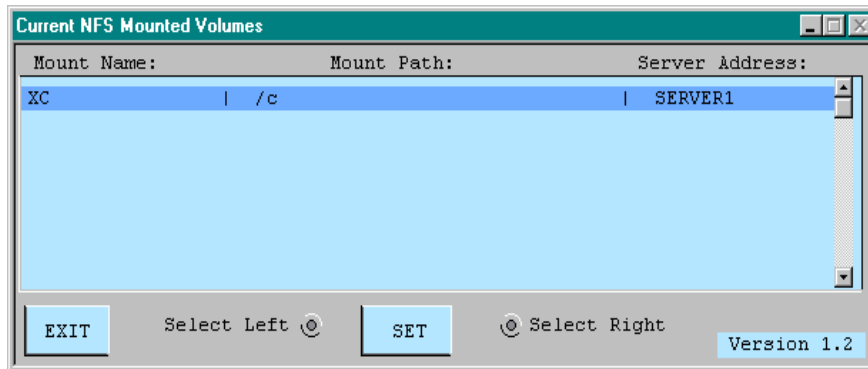
- ① **Server Name:**
This field will label the name of the computer that will be displayed to associate that computer in other menu displays.
- ② **Server Address:**
This is the IP ethernet address used by the server computer.
- ③ **Mount Name:**
The mount name is used by the user to specify the disk drive on the server computer. This label will be used by the user during any file operations.
- ④ **Mount Path:**
This path is the name of the Server Drive that will be used. Additionally, you can define a path on that drive to a specified directory or folder.
- ⑤ **Mount Button**
This button will perform the drive attachment based on the data specified.

NFS Mount Status Menu Page

Provided with the AIM Utilities software package is a status display page of the disk drives that are attached to the system via NFS software mounts. To display this menu page look under the SHOW pulldown menu and select the Current NFS Mounts Menu page.

SHOW ➡ Current NFS Mounts

The figure below is the menu page that displays the current drives available.



3.9 Tool Database

The tool offset database menu page has been modified to include a tool teaching utility that runs from the Adept manual control pendant. This utility allows the user to rotate the desired tool center about a common point. This rotation will allow the software to determine what the tool center is.

To create a new tool record:

Edit ➔ Tool ➔ *press F2* ➔ *enter record name*

To edit an existing tool record:

Edit ➔ Tool ➔ Seek ➔ Index ➔ *dbl clk on tool record*

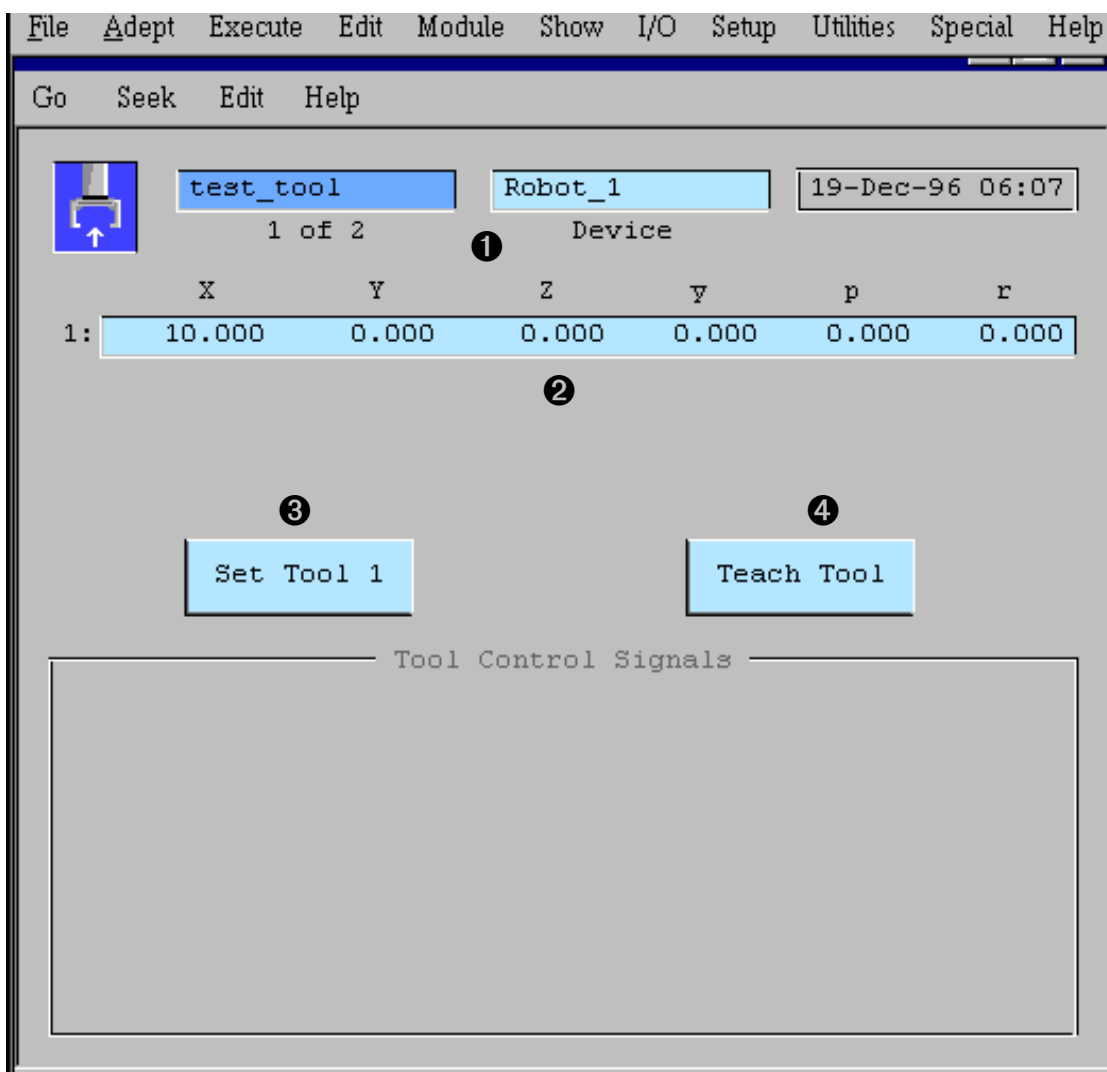


Figure 3-13 Tool Offset Menu

- ❶ Enter a name for this tool record. The numbers below the name field indicate the number of this record in the database and the total number of records in the database.

Select the proper robot device number for this tool offset. Double click on this field to select a different device.

Shows the date this record was last modified.

- ❷ Shows the offset (transformation) that will be applied to any location the robot moves to while this tool is in effect. These values can be entered directly (based on the engineering data for the offset tooling), or taught using the teach routine (option ❹).

- ❸ Press to use the values in the current record as the robot tool transformation.

- ❹ Press to use the manual control pendant to define a tool transformation. The tool offset teaching utility is described in detail in Section 4.5.

To test a tool transformation, Press SET TOOL button to invoke the tool offset, then select TOOL state from the manual control pendant and choose **RZ** from the mode control buttons. Move the robot using one of the speed bars. The robot should rotate about the axis of the tool.

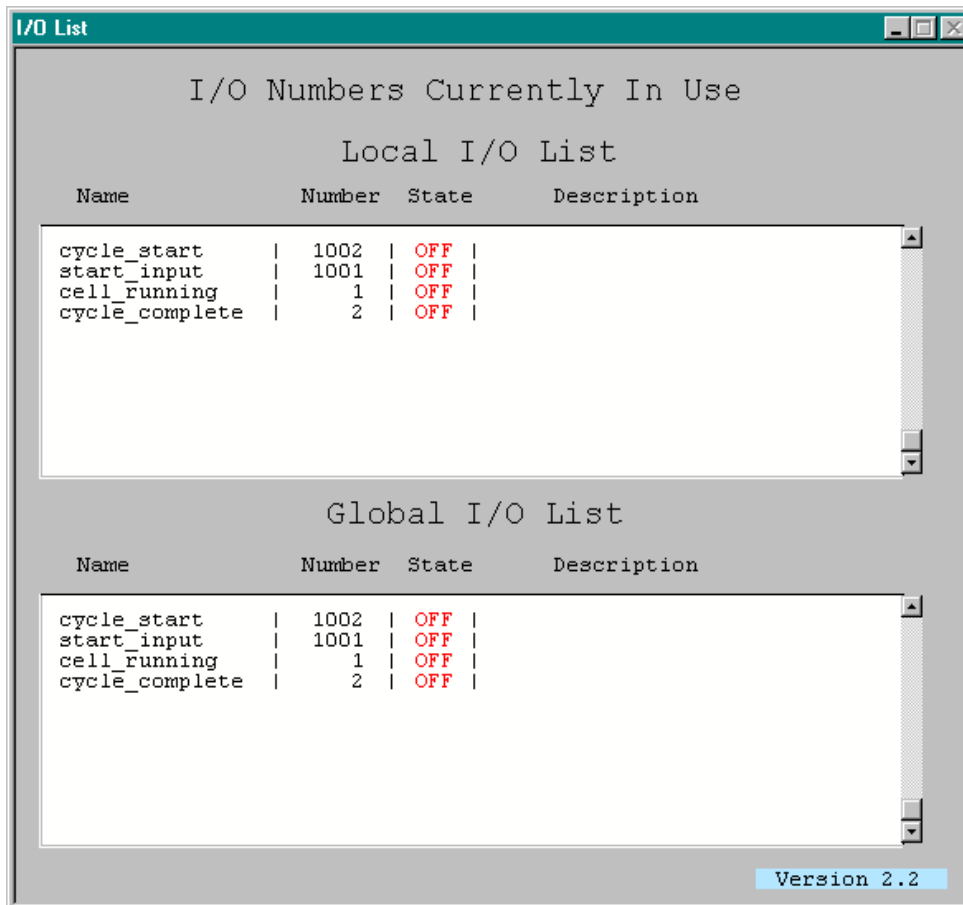
More documentation is available on tool offsets in the *MotionWare Users Guide*.

3.10 IO Status Display Page

Provided with the AIM Utilities software is a utility to display the current IO status based on the names defined in the variable database. This display shows both the global and the local database modules drive devices thru a network.

Show ➡ IO List

The figure below is the menu display of the variable names and input and output values for those IO's.



The screenshot shows a window titled "I/O List" with a teal header bar. Inside, the title "I/O Numbers Currently In Use" is centered. Below it, the "Local I/O List" section contains a table with four columns: Name, Number, State, and Description. The table lists four items: cycle_start (1002, OFF), start_input (1001, OFF), cell_running (1, OFF), and cycle_complete (2, OFF). The "Global I/O List" section below it contains an identical table. The "Version 2.2" label is in the bottom right corner.

Name	Number	State	Description
cycle_start	1002	OFF	
start_input	1001	OFF	
cell_running	1	OFF	
cycle_complete	2	OFF	

Name	Number	State	Description
cycle_start	1002	OFF	
start_input	1001	OFF	
cell_running	1	OFF	
cycle_complete	2	OFF	

Version 2.2

3.11 Cell Status Display Page

Provided with the AIM Utilities software is a menu screen that shows the current system status. This status menu runs in task 6 where the typical AIM status screen is shown. There is an initialization database provided for the text that appears next to the IO displays on the menu. The initialization database is called *stsin.db* under the initialization database selection in the pulldown menu called *SETUP*.

This screen shows the IO state and allows the outputs to be set per the display. Additionally, other system information is shown on this screen. This menu can be accessed by using the *Show* pulldown and selecting *Cell Status*.

Show ➡ Cell Status

The figure below is the Cell Status Menu.

INPUTs		OUTPUTs		STATUS	
<input type="checkbox"/> (1001) Change these	<input type="checkbox"/> (3001) Gripper Open	Cal	Not Calibrated		
<input type="checkbox"/> (1002) descriptions	<input type="checkbox"/> (3002) Gripper Close	Power	Disabled		
<input type="checkbox"/> (1003) and IO values	<input type="checkbox"/> (3003) Spare	Meter	823		
<input type="checkbox"/> (1004) in the STSINI	<input type="checkbox"/> (3004) Spare	Estop	Panic Pushed		
<input type="checkbox"/> (1005) database.	<input type="checkbox"/> (0001) SIO Output	Pndnt	Automatic Mode		
<input type="checkbox"/> (1006) None	<input type="checkbox"/> (0002) SIO Output				
User: System_startup		POS: % Complete 100		Robot: 862-106	
14-Sep-98 10:51:15		1066.800 0.000 876.300		System: 3404-285	
Status		ERR: Not complete		OS Ver: 12.2 Edit C1	

Figure 3-14 Cell Status Menu

3.12 MCP Teach Menu Changes

The AIM Utilities package has changed the MCP menus that are used by both the Location and Precision Point databases. New MCP menu selections are **Prv Rec** and **Nxt Rec**. These buttons allow the user to select other records in the database from the teach pendant. The **TOG_SEL** allows the user to toggle thru the Approach, Location and Depart teaching when using the locations database.

This menu appears on the teach pendant when the **TEACH** button is pressed when editing a location or precision point record. The other buttons on the pendant remain the same as standard AIM software.

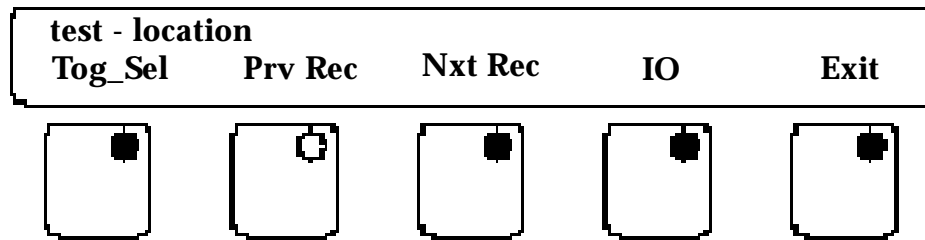


Figure 3-15 MCP Teach Buttons

Using AIM Utilities

4

4.1 AIM Utilities Statements	56
AIO	56
ARM_CONFIG	56
ATTACH_ROBOT	56
CALIBRATE	57
CLEAR_MESSAGE	57
DETACH	57
DISTANCE_FROM	57
GET_BITS	58
GET_PARM	58
GET_STATE	59
GET_SWITCH	59
GET_TIME	60
J4_INERTIA	61
MCP_CHECK	61
MCP_WRITE	61
MOUNT_NFS	62
MOVE_UT	63
OPEN_MENU	64
PMOVE	64
RETRACT_Z	64
RETURN_LOC	65
RETURN_JTS	65
SET_BITS	65
SET_OPRMODES	66
SET_PARM	66
SET_PATH	66
SET_SWITCH	67
SHIFT_MOVE	67
STATUS_MESSAGE	68
TASK_MODE	68
WAIT_FOR_IO	69
TYPE_MON	69
4.2 Calibration from Sequences	70
4.3 Using the Message Statements	71
4.4 MCP example	71
4.5 Tool Offset Teaching	72
Testing the Tool Offset	74
4.6 Paths Relative to Frames	75
4.7 Precision Point Module	75
4.8 Software Plug-Ins	75
4.9 Example Sequences	76

4.1 AIM Utilities Statements

This chapter describes the sequence statements that are provided with the AIM Utility package. First, the statements and arguments are documented. Finally, example applications are presented to demonstrate the use of some of the statements.

Later sections show examples on how to use these statements as well as information on other available software modules provided with the AIM Utility Package.

AIO

This statement will allow the user to either read or write to an analog board. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
AIO {Input Channel --variable-- {Gain (0 to 3) --variable-- } To (-10 to 10) --variable-- }  
    {Output Channel --variable-- Value (-10 to 10) --variable--}
```

The statement performs the following steps:

1. If the Input Channel variable is defined the statement will check the gain value and read the specified channel and place the value in the variable argument. The gain value must match the gain set by the jumpers on the AIO board. Values are 0-3
2. If the Output Channel variable is defined the statement will write to the output channel the value specified in the second argument. Please note that the value needs to be between the value of -10 to +10 to represent 10 volts.

ARM_CONFIG

This statement will return the current arm configuration of the robot. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
ARM_CONFIG Return Config --variable--
```

The statement performs the following steps:

1. This statement will return to the specified variable a value of true if the arm is in the righty configuration. Otherwise the value will be false or zero.

ATTACH_ROBOT

This statement will attach the current selected robot to program control. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
ATTACH_ROBOT
```

The statement performs the following steps:

1. This statement will perform the V+ attach statement. This is intended to be used with calibration from sequences. See the *V+ Language Reference Guide* for more details on the attach statement.

CALIBRATE

This statement will calibrate the robot from a sequence. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

CALIBRATE

The statement performs the following steps:

1. This statement performs a robot calibration from a sequence. Make sure robot power is on and other robot states are ready before calibration. Refer to the example sequence on calibration.

CLEAR_MESSAGE

This statement will clear a status message displayed by the Status_Message instruction. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

CLEAR_MESSAGE AI.CTL Array --constant--

The statement performs the following steps:

1. The *Array --constant--* argument refers to the specified array number used for the Status_Message instruction. This statement will set the ai.ctl string variable to a null string.

DETACH

This statement will detach the robot from computer control or detach the pendant. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

DETACH {Robot --yes/no--} {Pendant --yes/no--}

The statement performs the following steps:

1. By selecting the Robot option, the current robot running in the task will be detached from computer control.
2. By selecting the Pendant option the MCP will be detached.

DISTANCE_FROM

This statement will return the current robot distance from a specified location. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

DISTANCE_FROM Current POS To --location-- Return To--variable--

The statement performs the following steps:

1. The *--location--* argument is used as a reference to determine the distance in millimeters from the current robot location.
2. The actual distance is output to the *Return To --variable--* in the variable database.

GET_BITS

This statement will return a BCD value based on the current setting of a block of input signals. It also allows the user to wait for the proper input value before continuing the program. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
GET_BITS NUM of BITS --constant-- BEGINING SIGNAL --input--  
RETURN VALUE TO --variable-- {WAIT UNTIL --opr-- --variable--  
TIMEOUT --variable--}
```

The statement performs the following steps:

1. The *NUM of BITS* **--constant--** argument specifies the size of the block of inputs to be used to calculate a BCD number.
2. The *Beginning Signal* **--input--** argument is the first signal of the block of inputs used for the calculation.
3. The *Return Value To* **--variable--** argument is the output variable where the BCD value is placed.
4. The Optional *Wait Until* **--opr-- --variable--** argument can be used to wait until a comparison value occurs. The **--opr--** value allows you to specify the type of comparison operation.
5. The *Timeout* argument allows the user to specify a maximum amount of time to wait for the comparison value to occur.

GET_PARM

This statement will return the specified V+ parameter value to a variable. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
GET_PARM --parameters-- RETURN TO --variable--
```

The statement performs the following steps:

1. The **--parameters--** argument provides a pick list of the available parameters that can be selected. The V+ parameters are various system related values that can be returned to a variable.

Provided below is a listing of the available parameters that can be used with the Get_Parm and Set_Parm statements. Please refer to the *V+ Operating System Reference Guide* for more information on parameters.

belt.mode - Controls the operation of conveyor tracking.

belt.zero.count- Zero index checking for the conveyor encoder

display.camera- Display the selected camera number on the vision screen

hand.time- Delay timed used by V+ when using OpenI and CloseI Instructions

kermit.retry - Controls the number of Kermit retries with the RS-232 communication.

kermit.timeout - The amount of time before the Kermit connection is failed

screen.timeout - Time after last terminal use for screen blanking

terminal - Returns the current serial port number for the terminal

not.calibrated - Calibration status of the robot devices

2. The *-variable*- argument allows the current value of the parameter to be passed to that specified variable.

GET_STATE

This statement will return the current status of the system states. This is based on the V+ State instruction. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
GET_STATE --states-- RETURN TO --variable--
```

The statement performs the following steps:

1. The *--states--* argument provides a pick list of the available system states that can be selected. The V+ states are various system related functions that can be returned to a variable.

Provided below is a listing of the available state conditions that can be used with the Get_State statement. Please refer to the *V+ Operating System Reference Guide* for more information on parameters.

panic.button - Returns the status of the system E-Stop Button

high.power - Returns the status of the robot high power

program.start - Returns the status of the Program Start Button

2. The *-variable*- argument allows the current value of the system state to be passed to that specified variable. The variable will be returned with a true or false state.

GET_SWITCH

This statement will return the V+ system switches settings. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
GET_SWITCH --switch-- RETURN VALUE TO --variable--
```

The statement performs the following steps:

1. The *--switch--* argument provides a list of selectable V+ switches that indicate system settings.

Below is a list of the supported switches with a description of their purpose. Please refer to *V+ Operating System Reference Guide* for more information on system switches.

auto.power.off - Stops V+ from disabling High Power when certain motion errors occur. Please see the V+ reference guides for more information.

belt - Belt Tracking feature is On or Off

cp - Robot Continuous Path motion is On or Off

decel.100 - Allows the user to set the deceleration in the Accel V+ instruction.

dry.run - V+ Dry Run Feature -- runs a robot task without motion

interactive - Suppresses interactive messages displayed on the terminal

mcp.message - Suppresses messages displayed on the MCP

mcs.message - Suppresses messages displayed by the MCS instruction

monitors - Allows multiple V+ monitors if multiple processes are used

power - Robot Power is On or Off

profile - Default acceleration profile the robot is running

retry - Controls if a resume will occur if the program start button is pressed

robot[1] - Enables robot 1 to function

robot[2] - Enables robot 2 to function

robot[3] - Enables robot 3 to function

robot[4] - Enables robot 4 to function

scale.accel[1] - allows V+ to automatically scale the acceleration relative to velocity

scale.accel[2] - Same as above except for robot 2

scale.accel[3] - Same as above except for robot 3

scale.accel[4] - Same as above except for robot 4

scale.accel.rot[1] - Allows V+ to scale the acceleration for rotary axis (J4)

scale.accel.rot[2] - Same as above except for robot 2

scale.accel.rot[3] - Same as above except for robot 3

scale.accel.rot[4] - Same as above except for robot 4

set.speed - Enables the changing of the monitor speed from the MCP

trace - Selects the V+ trace options which will display the program statement as the program executes

upper - Determines if string comparisons are case sensitive.

vision - Enables the Adept Vision System to function

2. The **-variable-** argument allows the current value of the system state to be passed to that specified variable. The variable will be returned with a true or false state.

GET_TIME

This statement will return the current timer value based on the V+ timer selected. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
GET_TIME Timer Number --constant-- Push Time --variable--
```

The statement performs the following steps:

1. The **--constant--** argument refers the V+ timer value. V+ has a total of 16 timers as well as special timer modes that are a negative value. Please refer to the *V+ Language Reference Guide* for more details.
2. The **--variable--** argument will return a time value to the AIM variable database.

J4_INERTIA

This statement will set the V+ Gain.Set and Payload instructions based on the values specified in the statement. These values will change the tuning for the Joint 4 of an Adept robot to compensate for inertia payloads on the tool mounting flange. Please see the *V+ Language Reference Guide* and the *Robot User's Guide* for more information on the values needing to be specified in this statement. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
J4_INERTIA {Gain.Set (1-4) --variable--} {Payload (1-100) --variable--}
```

The statement performs the following steps:

1. If the Gain.Set variable is specified, that value will be executed in a V+ gain.set instruction. The values will be different depending on the model of Adept robot that is running. (See the *Robot User's Guide* for more information on gain values.)
2. If the Payload variable is specified, that value will be executed in a V+ payload instruction. The values will be different depending on the model of Adept robot and how much payload is on the tool mounting flange.

MCP_CHECK

This statement will return a value, 1 to 5, to correspond to the button pressed on the MCP. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
MCP_CHECK BUTTON # (1-5) --variable-- {Detach Pendant --yes/no--}
```

The statement performs the following steps:

1. The --variable-- argument above allows the user to determine which of the 5 function buttons were pressed. A value 1 thru 5 will be returned which can be handled from the AIM sequence.
2. The *Detach Pendant* selection allows the user to detach the pendant after completion of the task. Keeping the pendant attached will prohibit selection of the pendant from other V+ tasks.

MCP_WRITE

This statement will write to the MCP the string data that will come from the variable database. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
MCP_WRITE BUTTON 1 --string-- BUTTON 2 --string-- BUTTON 3 --string--  
BUTTON 4 --string-- BUTTON 5 --string-- {Line 1 Text --string_var--  
{Line Counter --variable-- }} {Detach Pendant --yes/no--}
```

The statement performs the following steps:

1. The --string-- argument will write to the MCP function button centered above the button number specified. This allows the user to create a sequence that can use the MCP as an operator interface.
2. The *Line 1 Text* option allows the user to display a text string on the first line on the LCD display on the teach pendant. This text string is read from the variable database.

3. The *Line Counter* argument allows the user to specify a number value that will be appended to the first line string displayed on the pendant.
4. The *Detach Pendant* selection allows the user to detach the pendant after completion of the task. Keeping the pendant attached will prohibit selection of the pendant from other V+ tasks.

MOUNT_NFS

This statement will allow the user to mount another drive or memory source to be used with the Adept controller via NFS. This statement allows the user to mount a drive via an AIM sequence. This may be useful in a startup sequence. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
MOUNT_NFS Server Name --string_var-- Server Address --string_var--  
Mount Name --string_var-- Mount Path --string_var--
```

The statement performs the following steps:

1. This statement will mount another computer's disk drive or memory source via NFS. It basically performs the equivalent of the V+ monitor instruction FSET to mount the drive. The *Server Name* argument is used to specify the node or name of the computer you are attached to. Please note that you can have as many as nine attachments to other sources, which includes multiple computers.
2. The *Server Address* argument specifies the IP address of the computer that is running on ethernet. Please check your computer for this IP address. Please make sure the address is not duplicated on another computer.
3. The *Mount Name* argument specifies the drive name you will be using within the Adept controller. An example is that the **D:** is the flash ram on the Adept AWC. The name, for example, may be *PCA* for pc computer drive A.
4. The Mount Path argument specifies the path to the other computers folders. This path may include other subdirectories or folders within the computer. The following is an example:
 \A --- for the A drive on the other computer
 \C --- for the C drive
 \C:\ADEPT\ --- for the Adept directory on the C drive

MOVE_UT

This statement is similar to the AIM *MOVE* statement, except it allows the user to specify if the Single or Multiple motion instructions are applied to the move. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
MOVE_UT {{APPROACH --path--} FROM --location--} {ALONG --path--}
TO --location-- {DEPART --path--} {USING --tool--}
{{ALONG --path--} REJECT --location-- {RETURN --path--}}
{WAIT_AT --location--} {OK_SIGNAL --o_variable--}
{Single@Appro --yes/no--} {Single@Move --yes/no--}
```

The statement performs the following steps:

1. The *APPROACH* argument specifies a controlled path motion to the location specified in the *FROM* argument. This is an optional argument.
2. The *FROM* argument allows the user to move to a specified location. This argument is optional.
3. The *ALONG* argument allows the user to specify a controlled path motion to the location specified in the *TO* argument.
4. The *TO* argument specifies the location the robot is moving to. This is the only required argument in this statement.
5. The *Depart* argument allows the user to specify an exit path away from the specified location.
6. The *USING* argument allows the user to specify a tool offset that will be applied during the motions.
7. The *ALONG* path argument allows the user to move along a path to the selected location.
8. The *REJECT* argument allows the user to specify a location for the robot to move should a failure occur. Please refer to the *MotionWare User's Guide* for more information.
9. The *RETURN* argument allows a path to be taught away from the reject location.
10. The *WAIT_AT* argument is a location where the robot will wait at in the case of a moving frame that is out of reach.
11. The *OK_SIGNAL* argument is a signal that gets set if the motion is successful.
12. The *SINGLE@APPRO* if set to yes will force the Joint 4 motion to remain within +_180 degrees relative to the zero position of the joint. This will force the joint to "unwind" during the motion to the position. This occurs during the motion to the approach location. This is sometimes desired if the motion is occurring with line tracking to prevent an unwind condition during tracking. If not defined it is set to the multiple selection.
13. The *SINGLE@MOVE* if set to yes will force the Joint 4 motion to remain within +_180 degrees relative to the zero position of the joint. This will force the joint to "unwind" during the motion to the position. This occurs during the motion to the final location. If not defined it is set to the multiple selection.

OPEN_MENU

This statement will allow the user to open an AIM menu from a sequence. A common use would be opening a menu during a startup sequence. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
OPEN_MENU Page Name --string_var-- File Name --string_var--
```

The statement performs the following steps:

1. The Page Name --string_var-- argument specifies the menu page name to be opened by this statement.
2. The File Name --string_var-- argument specifies the Menu File to be opened by this statement.

PMOVE

This statement will allow the user to move to a precision point location. The location is defined in joint coordinates rather than world coordinates. This is very useful when using jointed robot arms. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
PMOVE {FROM --ppoint-- } TO --ppoint--
```

The statement performs the following steps:

1. The *FROM* argument allows the user to move to a precision point location and then to a second precision point location from a single statement. This is an optional argument.
2. The *TO* argument will move to a joint defined location based on the specified precision point record in the database.

RETRACT_Z

This statement will move the Z axis of the robot to a specified height. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
RETRACT_Z Robot Height --constant-- {Speed --constant--} {Accel --constant--}  
{Decel --constant--}
```

The statement performs the following steps:

1. The Robot Height --constant-- argument allows the user to specify the absolute world coordinate Z height for the robot to move.
2. The Speed, Accel and Decel arguments specify the motion parameters for the motion.

RETURN_LOC

This statement will return the current world coordinate location of the robot. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
RETURN_LOC {X Pos --variable--} {Y Pos --variable--} {Z Pos --variable--}
{Yaw Pos --variable--} {Pitch Pos --variable--} {Roll Pos --variable--}
```

The statement performs the following steps:

1. The --variable-- arguments are all optional and will return the world coordinate value based on the robot's current position. Coordinates are returned in units of millimeters or degrees.

RETURN_JTS

This statement will return the current location of the robot in joint coordinates. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
RETURN_JTS {JT 1 --variable--} {JT 2 --variable--} {JT 3 --variable--}
{JT 4 --variable--} {JT 5 --variable--} {JT 6 --variable--}
```

The statement performs the following steps:

1. The --variable-- arguments are all optional and will return the Joint coordinate value based on the robot's current position. Coordinates are returned in units of millimeters or degrees.

SET_BITS

This statement will turn on a block of outputs based on the provided BCD number. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SET_BITS Num of Bits --constant-- Begining Signal --output--
Set Value To --variable--
```

The statement performs the following steps:

1. The *Num of Bits* --variable-- argument will specify the number of output signals used to represent the provided BCD number.
2. The *Begining Signal* --output-- argument specifies the first signal number of the block of outputs to be used.
3. The *Set Value To* --variable-- argument is the BCD value the outputs will be set to.

SET_OPRMODES

This statement allows the user to set internal AIM settings that are normally set by the Task Control panel or the initialization database. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SET_OPRMODES TASK --constant-- TYPE --oprmodes-- SETTING --variable--
```

The statement performs the following steps:

1. The *TASK --constant--* argument specifies the task to which the settings will apply. This allows the user to configure different values for different sequences.
2. The *TYPE --oprmodes--* argument allows the user to select from a pick list different AIM selections that can occur. Currently the only selections that are available are the settings from the Task Control Panel. The following is a list of the selections.

SPEED - Refers to the robot speed that is set in the *Task Control Panel*

REPEAT -Refers to the setting of the repeat selection box in the *Task Control Panel*.

3. The *SETTING --variable--* argument sets the selected argument to the specified value. The repeat selection will run continuously if set to 0.

SET_PARM

This statement will set the V+ parameters to the specified value in the statement. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SET_PARM --parameters-- EQUAL TO --variable--
```

The statement performs the following steps:

1. The *--parameters--* argument provides a pick list of the available V+ parameters that this statement will change. Please refer to GET_PARM on page 58 for more details on the available pick list options.
2. The *EQUAL TO --variable--* will set the selected V+ parameter to the specified value.

SET_PATH

This statement will allow the user to modify a path segment position by the coordinates specified. This allows the user to modify a position based on variable data. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SET_PATH --path-- SEGMENT --variable--  
{ X: --variable-- } { Y: --variable-- } { Z: --variable-- } { y: --variable-- }  
{ p: --variable-- } { r: --variable-- }
```

The statement performs the following steps:

1. The *--path--* argument specifies the AIM path that the position will be applied toward.
2. The *SEGMENT* argument specifies which path node will be changed.
3. The arguments (X,Y,Z,y,p,r) represent the transformation for the new position.

SET_SWITCH

This statement will enable or disable the specified V+ switch selected. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SET_SWITCH --switch-- EQUAL TO --on/off--
```

The statement performs the following steps:

1. The `--switch--` argument provides a pick list of the available V+ system switches that can be set. Please refer to GET_SWITCH on page 59 for more details on the available V+ switches.
2. The `EQUAL TO --on/off--` argument will enable or disable the selected V+ switch.

SHIFT_MOVE

This statement will allow the user to move to a location offset by X,Y,Z,T dimensional data. This allows the user to modify a position based on variable data. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
SHIFT_MOVE World_Loc --location-- By X: --variable-- Y: --variable--  
Z: --variable-- T: --variable-- Along Tool --yes/no-- {Appro_Only --yes/no-- }  
{OK_SIGNAL --o_variable--}
```

The statement performs the following steps:

1. The *World_Loc* argument specifies the location that the offset will be applied toward.
2. The arguments (X,Y,Z,T) represent the amount the location will be offset based on the axis or coordinate specified.
3. The *Along Tool* argument allows the motion to be applied to tool coordinates rather than world coordinates.
4. The *Appro_Only* argument if set to yes will only move to the approach location and bypass the other motions in the record specification.
5. The *OK_SIGNAL* argument will turn on the signal if everything is successful in the specified motion.

STATUS_MESSAGE

This statement will post to a specified \$ai.ctl array variable the specified string variable. This allows the developer to have a separate status message display on a menu. The message can be set from various sequences. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
STATUS_MESSAGE AI.CTL Array --constant-- Message String --string_var--  
{Wait For --constant--}
```

The statement performs the following steps:

1. The *AI.CTL Array --constant--* argument refers to a user specified array number that can be used by an AIM menu. We recommend using values greater than 200, because AIM uses lower array values for other messages that may be displayed. Several message displays can be used with this statement.
2. The *Message String --string_var--* argument will take text from the variable database based on the variable specified.
3. The *Wait For --constant--* argument will wait for the user specified input signal to go to a true state before continuing. The input signal specified can be a soft signal (2000-2512) if desired.

TASK_MODE

This statement will return the current state of the AIM sequence based on the task specified. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
TASK_MODE TASK --constant-- MODE --variable-- {WAIT UNTIL --variable--}
```

The statement performs the following steps:

1. The *TASK --constant--* argument specifies the desired AIM task number to determine the current status. You can look at the Task Control Panel to verify the task number for the sequence if needed.
2. The *MODE --variable--* argument will return to the specified variable database the following values indicating the state of the AIM task.
 - 0 - Sequence is currently Running
 - 1 - Sequence is in Teach mode
 - 2 - Operator Attention has occurred
 - 3 - Sequence is Idle
 - 4 - Group Wait
3. The *WAIT UNTIL --variable--* argument will allow the statement to wait until the specified task state has been reached. The expected state is specified in the variable field provided.

WAIT_FOR_IO

This statement will look for the specified input state and return the amount of time it took to achieve this state as well as allowing a time-out period. The statement's syntax is as follows, where the braces (`{ . . . }`) define optional clauses:

```
WAIT_FOR_IO INPUT --constant-- Timer Num --constant--  
{Time to State --variable--} {TIME OUT --constant--}
```

The statement performs the following steps:

1. The *INPUT* *--constant--* argument specifies a digital I/O number or software output signal.
2. The *Timer Num* *--constant--* allows the user to specify a V+ timer number to be used during the wait period.
3. The *Time to State* *--variable--* returns to the variable database the amount of time it took for the specified I/O to achieve a true state.
4. The *TIME OUT* *--constant--* argument allows the statement to wait a maximum amount of time in this statement. The sequence can monitor the Time to State variable to indicate the maximum time to achieve the desired state.

TYPE_MON

This statement will type text to the V+ monitor based on the argument supplied. The statement's syntax is as follows, where the braces (`{ . . . }`) define optional clauses:

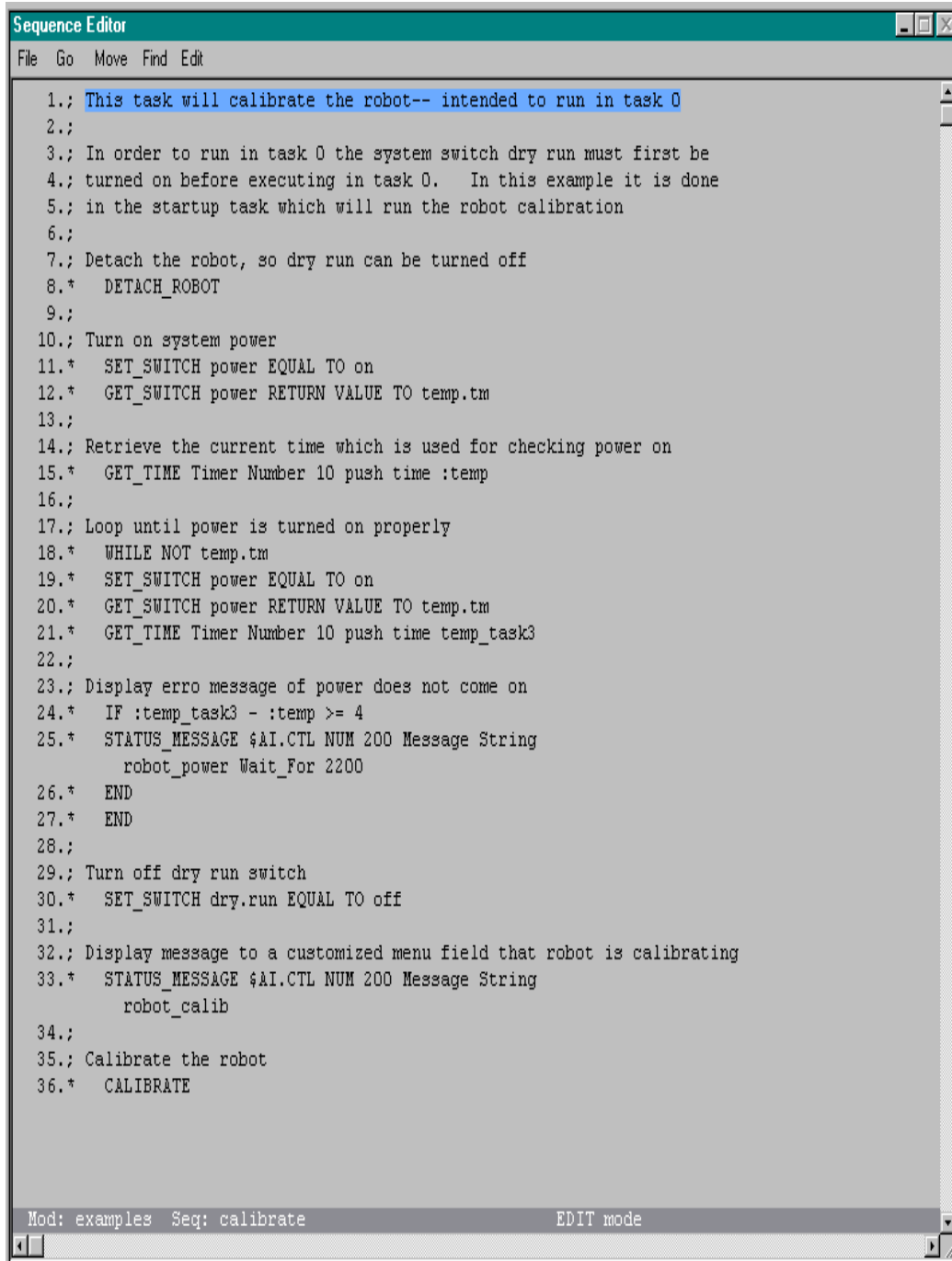
```
TYPE_MON String --string-- VAR --variable-- {String --string-- VAR --variable--}
```

The statement performs the following steps:

1. The String *--string--* arguments allow the user to provide a text string to be typed on the V+ monitor.
2. The VAR *--variable--* allows the user to type the current variable value from the variable database to the V+ monitor.

4.2 Calibration from Sequences

The following sequence is an example of calibrating from a control sequence. This sequence is provided in the example sequence supplied with this software package.



```

Sequence Editor
File Go Move Find Edit

1.; This task will calibrate the robot-- intended to run in task 0
2.;
3.; In order to run in task 0 the system switch dry run must first be
4.; turned on before executing in task 0. In this example it is done
5.; in the startup task which will run the robot calibration
6.;
7.; Detach the robot, so dry run can be turned off
8.* DETACH_ROBOT
9.;
10.; Turn on system power
11.* SET_SWITCH power EQUAL TO on
12.* GET_SWITCH power RETURN VALUE TO temp.tm
13.;
14.; Retrieve the current time which is used for checking power on
15.* GET_TIME Timer Number 10 push time :temp
16.;
17.; Loop until power is turned on properly
18.* WHILE NOT temp.tm
19.* SET_SWITCH power EQUAL TO on
20.* GET_SWITCH power RETURN VALUE TO temp.tm
21.* GET_TIME Timer Number 10 push time temp_task3
22.;
23.; Display error message of power does not come on
24.* IF :temp_task3 - :temp >= 4
25.* STATUS_MESSAGE $AI.CTL NUM 200 Message String
    robot_power Wait_For 2200
26.* END
27.* END
28.;
29.; Turn off dry run switch
30.* SET_SWITCH dry.run EQUAL TO off
31.;
32.; Display message to a customized menu field that robot is calibrating
33.* STATUS_MESSAGE $AI.CTL NUM 200 Message String
    robot_calib
34.;
35.; Calibrate the robot
36.* CALIBRATE

Mod: examples Seq: calibrate EDIT mode

```

Figure 4-1 Calibration Sequence

4.3 Using the Message Statements

The message statements are provided for a controlled method of displaying messages from a menu display. The statements are STATUS_MESSAGE and CLEAR_MESSAGE. Many AIM installations have error recovery by the use of a sequence and therefore many of the AIM messages that would normally be used by the operators can be ignored. This allows the sequence to display a message to a developed menu via the AIM \$ai.ctl array values. This instruction basically transfers a predefined variable string from the variables database to the \$ai.ctl variable. This variable can then be displayed on a menu screen. Additionally a software signal can be specified as an acknowledgment that the error occurred.

4.4 MCP example

The MCP sequence statements MCP_WRITE and MCP_CHECK are provided to allow users to develop AIM sequences that can control an interface for use with Adept's MCP. These statements allow the user to write messages to the MCP and check for the function button presses. Below is a sequence that was developed for a customer that wanted to change the pallet rows and columns from the teach pendant. This sequence can be found on the AIM Utilities Database Examples disk.

```

Sequence Editor
File Go Move Find Edit

1.; This is the main Pendant Routine for station 4
2.  diverter_column:
3.    WHILE -1
4.      MCP_WRITE BUTTON 1 Increase BUTTON 3 Decrease BUTTON 5
5.      Done Line 1 Text div_lline_col Line Counter tray_col
6.      MCP_CHECK BUTTON 1 increase BUTTON 3 decrease BUTTON 5
7.      done
8.      IF increase
9.        IF tray_col >= 1 AND tray_col < tray_col_max
10.         SET tray_col = tray_col + 1
11.       END
12.     ELSE IF decrease
13.       IF tray_col > 1 AND tray_col <= tray_col_max
14.         SET tray_col = tray_col - 1
15.       END
16.     ELSE
17.       GOTO diverter_row
18.     END
19.   END
20.  diverter_row:
21.    WHILE -1
22.      MCP_WRITE BUTTON 1 Increase BUTTON 3 Decrease BUTTON 5
23.      Done Line 1 Text div_lline_row Line Counter tray_row
24.      MCP_CHECK BUTTON 1 increase BUTTON 3 decrease BUTTON 5
25.      done
26.      IF increase
27.        IF tray_row >= 1 AND tray_row < tray_row_max
28.         SET tray_row = tray_row + 1
29.       END
30.     ELSE IF decrease
31.       IF tray_row > 1 AND tray_row <= tray_row_max
32.         SET tray_row = tray_row - 1
33.       END
34.     ELSE
35.       GOTO diverter_column
36.     END
37.   END
38. END
  
```

Mod: ex_mcp Seq: ex_mcp_sequence EDIT mode

Figure 4-2MCP Sequence

4.5 Tool Offset Teaching

Figure 4-3 shows a tool transformation for an Adept Robot. If no tool transformation is in effect, the x-axis of the tool frame points along the tool flange key-way and, the z-axis points down. In this example, the tool transformation (defined by the X' , Y' , and Z' coordinate system) has offsets in the X, Y, and Z directions, but no change in the y, p, and r components. However, tool transformations can have values for all six transformation components.

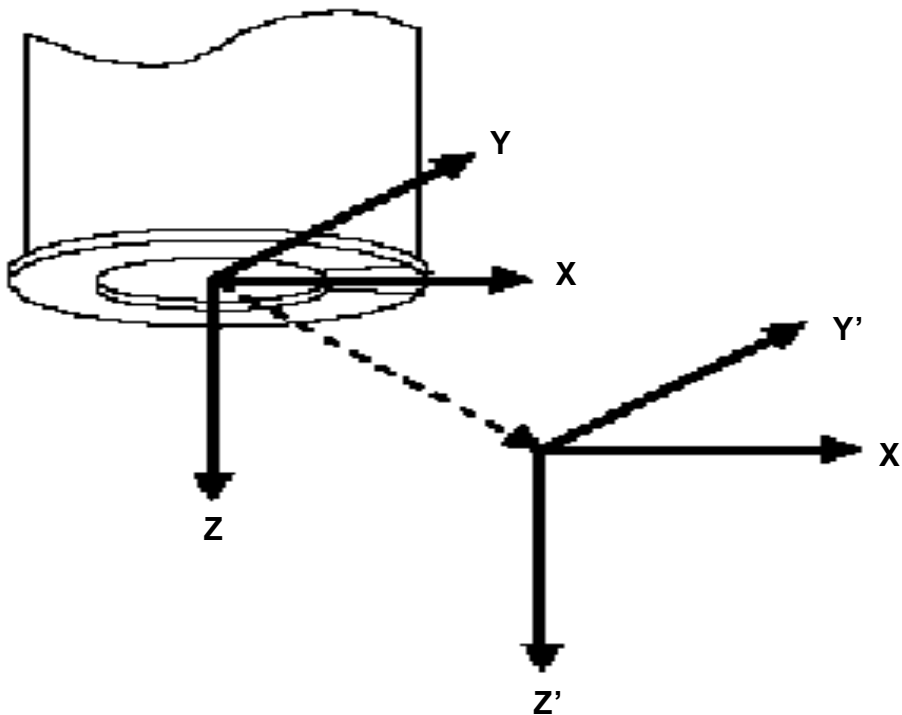


Figure 4-3 Cartesian Tool Offset

To teach a tool transformations, open a tool record and press Teach. The menu shown in Figure 4-4 will be displayed on the pendant.

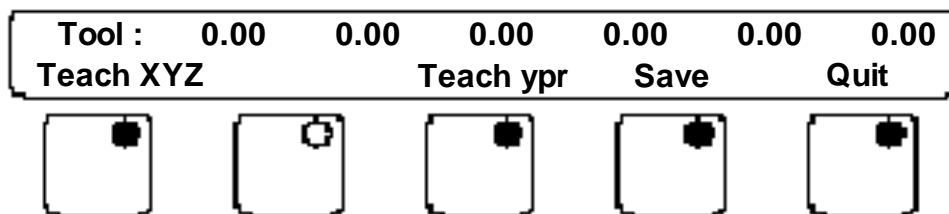


Figure 4-4 Tool Offset Utility Menu Display

Both Cartesian (X, Y, Z) offsets and orientation (yaw, pitch, roll) offsets can be measured. To measure the Cartesian offsets:

1. Press the TEACH XYZ soft key.
2. The pendant will prompt you to teach a series of locations. Place the tooling center tip at a given location and press the pendant REC/DONE key. Rotate the tip (at least 30°) about this given location and press REC/DONE again. A minimum of two locations must be taught to measure the offset. However, more accurate results are obtained if more locations are taught. (The robot 'FREE' mode can be used to simplify the teaching process.)
3. Once the locations are taught, press the "CALC TOOL" soft key (Figure 4-5). The locations are "averaged" and used to define the tool offset. The routine will give you the option of not using the location furthest away from the calculated center in the tool transformation calculation. Press "QUIT" to return to the previous menu.

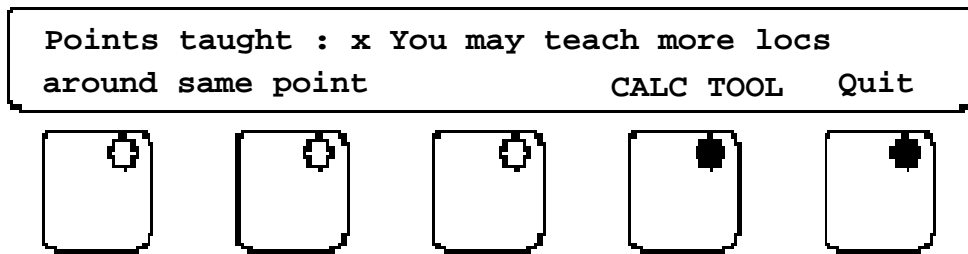


Figure 4-5 Calculating XYZ Offsets

If the tooling is attached at an angle with respect to the robot tool flange, and is not aligned normal to the tool z-axis of the robot and not parallel to the tool coordinate system (i.e., not perpendicular to the robot tool flange surface), the orientation offset might need to be measured as well. Figure 4-6 shows an example of a tool transformation that contains yaw, pitch, and roll components.

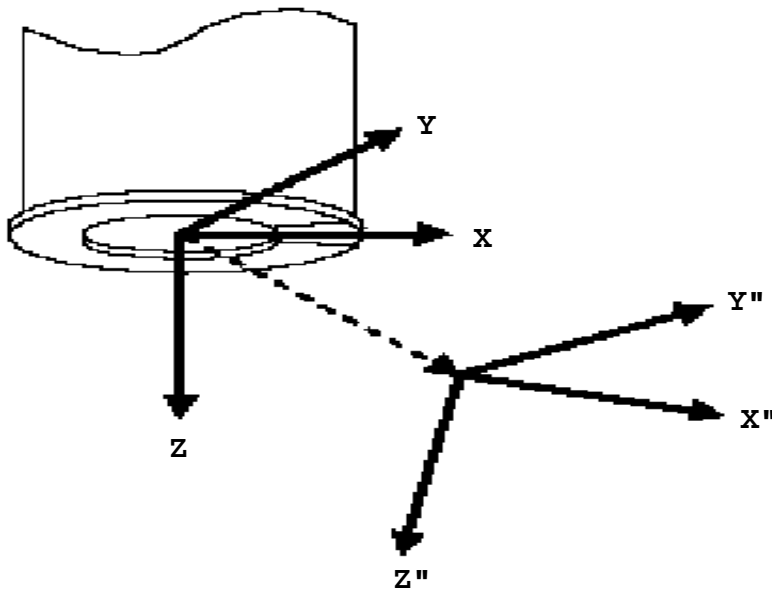


Figure 4-6
General Tool Offset

To teach the yaw, pitch, and roll components of a tool transformation:

1. Press the "TEACH YPR" soft key (see Figure 4-4).
2. The pendant will prompt you to teach two robot locations along the tool's z-axis (the Z" axis in Figure 4-6.). Both locations must have the same orientation. Press the "REC/DONE" key to record the locations.
3. After both locations have been taught, you will be prompted to teach a location along the positive x-direction or the positive y-direction of the tool (the X" or Y" axes of Figure 4-6 respectively). Again, the orientation of the tool should not be changed. Once the last location is taught, the yaw, pitch, and roll offsets are automatically calculated. Press the "QUIT" soft key to return to the top-level menu.

Press the "SAVE" soft key to record the measured transformation in the Tool record. Press the "QUIT" soft key to terminate the teaching operation.

Testing the Tool Offset

The method to test the tool transformation with the AIM Utilities software is the same as the standard AIM Package. To test the tool transformation:

1. Open a Tool record:
Edit ➡ Tool ➡ Seek ➡ Index ➡ *double click on desired record*
2. Press **SET TOOL** to set the tool specified in the Tool Menu.

3. From the pendant, choose the TOOL manual mode to move the robot around the defined tool location. If the tool offset is correct, the robot should rotate about its tool tip when rotated about X, Y, and Z, and it should move correctly along the tool X-, Y-, or Z-axes

4.6 Paths Relative to Frames

The AIM Utilities package allows the standard AIM Path data to be relative to frames. The file PATH.SQU provided with the AIM Utilities disk is different than the standard file provided with AIM. Paths relative to frames are accomplished by selecting the Named frame selection in the individual path records. This must be selected for every path record along your process using the frame.

4.7 Precision Point Module

The precision point software module is provided for those applications where robot rotary axes must be defined in a joint position rather than a world coordinate position. Examples are robots that have multiple axis definition to define a point or where an axis must unwind a joint before motion can begin. The Precision Point database menus are shown in Chapter 3 of this manual. The sequence statement Pmove is provided for these motions.

4.8 Software Plug-Ins

Several software modules are provided with the easy install file(*.INF). We call these modules Plug-Ins. These modules are additional software functions that are neat features or are totally separate applications. Future software features will be available in a Plug-In form for ease of installation. Plug-Ins can be installed by accessing the *Software Module Loader* under the *SETUP* pulldown. Select the *Install* button to get to the menu shown below. The AIM Utilities Software Plug-In disk contains the following software modules. The display below is the actual Install window that will appear when using this disk.

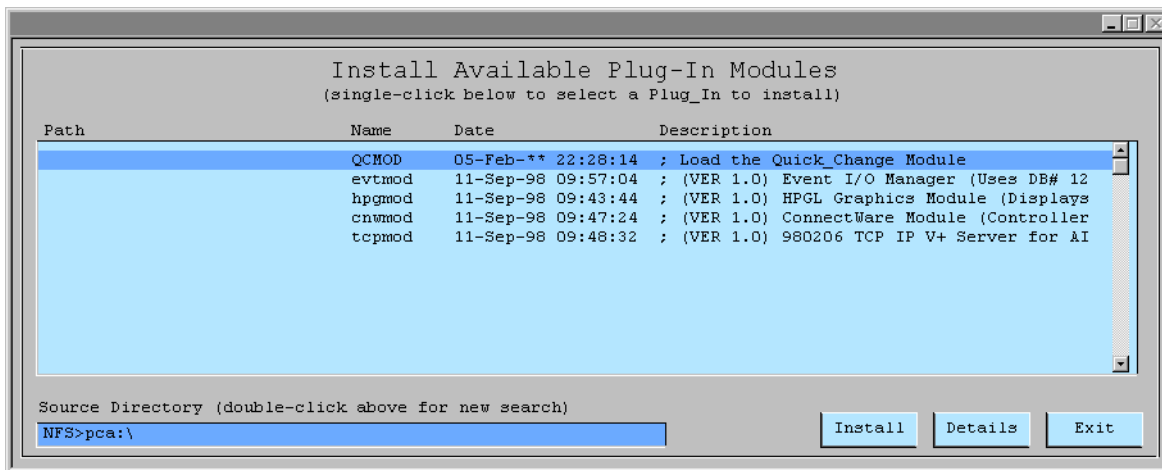


Figure 4-7
Available Software Plug-Ins

The available Plug-Ins are listed below:

1. QCMOD.INF --- Quick Change Module for Changeable Tooling
This software module provides utilities to deal with tool signatures and placing and retrieving tooling from nest locations. See the appendix chapters provided in this manual for more details.
2. EVTMOD.INF --- Event I/O Manager. This module allows the user to setup I/O events that can be measured in Time and a selected output triggered if the event fails. See the appendix chapters provided in this manual for more details.
3. HPGMOD.INF --- HPGL Graphics Module (Displays Corel HPGL Exports)
This module allows the user to setup graphic panels on a menu page to display disk resident files onto the menu. HPGL export is currently the only file format supported. See the appendix chapters provided in this manual for more details.
4. CNWMOD.INF --- ConnectWare Module. This module adds new custom statements that allows multiple MV Controllers that are running AdeptNet to communicate through AIM DDE. See the appendix chapters provided in this manual for more details.
5. TCPMOD.INF --- TCP IP V+ Server. This module allows a PC or MV TCP Client to retrieve information from the controller that is running this application. See the appendix chapters provided in this manual for more details.

4.9 Example Sequences

A diskette is provided with AIM utilities that provide examples for different applications. These files are Imported using the AIM Modules utilities. In most cases there are several databases provided with these examples. The examples are provided to give ideas to the user to solve their application problems. They are not intended to work directly in a customer's application. The list below describes the database modules available at this time. Example sequences are documented in appendix A of this manual.

1. EX_DISP.MOD --- This database module contains dispense examples for use with the AIM Dispense Module. Included with in this example is a conveyor dispensing application, a Cell Control or supervisor sequence for error recovery and other dispense examples.
2. EX_MCP.MOD --- This example database module is a sample sequence of how to use the MCP sequence task statements provided in the AIM Utility package. This example shows how to create a sequence to allow a pendant routine to change the palletizing data in the locations database.
3. EX_PALWR.MOD --- This example database module is an example PalletWare application. The AIM PalletWare software must be loaded to view these example databases.
4. EX_START.MOD --- This example shows a startup sequence which will also enable power and calibrate the robot.

Sequence Examples



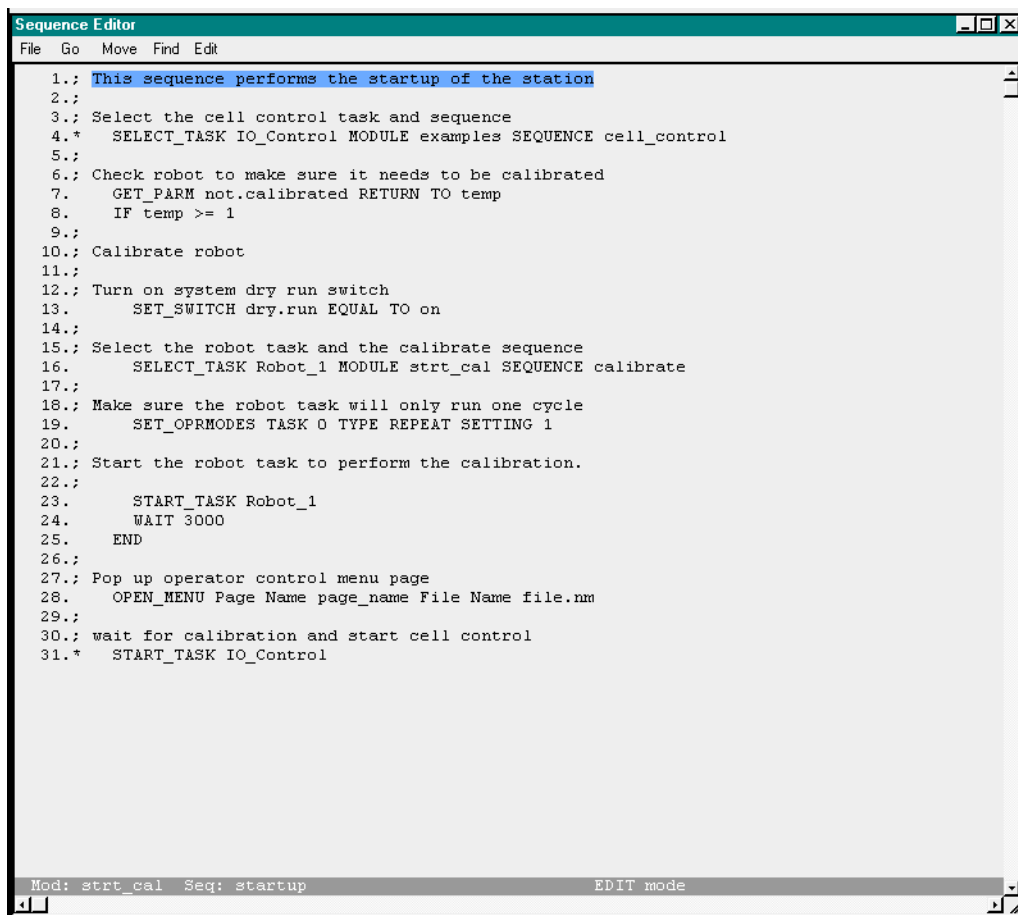
A.1 Introduction and Overview	78
Start-Up and Calibrate Sequences.	78
Calibrate Sequence	79
Start-Up Sequence Example from PalletWare	80
Start Button Control Sequence	81
Cell Control Sequence Example	82
Second Cell Control Example	86
Robot Main Sequence	91
Main Sequence Example from PalletWare	92
MCP Sequence Example	101

A.1 Introduction and Overview

Provided in the appendix are several example sequences which are available in various files on the AIM Utilities Example Database disk. They are listed here for reference.

Start-Up and Calibrate Sequences

These sequences are examples of how to auto-start and calibrate robot systems. This first example resides in the 'EX_START.MOD' modules files. This sequence will startup the production cell and calibrate the robot. The calibrate sequence is the second sequence provided.



```

Sequence Editor
File  Go  Move  Find  Edit

1.; This sequence performs the startup of the station
2.;
3.; Select the cell control task and sequence
4.*  SELECT_TASK IO_Control MODULE examples SEQUENCE cell_control
5.;
6.; Check robot to make sure it needs to be calibrated
7.   GET_PARM not.calibrated RETURN TO temp
8.   IF temp >= 1
9.;
10.; Calibrate robot
11.;
12.; Turn on system dry run switch
13.   SET_SWITCH dry.run EQUAL TO on
14.;
15.; Select the robot task and the calibrate sequence
16.   SELECT_TASK Robot_1 MODULE strt_cal SEQUENCE calibrate
17.;
18.; Make sure the robot task will only run one cycle
19.   SET_OPRMODES TASK 0 TYPE REPEAT SETTING 1
20.;
21.; Start the robot task to perform the calibration.
22.;
23.   START_TASK Robot_1
24.   WAIT 3000
25.   END
26.;
27.; Pop up operator control menu page
28.   OPEN_MENU Page Name page_name File Name file.nm
29.;
30.; wait for calibration and start cell control
31.*  START_TASK IO_Control

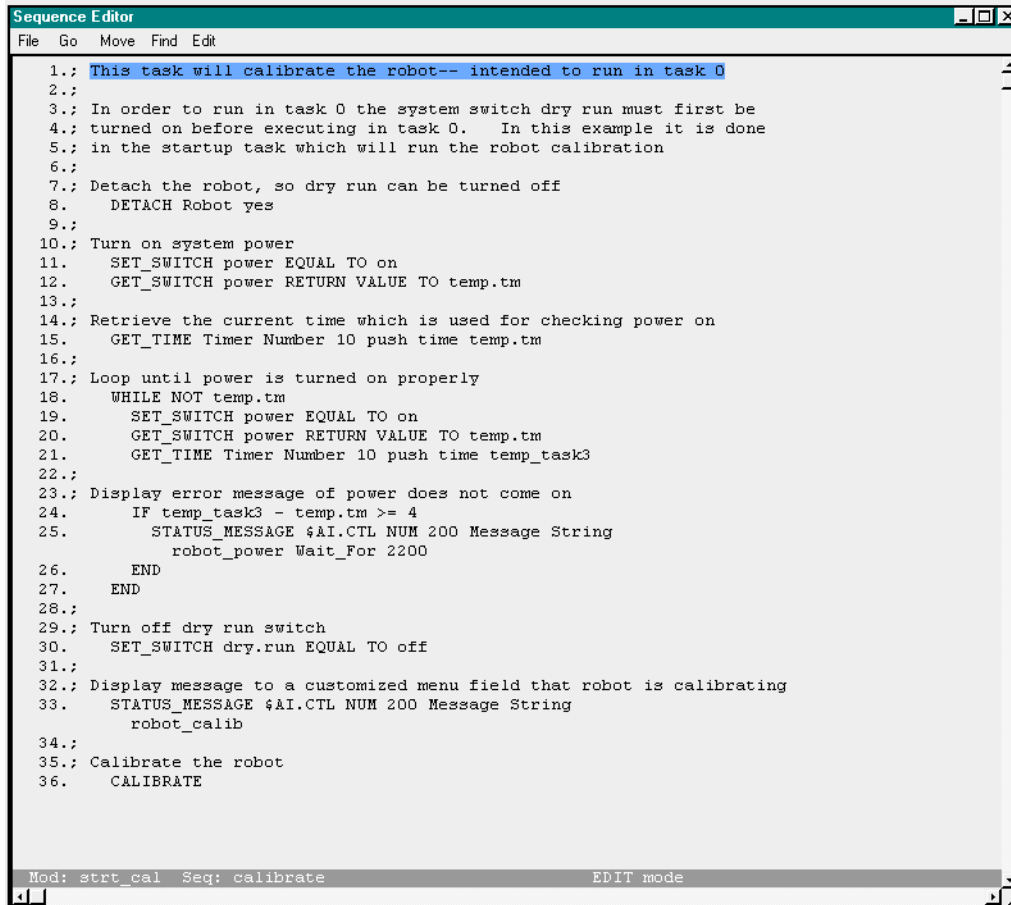
Mod: strt_cal  Seq: startup  EDIT mode

```

Figure A-1Auto Start-Up Sequence

Calibrate Sequence

This calibrate sequence is located in the 'EX_START.MOD' modules files provided with AIM Utilities.



The screenshot shows a window titled "Sequence Editor" with a menu bar (File, Go, Move, Find, Edit) and a text area containing the following code:

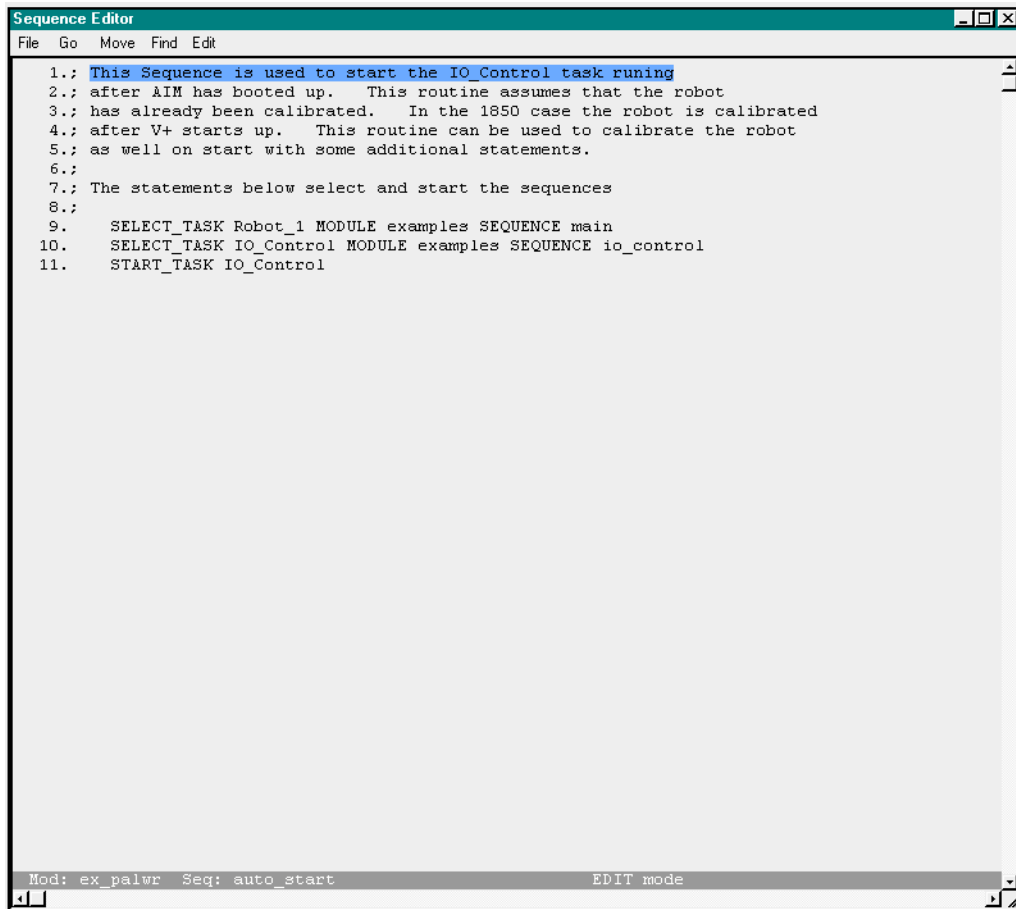
```
1.; This task will calibrate the robot-- intended to run in task 0
2.;
3.; In order to run in task 0 the system switch dry run must first be
4.; turned on before executing in task 0. In this example it is done
5.; in the startup task which will run the robot calibration
6.;
7.; Detach the robot, so dry run can be turned off
8.   DETACH Robot yes
9.;
10.; Turn on system power
11.   SET_SWITCH power EQUAL TO on
12.   GET_SWITCH power RETURN VALUE TO temp.tm
13.;
14.; Retrieve the current time which is used for checking power on
15.   GET_TIME Timer Number 10 push time temp.tm
16.;
17.; Loop until power is turned on properly
18.   WHILE NOT temp.tm
19.     SET_SWITCH power EQUAL TO on
20.     GET_SWITCH power RETURN VALUE TO temp.tm
21.     GET_TIME Timer Number 10 push time temp_task3
22.;
23.; Display error message of power does not come on
24.   IF temp_task3 - temp.tm >= 4
25.     STATUS_MESSAGE $AI.CTL NUM 200 Message String
26.       robot_power Wait_For 2200
27.   END
28.;
29.; Turn off dry run switch
30.   SET_SWITCH dry.run EQUAL TO off
31.;
32.; Display message to a customized menu field that robot is calibrating
33.   STATUS_MESSAGE $AI.CTL NUM 200 Message String
34.     robot_calib
35.; Calibrate the robot
36.   CALIBRATE
```

At the bottom of the window, a status bar shows "Mod: strt_cal Seq: calibrate" and "EDIT mode".

Figure A-2 Calibrate Sequence Example

Start-Up Sequence Example from PalletWare

This sequence is a startup sequence that came from the PalletWare example. With the Adept 1850 Robot, calibration occurs during V+ boot-up because the robot uses absolute encoders. This is a very simple way to start another sequence running in a different task. This example is located in the File 'EX_PALW.MOD



The screenshot shows a window titled "Sequence Editor" with a menu bar (File, Go, Move, Find, Edit) and a text area containing the following sequence code:

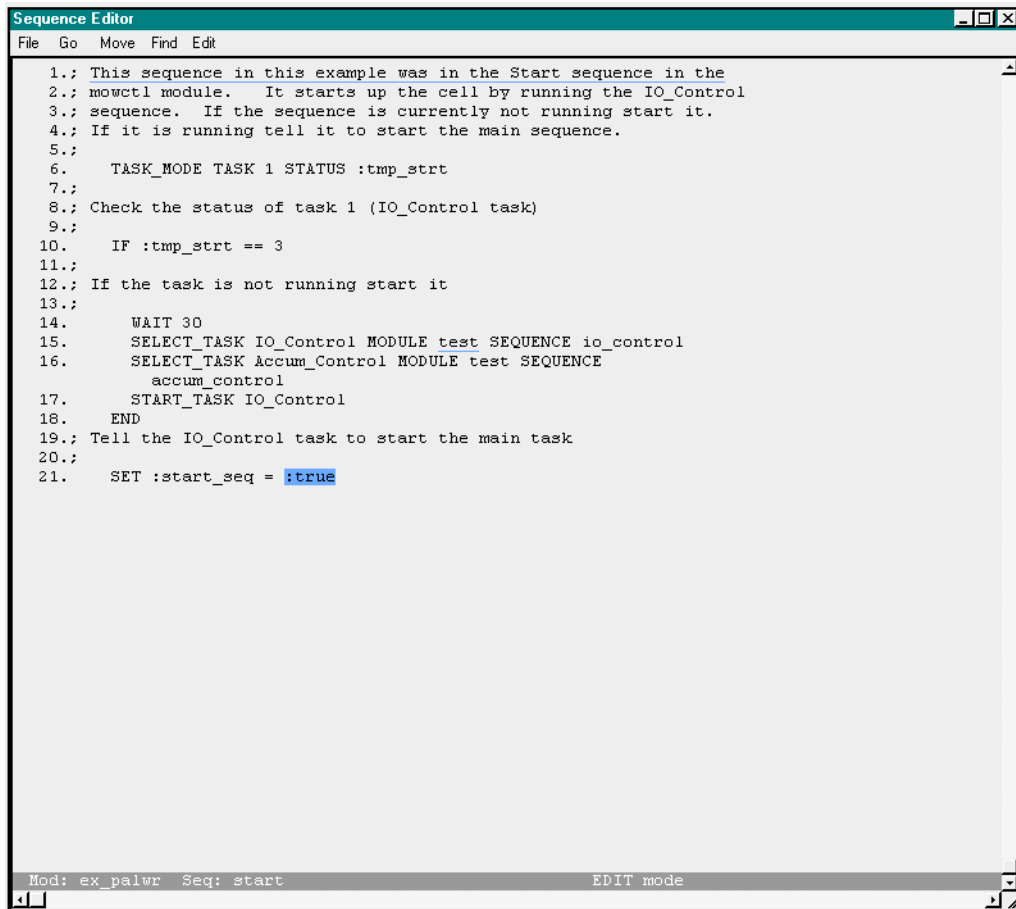
```
1.; This Sequence is used to start the IO_Control task runing
2.; after AIM has booted up. This routine assumes that the robot
3.; has already been calibrated. In the 1850 case the robot is calibrated
4.; after V+ starts up. This routine can be used to calibrate the robot
5.; as well on start with some additional statements.
6.;
7.; The statements below select and start the sequences
8.;
9.    SELECT_TASK Robot_1 MODULE examples SEQUENCE main
10.   SELECT_TASK IO_Control MODULE examples SEQUENCE io_control
11.   START_TASK IO_Control
```

The status bar at the bottom indicates "Mod: ex_palwr Seq: auto_start" and "EDIT mode".

Figure A-3PalletWare Start-Up

Start Button Control Sequence

This example is a modified version of the provided START sequence. This modification will start the IO_Control task. This example is located in the file 'EX_PALW'.



```
Sequence Editor
File  Go  Move  Find  Edit

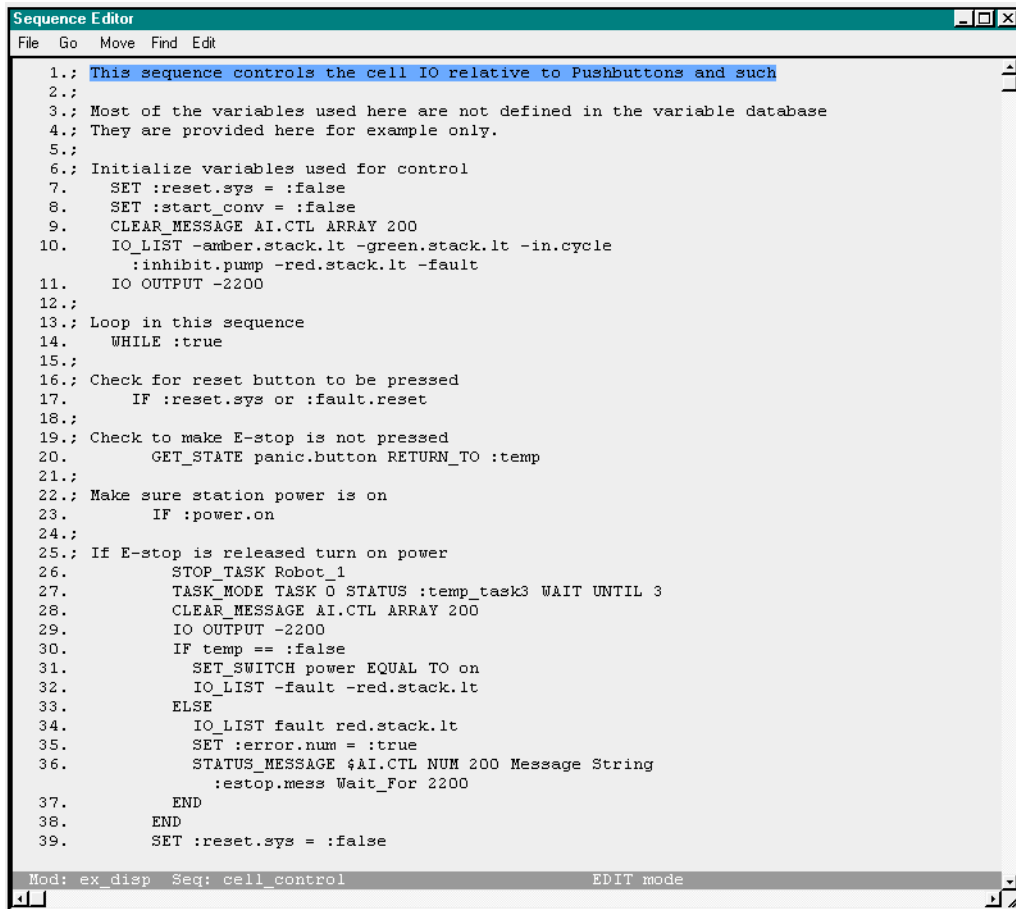
1.; This sequence in this example was in the Start sequence in the
2.; mowctl module. It starts up the cell by running the IO_Control
3.; sequence. If the sequence is currently not running start it.
4.; If it is running tell it to start the main sequence.
5.;
6.    TASK_MODE TASK 1 STATUS :tmp_strt
7.;
8.; Check the status of task 1 (IO_Control task)
9.;
10.   IF :tmp_strt == 3
11.;
12.; If the task is not running start it
13.;
14.       WAIT 30
15.       SELECT_TASK IO_Control MODULE test SEQUENCE io_control
16.       SELECT_TASK Accum_Control MODULE test SEQUENCE
           accum_control
17.       START_TASK IO_Control
18.   END
19.; Tell the IO_Control task to start the main task
20.;
21.   SET :start_seq = :true

Mod: ex_palwr Seq: start EDIT mode
```

Figure A-4Start Button Sequence Example

Cell Control Sequence Example

This example comes from an application that was running the dispense module software. In this example a supervisory sequence is running to handle error recovery and startup of the main robot task when needed. The cell control task is always running during normal operation.



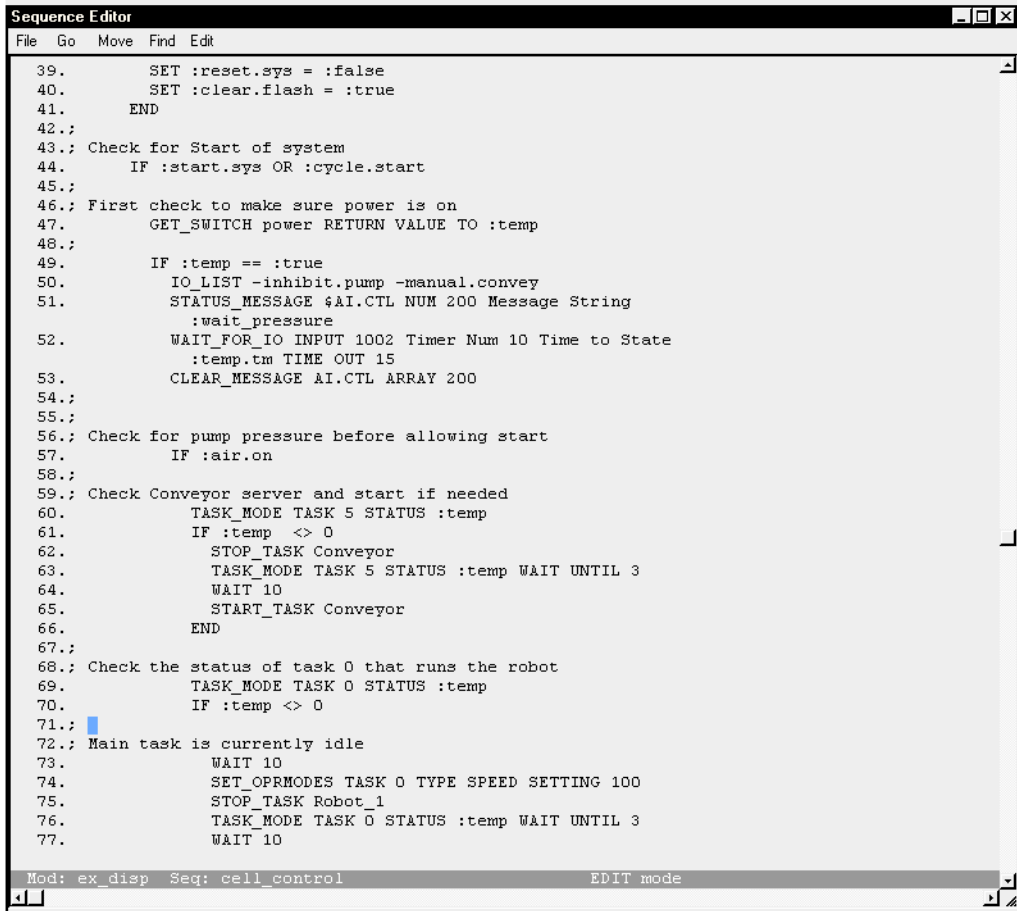
```

Sequence Editor
File Go Move Find Edit

1.; This sequence controls the cell IO relative to Pushbuttons and such
2.;
3.; Most of the variables used here are not defined in the variable database
4.; They are provided here for example only.
5.;
6.; Initialize variables used for control
7.   SET :reset.sys = :false
8.   SET :start_conv = :false
9.   CLEAR_MESSAGE AI.CTL ARRAY 200
10.  IO_LIST -amber.stack.lt -green.stack.lt -in.cycle
    :inhibit.pump -red.stack.lt -fault
11.  IO OUTPUT -2200
12.;
13.; Loop in this sequence
14.  WHILE :true
15.;
16.; Check for reset button to be pressed
17.    IF :reset.sys or :fault.reset
18.;
19.; Check to make E-stop is not pressed
20.    GET_STATE panic.button RETURN_TO :temp
21.;
22.; Make sure station power is on
23.    IF :power.on
24.;
25.; If E-stop is released turn on power
26.      STOP_TASK Robot_1
27.      TASK_MODE TASK 0 STATUS :temp_task3 WAIT UNTIL 3
28.      CLEAR_MESSAGE AI.CTL ARRAY 200
29.      IO OUTPUT -2200
30.      IF temp == :false
31.        SET_SWITCH power EQUAL TO on
32.        IO_LIST -fault -red.stack.lt
33.      ELSE
34.        IO_LIST fault red.stack.lt
35.        SET :error.num = :true
36.        STATUS_MESSAGE $AI.CTL NUM 200 Message String
          :estop.mess Wait_For 2200
37.      END
38.    END
39.    SET :reset.sys = :false
  
```

Mod: ex_disp Seq: cell_control EDIT mode

Figure A-5Cell Control, (First Example Page 1)



```
Sequence Editor
File Go Move Find Edit

39.      SET :reset.sys = :false
40.      SET :clear.flash = :true
41.      END
42.;
43.; Check for Start of system
44.      IF :start.sys OR :cycle.start
45.;
46.; First check to make sure power is on
47.      GET_SWITCH power RETURN VALUE TO :temp
48.;
49.      IF :temp == :true
50.          IO_LIST -inhibit.pump -manual.convey
51.          STATUS_MESSAGE $AI.CTL NUM 200 Message String
           :wait_pressure
52.          WAIT_FOR_IO INPUT 1002 Timer Num 10 Time to State
           :temp.tm TIME OUT 15
53.          CLEAR_MESSAGE AI.CTL ARRAY 200
54.;
55.;
56.; Check for pump pressure before allowing start
57.      IF :air.on
58.;
59.; Check Conveyor server and start if needed
60.          TASK_MODE TASK 5 STATUS :temp
61.          IF :temp <> 0
62.              STOP_TASK Conveyor
63.              TASK_MODE TASK 5 STATUS :temp WAIT UNTIL 3
64.              WAIT 10
65.              START_TASK Conveyor
66.          END
67.;
68.; Check the status of task 0 that runs the robot
69.          TASK_MODE TASK 0 STATUS :temp
70.          IF :temp <> 0
71.;
72.; Main task is currently idle
73.          WAIT 10
74.          SET_OPRMODES TASK 0 TYPE SPEED SETTING 100
75.          STOP_TASK Robot_1
76.          TASK_MODE TASK 0 STATUS :temp WAIT UNTIL 3
77.          WAIT 10

Mod: ex_disp Seq: cell_control EDIT mode
```

Figure A-6Cell Control, (First Example Page 2)

```

77.      WAIT 10
78.      SELECT_TASK Robot_1 MODULE examples SEQUENCE main
79.      START_TASK Robot_1
80.      SET :time.set = :false
81.      SET :start_conv = :true
82.;
83.      END
84.      SET :start.sys = :false
85.      ELSE
86.      SET :error.num = :true
87.      STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :pressure.mess Wait_For 2200
88.      END
89.      ELSE
90.      STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :press_reset
91.      END
92.      END
93.;
94.; Check for cycle stop
95.      IF :stop.sys or :cycle.stop
96.      WAIT_UNTIL :robot.busy == 0 or :power.on == 0
97.      IF :robot.busy == 0
98.      IO OUTPUT :stop.conveyor
99.      STOP_TASK Robot_1
100.     END
101.     SET :stop.sys = :false
102.     IO_LIST -green.stack.lt -in.cycle
103.     END
104.;
105.; Check for guard door or E-stop condition
106.     IF guard.bypass or :error.num == -608 or -guard.closed
107.     IO OUTPUT :stop.conveyor
108.     STOP_TASK Robot_1
109.     IO_LIST -green.stack.lt -in.cycle :stop.conveyor :fault
        :red.stack.lt
110.     STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :remove_product Wait_For 2200
111.     END
112.;
113.; Check for Air pressure off

```

Mod: ex_disp Seq: cell_control EDIT mode

Figure A-7Cell Control (First Example Page 3)

```
Sequence Editor
File Go Move Find Edit

86.      SET :error.num = :true
87.      STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :pressure.mess Wait_For 2200
88.      END
89.      ELSE
90.      STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :press_reset
91.      END
92.      END
93.;
94.; Check for cycle stop
95.      IF :stop.sys or :cycle.stop
96.      WAIT_UNTIL :robot.busy == 0 or :power.on == 0
97.      IF :robot.busy == 0
98.      IO OUTPUT :stop.conveyor
99.      STOP_TASK Robot_1
100.     END
101.     SET :stop.sys = :false
102.     IO_LIST -green.stack.lt -in.cycle
103.     END
104.;
105.; Check for guard door or E-stop condition
106.     IF guard.bypass or :error.num == -608 or -guard.closed
107.     IO OUTPUT :stop.conveyor
108.     STOP_TASK Robot_1
109.     IO_LIST -green.stack.lt -in.cycle :stop.conveyor :fault
        :red.stack.lt
110.     STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :remove_product Wait_For 2200
111.     END
112.;
113.; Check for Air pressure off
114.     IF :air.off
115.     IO OUTPUT :stop.conveyor
116.     STOP_TASK Robot_1
117.     IO_LIST -green.stack.lt -in.cycle :stop.conveyor
118.     STATUS_MESSAGE $AI.CTL NUM 200 Message String
        :pressure.mess Wait_For 2200
119.     END
120.     WAIT 2
121.     END

Mod: ex_disp Seq: cell_control EDIT mode
```

Figure A-8Cell Control (First Example Page 4)

Second Cell Control Example

This example comes from a PalletWare application. This application is similar to the dispense example in that the cell control task handles error recovery and startup of the main robot task. This example is several pages long. It is located in the file 'EX_PALW.MOD'

```

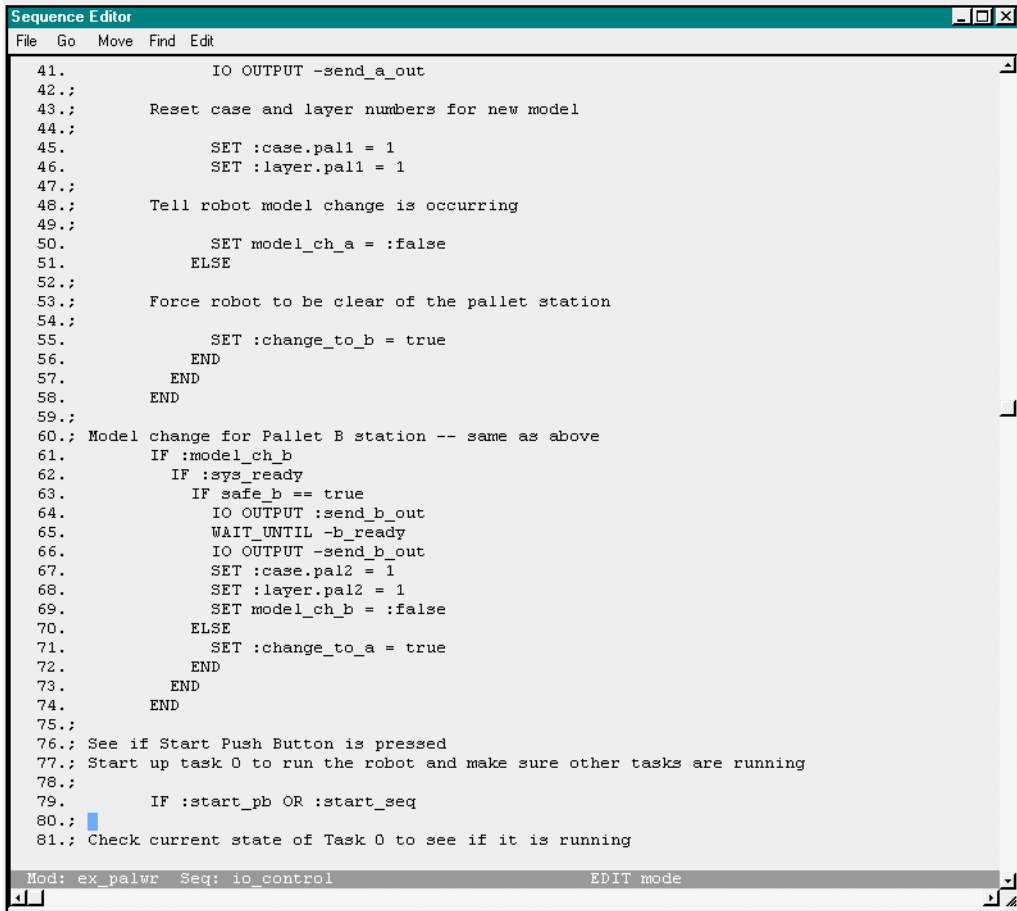
Sequence Editor
File  Go  Move  Find  Edit

1.; This routine controls the operation of the machine with the user buttons
2.; This routine runs in task one. It handles the error recovery,
3.; interface with the PLC and Menu button presses from the operator control
4.; panel. It basically runs in Auto and Manual Mode. Manual Mode allows
5.; operation of the pallet conveyor to exit Pallets out of the machine.
6.;
7.; Initialize variables and outputs
8.;
9.    SET :run_auto = :false
10.   SET :run_manual = :false
11.   IO OUTPUT :robot_ready
12.;
13.; Start continuous loop
14.;
15.   WHILE :true
16.;
17.       IF :auto_man and :auto_seq
18.;
19.;     Auto Mode operation    Tell Plc changing modes
20.;
21.         IF :run_auto == :false
22.             SET :run_auto = :true
23.             SET :run_manual = :false
24.             IO OUTPUT :auto_signal
25.         END
26.;
27.; See if Model Change Occurred Both A & B. Model_ch_a variable from
28.; model selection menu.
29.;
30.     IF :model_ch_a
31.         IF :sys_ready
32.;
33.;       The sys_ready comes from PLC to tell robot system is online.
34.;
35.;       Make sure the robot is clear of the station before sending the
36.;       pallet out of the station.
37.;
38.         IF safe_a == true
39.             IO OUTPUT :send_a_out
40.             WAIT_UNTIL -a_ready
41.             IO OUTPUT -send_a_out

Mod: ex_palwr  Seq: io_control  EDIT mode

```

Figure A-9Cell Control (Second Example Page 1)

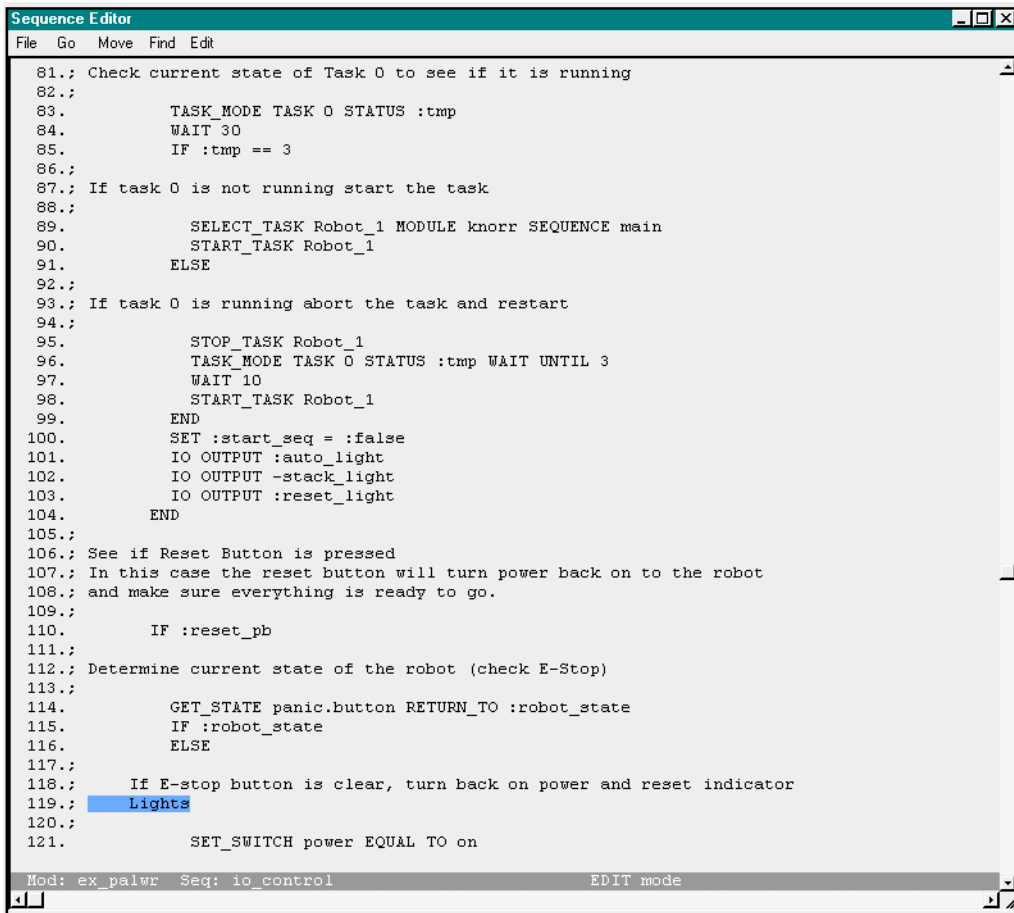


```
Sequence Editor
File Go Move Find Edit

41.          IO OUTPUT -send_a_out
42.;
43.;      Reset case and layer numbers for new model
44.;
45.          SET :case.pal1 = 1
46.          SET :layer.pal1 = 1
47.;
48.;      Tell robot model change is occurring
49.;
50.          SET model_ch_a = :false
51.          ELSE
52.;
53.;      Force robot to be clear of the pallet station
54.;
55.          SET :change_to_b = true
56.          END
57.      END
58.  END
59.;
60.; Model change for Pallet B station -- same as above
61.  IF :model_ch_b
62.    IF :sys_ready
63.      IF safe_b == true
64.        IO OUTPUT :send_b_out
65.        WAIT_UNTIL -b_ready
66.        IO OUTPUT -send_b_out
67.        SET :case.pal2 = 1
68.        SET :layer.pal2 = 1
69.        SET model_ch_b = :false
70.      ELSE
71.        SET :change_to_a = true
72.      END
73.    END
74.  END
75.;
76.; See if Start Push Button is pressed
77.; Start up task 0 to run the robot and make sure other tasks are running
78.;
79.  IF :start_pb OR :start_seq
80.;
81.; Check current state of Task 0 to see if it is running

Mod: ex_palwr Seq: io_control EDIT mode
```

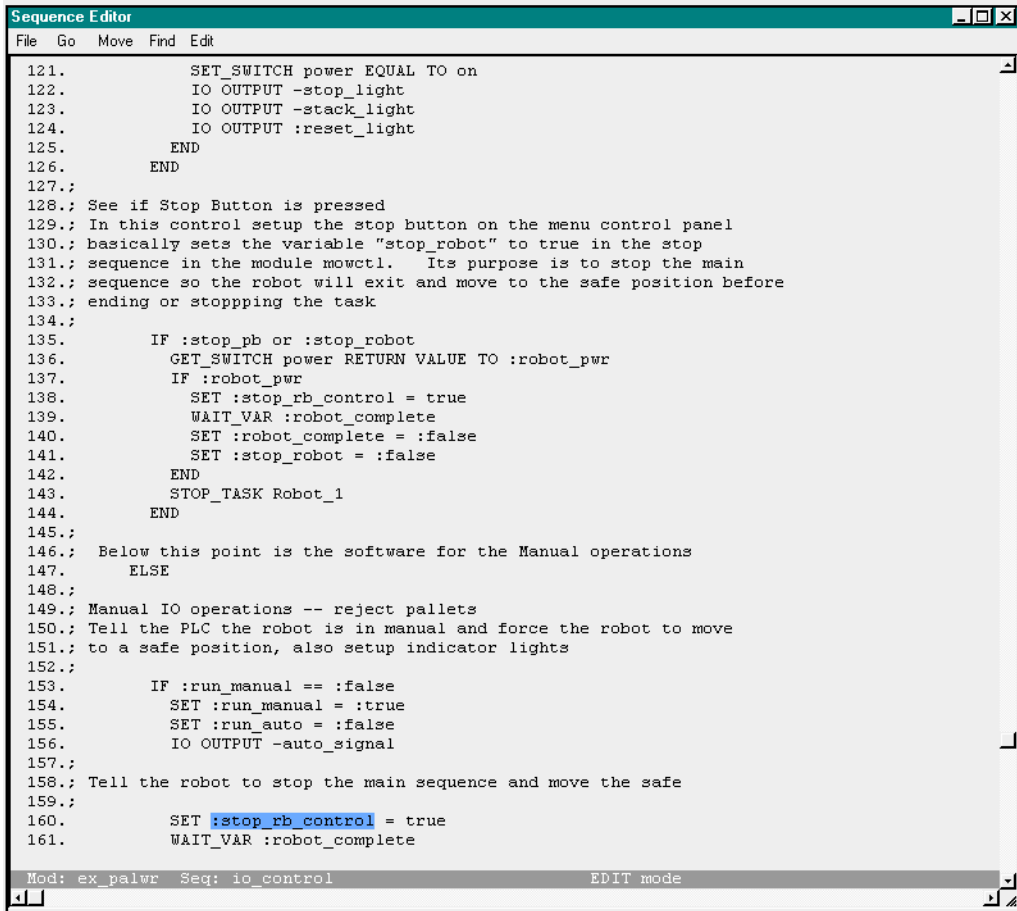
Figure A-10Cell Control (Second Example Page 2)



```
81.; Check current state of Task 0 to see if it is running
82.;
83.      TASK_MODE TASK 0 STATUS :tmp
84.      WAIT 30
85.      IF :tmp == 3
86.;
87.; If task 0 is not running start the task
88.;
89.      SELECT_TASK Robot_1 MODULE knorr SEQUENCE main
90.      START_TASK Robot_1
91.      ELSE
92.;
93.; If task 0 is running abort the task and restart
94.;
95.      STOP_TASK Robot_1
96.      TASK_MODE TASK 0 STATUS :tmp WAIT UNTIL 3
97.      WAIT 10
98.      START_TASK Robot_1
99.      END
100.     SET :start_seq = :false
101.     IO OUTPUT :auto_light
102.     IO OUTPUT -stack_light
103.     IO OUTPUT :reset_light
104.     END
105.;
106.; See if Reset Button is pressed
107.; In this case the reset button will turn power back on to the robot
108.; and make sure everything is ready to go.
109.;
110.     IF :reset_pb
111.;
112.; Determine current state of the robot (check E-Stop)
113.;
114.     GET_STATE panic.button RETURN_TO :robot_state
115.     IF :robot_state
116.     ELSE
117.;
118.; If E-stop button is clear, turn back on power and reset indicator
119.; Lights
120.;
121.     SET_SWITCH power EQUAL TO on

Mod: ex_palwr  Seq: io_control  EDIT mode
```

Figure A-11 Cell Control (Second Example Page 3)



```
Sequence Editor
File Go Move Find Edit

121.          SET_SWITCH power EQUAL TO on
122.          IO OUTPUT -stop_light
123.          IO OUTPUT -stack_light
124.          IO OUTPUT :reset_light
125.          END
126.          END
127.;
128.; See if Stop Button is pressed
129.; In this control setup the stop button on the menu control panel
130.; basically sets the variable "stop_robot" to true in the stop
131.; sequence in the module mowctl. Its purpose is to stop the main
132.; sequence so the robot will exit and move to the safe position before
133.; ending or stopping the task
134.;
135.          IF :stop_pb or :stop_robot
136.          GET_SWITCH power RETURN VALUE TO :robot_pwr
137.          IF :robot_pwr
138.          SET :stop_rb_control = true
139.          WAIT_VAR :robot_complete
140.          SET :robot_complete = :false
141.          SET :stop_robot = :false
142.          END
143.          STOP_TASK Robot_1
144.          END
145.;
146.; Below this point is the software for the Manual operations
147.          ELSE
148.;
149.; Manual IO operations -- reject pallets
150.; Tell the PLC the robot is in manual and force the robot to move
151.; to a safe position, also setup indicator lights
152.;
153.          IF :run_manual == :false
154.          SET :run_manual = :true
155.          SET :run_auto = :false
156.          IO OUTPUT -auto_signal
157.;
158.; Tell the robot to stop the main sequence and move the safe
159.;
160.          SET :stop_rb_control = true
161.          WAIT_VAR :robot_complete

Mod: ex_palwr Seq: io_control EDIT mode
```

Figure A-12Cell Control (Second Example Page 4)

```

Sequence Editor
File  Go  Move  Find  Edit

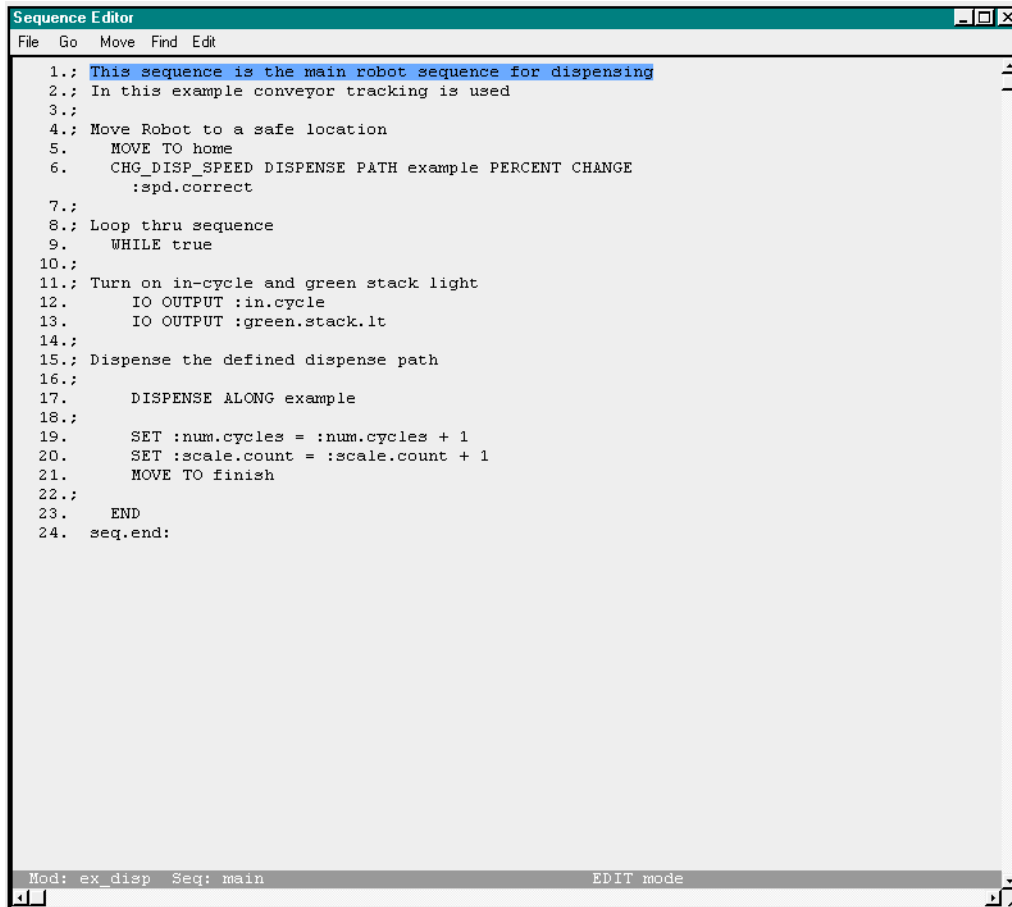
150.; Tell the PLC the robot is in manual and force the robot to move
151.; to a safe position, also setup indicator lights
152.;
153.      IF :run_manual == :false
154.          SET :run_manual = :true
155.          SET :run_auto = :false
156.          IO OUTPUT -auto_signal
157.;
158.; Tell the robot to stop the main sequence and move the safe
159.;
160.          SET :stop_rb_control = true
161.          WAIT_VAR :robot_complete
162.          SET :robot_complete = :false
163.          SET :stop_robot = :false
164.          IO OUTPUT -auto_light
165.      END
166.;
167.; Check reset button for a and exit pallet out of station
168.;
169.      IF :reject_a_pb or :reject_a_seq
170.          IO OUTPUT :send_a_out
171.          WAIT_UNTIL -a_ready
172.          IO OUTPUT -send_a_out
173.          WAIT 100
174.          SET :reject_a_seq = :false
175.      END
176.;
177.; Check the reject B button and send out the pallet
178.;
179.      IF :reject_b_pb or :reject_b_seq
180.          IO OUTPUT :send_b_out
181.          WAIT_UNTIL -b_ready
182.          IO OUTPUT -send_b_out
183.          WAIT 100
184.          SET :reject_b_seq = :false
185.      END
186.      END
187.;
188.; Allow .030 for additional processing of other tasks
189.      WAIT 3
190.      END

Mod: ex_palwr  Seq: io_control  EDIT mode
  
```

Figure A-13Cell Control (Second Example Page 5)

Robot Main Sequence

This sequence controls the motions of the robot system. This example comes from a dispense module application and is used for conveyor tracking. This is located in the 'EX_DISP.MOD' modules file.



The screenshot shows a 'Sequence Editor' window with a menu bar (File, Go, Move, Find, Edit) and a text area containing the following code:

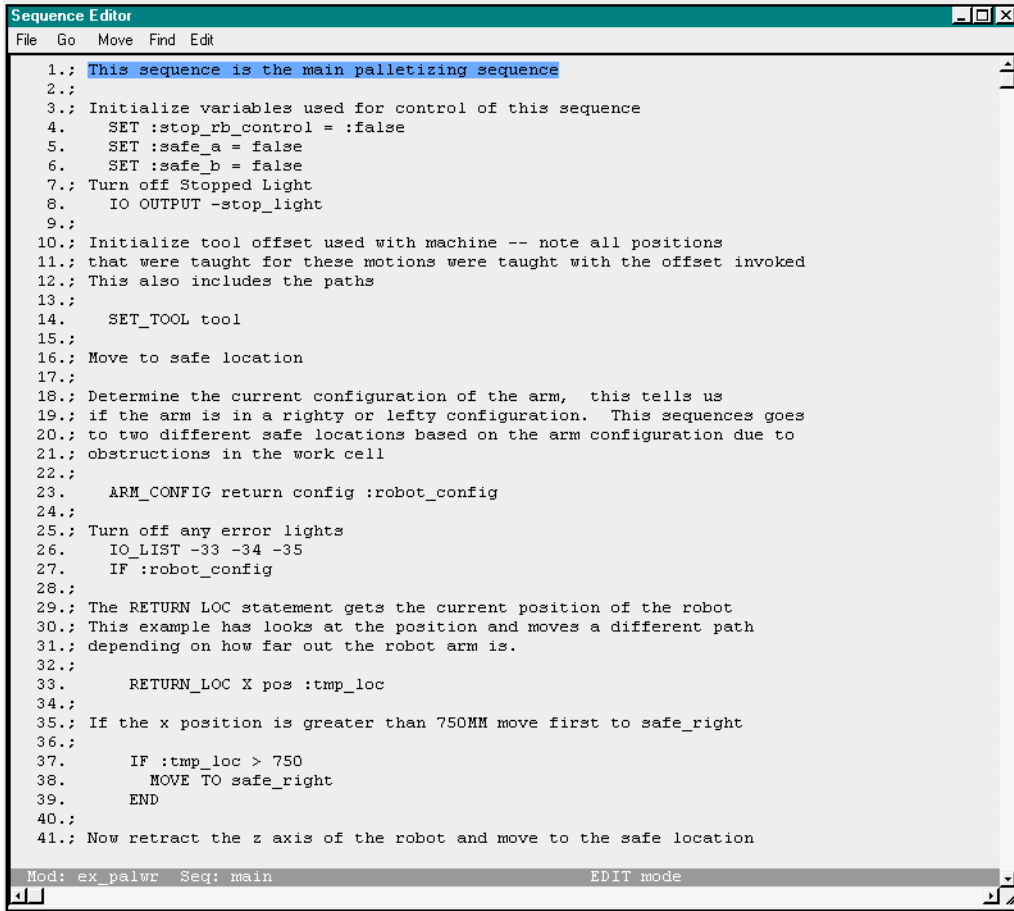
```
1.; This sequence is the main robot sequence for dispensing
2.; In this example conveyor tracking is used
3.;
4.; Move Robot to a safe location
5.    MOVE TO home
6.    CHG_DISP_SPEED DISPENSE PATH example PERCENT CHANGE
       :spd.correct
7.;
8.; Loop thru sequence
9.    WHILE true
10.;
11.; Turn on in-cycle and green stack light
12.    IO OUTPUT :in.cycle
13.    IO OUTPUT :green.stack.lt
14.;
15.; Dispense the defined dispense path
16.;
17.    DISPENSE ALONG example
18.;
19.    SET :num.cycles = :num.cycles + 1
20.    SET :scale.count = :scale.count + 1
21.    MOVE TO finish
22.;
23.    END
24. seq.end:
```

The status bar at the bottom indicates 'Mod: ex_disp Seq: main' and 'EDIT mode'.

Figure A-14Dispense Robot Main Sequence

Main Sequence Example from PalletWare

This example comes from a PalletWare example. It is very useful in showing how to handle Righty and Lefty arm configurations as well as a general guide in PalletWare applications.

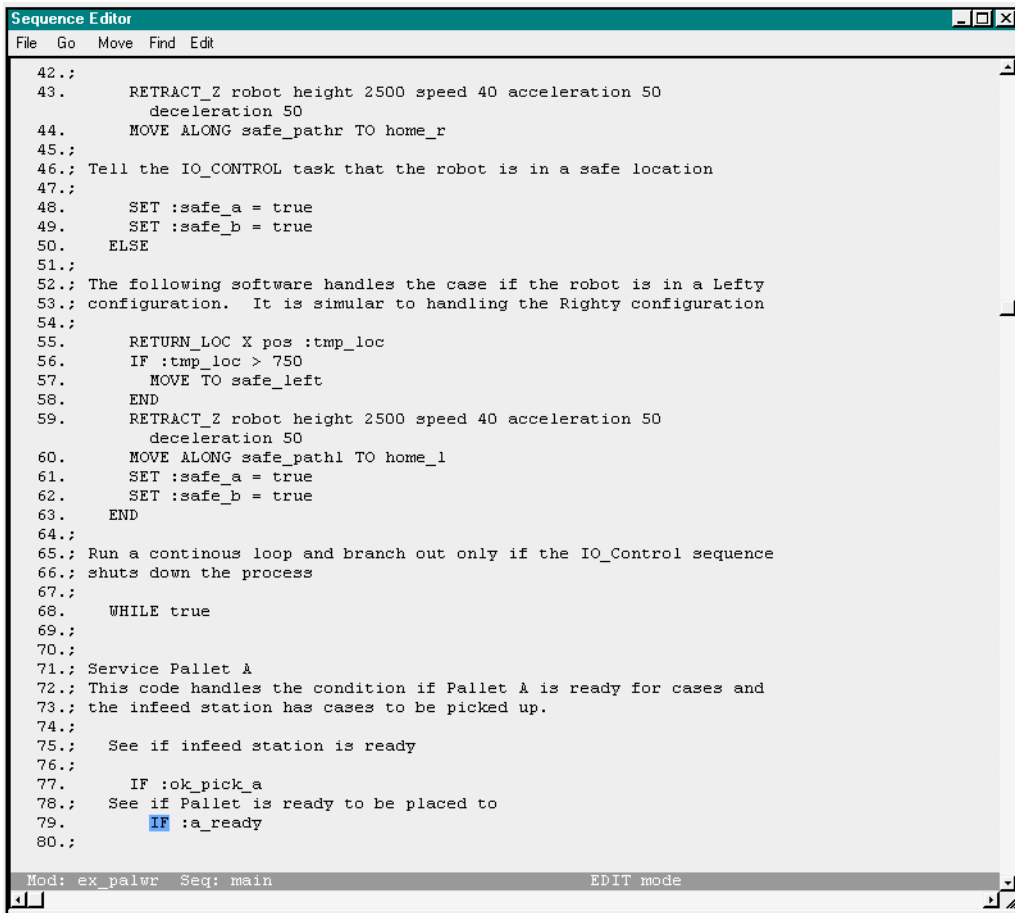


```

Sequence Editor
File Go Move Find Edit
1.; This sequence is the main palletizing sequence
2.;
3.; Initialize variables used for control of this sequence
4.   SET :stop_rb_control = :false
5.   SET :safe_a = false
6.   SET :safe_b = false
7.; Turn off Stopped Light
8.   IO OUTPUT -stop_light
9.;
10.; Initialize tool offset used with machine -- note all positions
11.; that were taught for these motions were taught with the offset invoked
12.; This also includes the paths
13.;
14.   SET_TOOL tool
15.;
16.; Move to safe location
17.;
18.; Determine the current configuration of the arm, this tells us
19.; if the arm is in a righty or lefty configuration. This sequence goes
20.; to two different safe locations based on the arm configuration due to
21.; obstructions in the work cell
22.;
23.   ARM_CONFIG return config :robot_config
24.;
25.; Turn off any error lights
26.   IO_LIST -33 -34 -35
27.   IF :robot_config
28.;
29.; The RETURN LOC statement gets the current position of the robot
30.; This example has looks at the position and moves a different path
31.; depending on how far out the robot arm is.
32.;
33.   RETURN_LOC X pos :tmp_loc
34.;
35.; If the x position is greater than 750MM move first to safe_right
36.;
37.   IF :tmp_loc > 750
38.     MOVE TO safe_right
39.   END
40.;
41.; Now retract the z axis of the robot and move to the safe location
Mod: ex_palwr Seq: main EDIT mode

```

Figure A-15PalletWare Main Sequence (Page 1)

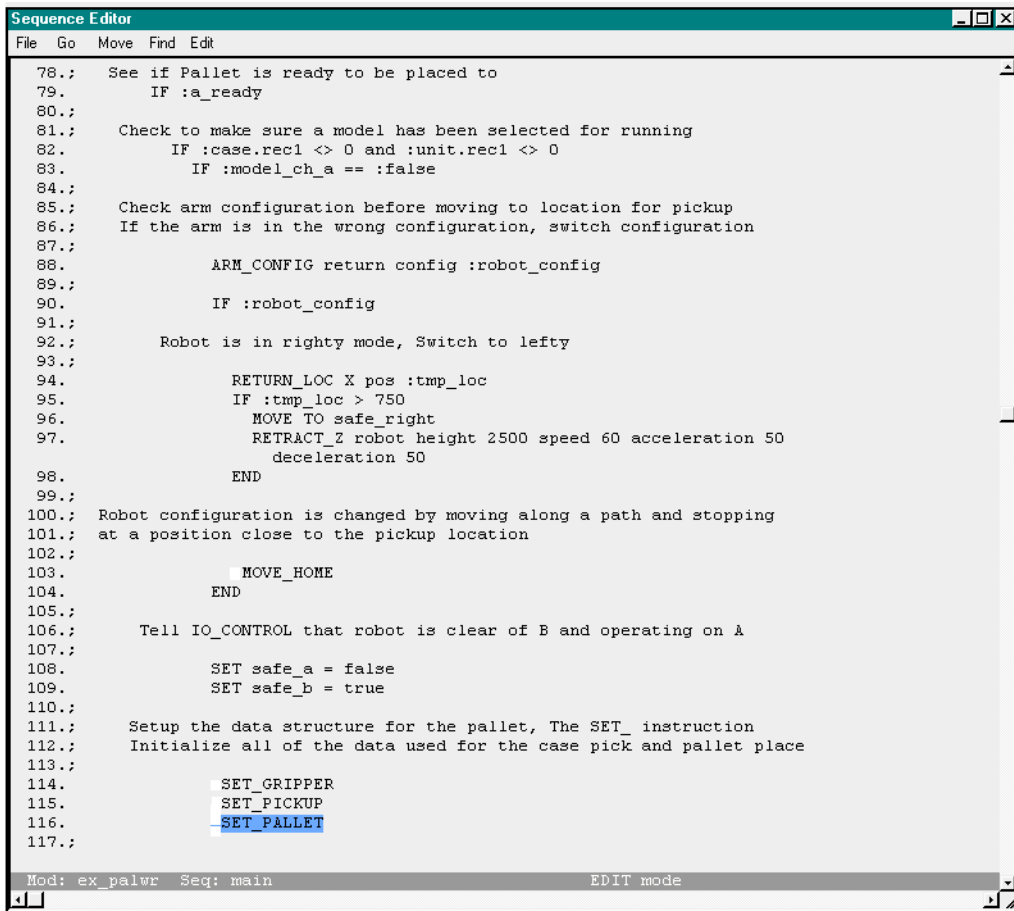


The screenshot shows a 'Sequence Editor' window with a menu bar (File, Go, Move, Find, Edit) and a text area containing the following code:

```
42.;
43.    RETRACT_Z robot height 2500 speed 40 acceleration 50
44.    deceleration 50
45.    MOVE ALONG safe_pathr TO home_r
46.; Tell the IO_CONTROL task that the robot is in a safe location
47.;
48.    SET :safe_a = true
49.    SET :safe_b = true
50.    ELSE
51.;
52.; The following software handles the case if the robot is in a Lefty
53.; configuration. It is similar to handling the Righty configuration
54.;
55.    RETURN_LOC X pos :tmp_loc
56.    IF :tmp_loc > 750
57.    MOVE TO safe_left
58.    END
59.    RETRACT_Z robot height 2500 speed 40 acceleration 50
60.    deceleration 50
61.    MOVE ALONG safe_pathl TO home_l
62.    SET :safe_a = true
63.    SET :safe_b = true
64.    END
65.; Run a continuous loop and branch out only if the IO_Control sequence
66.; shuts down the process
67.;
68.    WHILE true
69.;
70.;
71.; Service Pallet A
72.; This code handles the condition if Pallet A is ready for cases and
73.; the infeed station has cases to be picked up.
74.;
75.; See if infeed station is ready
76.;
77.    IF :ok_pick_a
78.; See if Pallet is ready to be placed to
79.    IF :a_ready
80.;
```

At the bottom of the window, a status bar shows 'Mod: ex_palwr Seq: main' and 'EDIT mode'.

Figure A-16PalletWare Main Sequence (Page 2)



```
78.; See if Pallet is ready to be placed to
79.  IF :a_ready
80.;
81.; Check to make sure a model has been selected for running
82.  IF :case.rec1 <> 0 and :unit.rec1 <> 0
83.  IF :model_ch_a == :false
84.;
85.; Check arm configuration before moving to location for pickup
86.; If the arm is in the wrong configuration, switch configuration
87.;
88.  ARM_CONFIG return config :robot_config
89.;
90.  IF :robot_config
91.;
92.  Robot is in righty mode, Switch to lefty
93.;
94.  RETURN_LOC X pos :tmp_loc
95.  IF :tmp_loc > 750
96.  MOVE TO safe_right
97.  RETRACT_Z robot height 2500 speed 60 acceleration 50
    deceleration 50
98.  END
99.;
100.; Robot configuration is changed by moving along a path and stopping
101.; at a position close to the pickup location
102.;
103.  MOVE_HOME
104.  END
105.;
106.; Tell IO_CONTROL that robot is clear of B and operating on A
107.;
108.  SET safe_a = false
109.  SET safe_b = true
110.;
111.; Setup the data structure for the pallet, The SET_ instruction
112.; Initialize all of the data used for the case pick and pallet place
113.;
114.  SET_GRIPPER
115.  SET_PICKUP
116.  SET_PALLET
117.;
```

Mod: ex_palwr Seq: main EDIT mode

Figure A-17PalletWare Main Sequence (Page 3)

```

Sequence Editor
File Go Move Find Edit

118.; The Stack_ instructions allow the robot to place cases based
119.; on a group, a layer or a pallet
120.;
121.;     STACK_CASE
122.;
123.; Check to see if the pallet is complete and move clear to allow
124.; the pallet to be moved out of the way
125.;
126.;     IF :pallet_a_done
127.;         RETURN_LOC X pos :tmp_loc
128.;         IF :tmp_loc > 750
129.;             MOVE TO safe_left
130.;             RETRACT_Z robot height 2500 speed 60 acceleration 50
131.;                 deceleration 50
132.;         END
133.;         MOVE_HOME
134.;     Tell IO_CONTROL Robot is clear of A
135.;         SET safe_a = true
136.;
137.;     Eject the pallet out of the station
138.;         IO OUTPUT :send_a_out
139.;         WAIT_UNTIL -a_ready
140.;         IO OUTPUT -send_a_out
141.;     END
142.; END
143.; END
144.; END
145.; END
146.;
147.; Check to see if the IO_Control task wants to stop the robot
148.;     IF :stop_rb_control
149.;         GOTO stop
150.;     END
151.;
152.;
153.; Service Pallet station B
154.;
155.; This section of code is similar to servicing station A
156.; It is not commented to any detail for ease of reading
157.;
Mod: ex_palwr Seq: main EDIT mode

```

Figure A-18PalletWare Main Sequence (Page 4)

```

Sequence Editor
File  Go  Move  Find  Edit

118.; The Stack_instructions allow the robot to place cases based
119.; on a group, a layer or a pallet
120.;
121.;     STACK_CASE
122.;
123.; Check to see if the pallet is complete and move clear to allow
124.; the pallet to be moved out of the way
125.;
126.;     IF :pallet_a_done
127.;         RETURN_LOC X pos :tmp_loc
128.;         IF :tmp_loc > 750
129.;             MOVE TO safe_left
130.;             RETRACT_Z robot height 2500 speed 60 acceleration 50
131.;             deceleration 50
132.;         END
133.;         MOVE_HOME
134.;
135.; Tell IO_CONTROL Robot is clear of A
136.;     SET safe_a = true
137.;
138.; Eject the pallet out of the station
139.;     IO OUTPUT :send_a_out
140.;     WAIT_UNTIL -a_ready
141.;     IO OUTPUT -send_a_out
142.;     END
143.;     END
144.;     END
145.;     END
146.;
147.; Check to see if the IO_Control task wants to stop the robot
148.;     IF :stop_rb_control
149.;         GOTO stop
150.;     END
151.;
152.;
153.; Service Pallet station B
154.;
155.; This section of code is similar to servicing station A
156.; It is not commented to any detail for ease of reading
157.;
Mod: ex_palwr  Seq: main  EDIT mode

```

Figure A-19PalletWare Main Sequence (Page 5)


```
Sequence Editor
File Go Move Find Edit

152.;
153.; Service Pallet station B
154.;
155.; This section of code is similar to servicing station A
156.; It is not commented to any detail for ease of reading
157.;
158.    IF :ok_pick_b
159.        IF :b_ready
160.            IF :case.rec2 <> 0 and :unit.rec2 <> 0
161.                IF :model_ch_b == :false
162.; Check arm configuration before moving
163.                ARM_CONFIG return config :robot_config
164.                IF :robot_config == 0
165.; Switch to righty mode
166.                RETURN_LOC X pos :tmp_loc
167.                IF :tmp_loc > 750
168.                    MOVE TO safe_left
169.                    RETRACT_Z robot height 2500 speed 60 acceleration 50
170.                    deceleration 50
171.                END
172.                MOVE_HOME
173.            END
174.            SET safe_a = true
175.            SET safe_b = false
176.            SET_GRIPPER
177.            SET_PICKUP
178.            SET_PALLET
179.            STACK_CASE
180.            IF :pallet_b_done
181.                RETURN_LOC X pos :tmp_loc
182.                IF :tmp_loc > 750
183.                    MOVE TO safe_right
184.                    RETRACT_Z robot height 2500 speed 60 acceleration 50
185.                    deceleration 50
186.                END
187.                MOVE_HOME
188.                SET safe_b = true
189.                IO OUTPUT :send_b_out
190.                WAIT_UNTIL -b_ready
191.                IO OUTPUT -send_b_out
192.            END
193.        END
194.    END

Mod: ex_palwr Seq: main EDIT mode
```

Figure A-20PalletWare Main Sequence (Page 6)

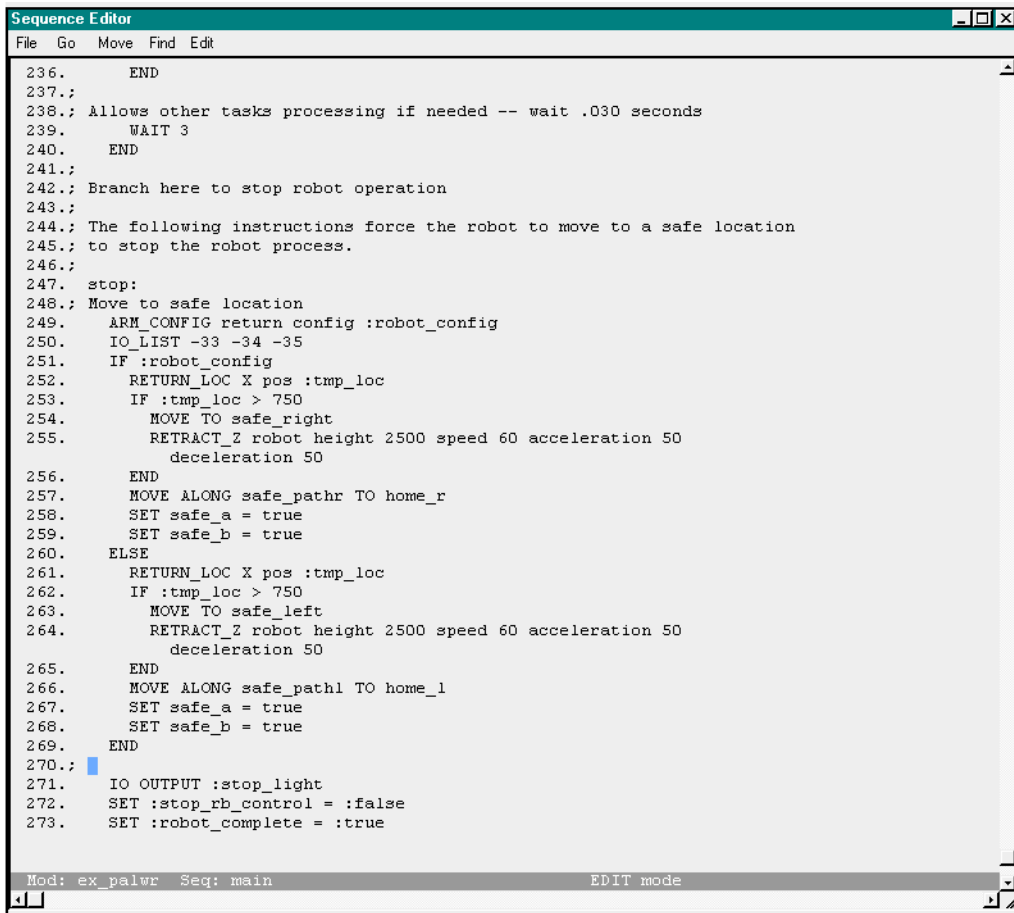
```

Sequence Editor
File  Go  Move  Find  Edit

190.      END
191.      END
192.      END
193.      END
194.      END
195.;
196.; Check to see if robot should be stopped
197.      IF :stop_rb_control
198.          GOTO stop
199.      END
200.;
201.; Clear and move to Lefty for model change
202.;
203.; This section of code forces the robot to be ready to service the other
204.; station during a model change.  This allows the previous pallet to
205.; be ejected as well as allowing the robot to continue offloading station B
206.;
207.      IF change_to_a
208.; Check robot for configuration and switch if needed
209.          ARM_CONFIG return config :robot_config
210.          IF :robot_config
211.; Switch to lefty mode
212.              RETURN_LOC X pos :tmp_loc
213.              IF :tmp_loc > 750
214.                  MOVE TO safe_right
215.                  RETRACT_Z robot height 2500 speed 60 acceleration 50
216.                      deceleration 50
216.              END
217.              MOVE_HOME
218.          END
219.          SET change_to_a = false
220.      END
221.;
222.; Clear and move to righty config
223.      IF change_to_b
224.; Check arm configuration before moving
225.          ARM_CONFIG return config :robot_config
226.          IF :robot_config == 0
227.; Switch to righty mode
228.              RETURN_LOC X pos :tmp_loc
229.              IF :tmp_loc > 750
  
```

Mod: ex_palwr Seq: main EDIT mode

Figure A-21 PalletWare Main Sequence (Page 7)



```
Sequence Editor
File Go Move Find Edit

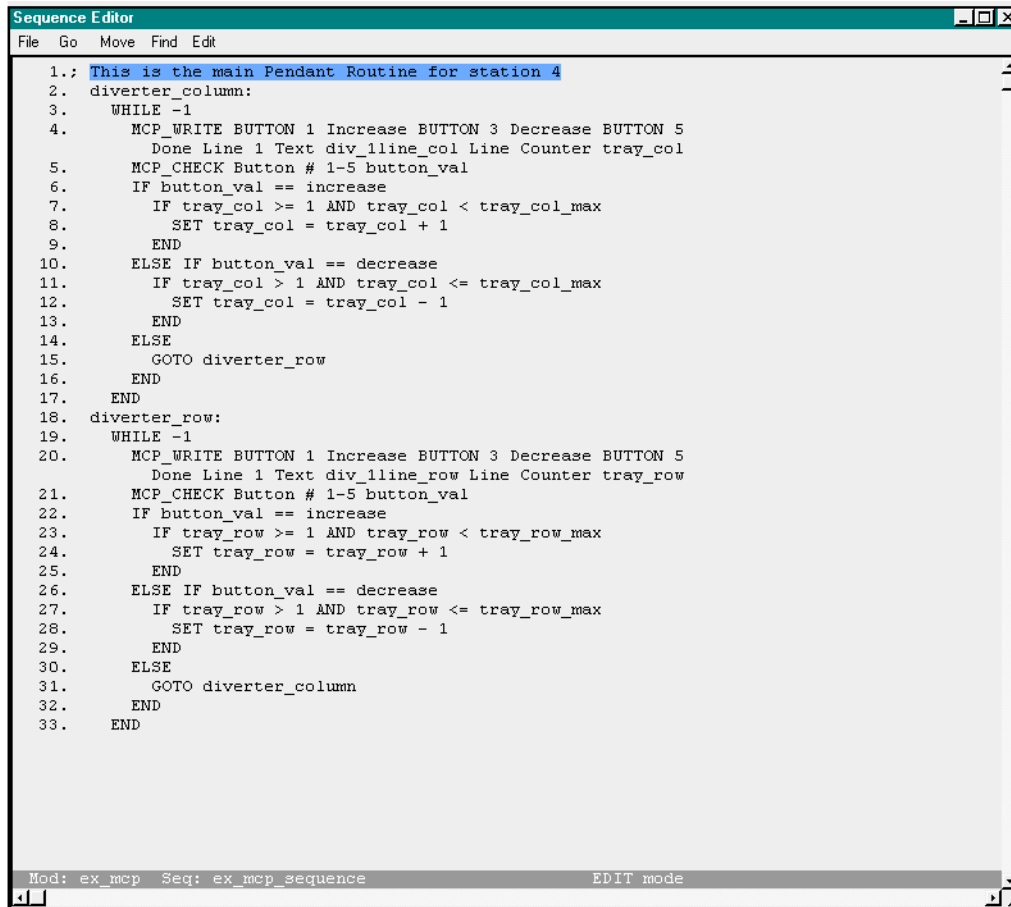
236.      END
237.;
238.; Allows other tasks processing if needed -- wait .030 seconds
239.      WAIT 3
240.      END
241.;
242.; Branch here to stop robot operation
243.;
244.; The following instructions force the robot to move to a safe location
245.; to stop the robot process.
246.;
247. stop:
248.; Move to safe location
249.      ARM_CONFIG return config :robot_config
250.      IO_LIST -33 -34 -35
251.      IF :robot_config
252.          RETURN_LOC X pos :tmp_loc
253.          IF :tmp_loc > 750
254.              MOVE TO safe_right
255.              RETRACT_Z robot height 2500 speed 60 acceleration 50
                deceleration 50
256.          END
257.          MOVE ALONG safe_pathr TO home_r
258.          SET safe_a = true
259.          SET safe_b = true
260.      ELSE
261.          RETURN_LOC X pos :tmp_loc
262.          IF :tmp_loc > 750
263.              MOVE TO safe_left
264.              RETRACT_Z robot height 2500 speed 60 acceleration 50
                deceleration 50
265.          END
266.          MOVE ALONG safe_pathl TO home_l
267.          SET safe_a = true
268.          SET safe_b = true
269.      END
270.;
271.      IO OUTPUT :stop_light
272.      SET :stop_rb_control = :false
273.      SET :robot_complete = :true

Mod: ex_palwr Seq: main EDIT mode
```

Figure A-23PalletWare Main Sequence (Page 9)

MCP Sequence Example

This example demonstrates running a sequence to use the MCP as the interface. This particular example uses the MCP to allow the operator to change the pallet parameters for a standard motionware depalletizing pick and place application. This sequence is available in the 'EX_MCP.MOD' modules file.



```
Sequence Editor
File Go Move Find Edit

1.; This is the main Pendant Routine for station 4
2. diverter_column:
3.   WHILE -1
4.     MCP_WRITE BUTTON 1 Increase BUTTON 3 Decrease BUTTON 5
       Done Line 1 Text div_lline_col Line Counter tray_col
5.     MCP_CHECK Button # 1-5 button_val
6.     IF button_val == increase
7.       IF tray_col >= 1 AND tray_col < tray_col_max
8.         SET tray_col = tray_col + 1
9.       END
10.    ELSE IF button_val == decrease
11.      IF tray_col > 1 AND tray_col <= tray_col_max
12.        SET tray_col = tray_col - 1
13.      END
14.    ELSE
15.      GOTO diverter_row
16.    END
17.  END
18. diverter_row:
19.   WHILE -1
20.     MCP_WRITE BUTTON 1 Increase BUTTON 3 Decrease BUTTON 5
       Done Line 1 Text div_lline_row Line Counter tray_row
21.     MCP_CHECK Button # 1-5 button_val
22.     IF button_val == increase
23.       IF tray_row >= 1 AND tray_row < tray_row_max
24.         SET tray_row = tray_row + 1
25.       END
26.     ELSE IF button_val == decrease
27.       IF tray_row > 1 AND tray_row <= tray_row_max
28.         SET tray_row = tray_row - 1
29.       END
30.     ELSE
31.       GOTO diverter_column
32.     END
33.   END

Mod: ex_mcp Seq: ex_mcp_sequence EDIT mode
```

Figure A-24MCP Example

Quick-Change Documentation **B**

B.1 Introduction and Overview	104
The CHANGE_HAND Statement	104
Installation Procedure	104
B.2 The Quick Change Database Menu	107
Quick-Change Database	107
Quick-Change IO Debug	108
B.3 Quick-Change Operation	112
The CHANGE_HAND Statement	112
B.4 Quick-Change Example	113
Quick-Change Database	113
B.5 Quick-Change Routines	117
ch.set.io()	119
rn.ch.get()	120
rn.ch.put()	121

B.1 Introduction and Overview

The Quick-Change Module is a fully integrated software package designed specifically for use with Adept products. This software module simplifies the programming of changing multiple robot tooling with Adept Aim software. This software is for use with Aim revision 3.1 and higher. This software module is supplied with the AIM Utility software package.

This document includes all the information to implement this software with other Aim Modules. The document is divided into 4 sections: Introduction and Overview, Quick-Change Database, Software Operation, and Routine Dictionary.

The Introduction and Overview section provides information on sequence task statements, software requirement, and hardware requirements.

The section on the Quick-Change Database provides information about the requirements and uses of the database.

The Software Operation section provides information on how to edit and use the Quick-Change database, and provides information on application requirements.

The Routine Dictionary section provides information about the V+ routines that run the Quick-Change Module.

The CHANGE_HAND Statement

The statement performs tool-change operations if quick-change end-effectors are used. With the CHANGE_HAND statement, the robot can be programmed to automatically return a tool to its nest, exchanging it for a new one. A new Aim database called for Quick-Change has been added to allow the positions and I/O signals to be entered.

Installation Procedure

This section describes the Quick-Change Module installation procedure. To simplify the installation, the user should have the **Adept Utility Disk**. The Utility Disk is supplied with all Adept controllers. Because this software is structured in AIM, the system must be equipped with "A" series controller.

Table B-1 lists the files that are included with the Quick-Change Module. The files reside on the **AIM Utilities** diskette; this diskette contains the runtime routines, and the database definition files. The files with ".SQU" filename extensions are "squeezed", *i.e.* all comments have been removed from the files. Squeezed files require less memory when they are loaded than their commented non-squeezed ".V2" counterparts. The squeezed files and their ".V2" counterparts are otherwise identical. In Table B-1, files marked with a "P" are protected files and cannot be read by the user.

To install the Quick-Change Module, perform the steps described below. These steps assume that AIM version 3.1 or later is already copied onto the system's hard drive. The installation procedure copies all the files listed in Table B-1 onto the hard disk drive.

1. Load and execute the DISKCOPY utility from **Adept Utility Disk**. To execute the program, place the Utility Disk in drive A and issue the following monitor commands.

```
LOAD A:DISKCOPY
EX A.DISKCOPY
```

2. Remove the Utility Disk and insert the **AIM Utilities** disk into drive A. Choose the "Copy multiple files" option from the DISKCOPY menu and copy all of the ".V2" and ".SQU" files to the AIM default subdirectory on the hard disk drive. These files must be reside on the AIM default subdirectory during AIM execution or an error will occur. Specifying the wildcards "*.V2" and "*.SQU" for the copy command will copy all of the ".V2" and ".SQU" files to the hard disk. Because some of the files replace standard AIM files, answer "Y" (for yes) to DISKCOPY's *supersede* prompt.

Table B-1
Quick-Change Module Files

Files	Contents
RUN_CH.V2	Commented Quick Change runtime routines.
RUN_CH.SQU	Squeezed Quick Change runtime routines
MENU_CH.SQU	Squeezed menu routines.
QCMOD.OVR	Startup and initializing Quick Change Software.
QUICKCH.MNU	Quick Change menu file.
QC_IO.MNU	Quick Change IO menu file
QUICKCH.RFD	Quick Change database rfd file.
QUICKCH.DB	Quick Change database.
QCICON.DAT	Quick Change menu icons.
ERRORQC.DB	Quick Change error database.
STATQC.DB	Quick Change statement database

The ".DB" , ".RFD", and ".MNU" files of Disk #1 must reside in the same subdirectory as the AIM Base Package ".DB" files. Copy those files to the appropriate subdirectory.

Specify a ".*" diskcopy if the program, database, and database definitions files were placed in the same subdirectory when the AIM Base Package was installed. Again,

supersede existing files with DISKCOPY since some **AIM Utilities** files replace AIM Base Package files.

After successful completion of the steps above, the installation is complete.

B.2 The Quick Change Database Menu

The Quick Change database supports all the needed data for the `Change_Hand` sequence statement. The Quick Change database includes a field that defines the tool nest location (where a quick-change end-effector is acquired and returned), fields for tool signature and nest presence signals, and fields that specify I/O reset routines. The I/O reset programs, executed before returning a tool to its nest or after the tool has been acquired, sets end-effector I/O to default settings. These routines can be used to prevent air lines from blowing when a tool is detached from the robot.

The Quick Change database record form is shown in Figure B-1 and the field descriptions are in Table B-3. Data from each field can be accessed in an applications program by using the V+ variable names listed in Table B-4. The record fields are described below.

The screenshot shows a window titled "Quick-Change Tool (exmpdis)" with a menu bar containing "Go", "Seek", "Edit", and "Help". The main area contains several input fields and sections:

- Top left: A small icon of a robot arm and a text field containing "nest2". Below it, "2 of 2" is displayed.
- Top right: "Device:" followed by a field containing "1", and a date/time field containing "28-Jan-97 14:17".
- Section: "Quick-Change Information" (bordered box)
 - Hand number: field containing "2"
 - Pickup routine: field containing "ch.set.io"
 - Tool-on-stand signal: field containing "2044"
 - Putdown routine: field containing "ch.set.io"
- Section: "Quick-Change Tool I/O" (bordered box)
 - Tool Attached: field containing "2450"
 - Total Tool Signatures: field containing "2"
 - First Signature Bit: field containing "2500"
 - Sub-section: "Outputs" (bordered box)
 - Acquire Tool: field containing "2030"
 - Release Tool: field containing "2031"
 - Gripper Open: field containing "2030"
 - Gripper Close: field containing "2031"
 - Sub-section: "Delays" (bordered box)
 - Acquire Tool Delay: field containing "0.50"
 - Release Tool Delay: field containing "0.50"
- Bottom: "Tool Location:" followed by a field containing "nest2_loc"

Figure B-1
Quick-Change Database Record Form

Quick-Change Database

Data for the ***Change_Hand*** statement is stored in the Quick-Change database. Important fields in the Quick-Change database include the tool nest location (where the end-effectors are acquired and replaced), I/O and Delay parameters, and the I/O reset routines.

The tool nest locations are specified with the standard AIM location database. Double click on the location field and the software will branch to the location record. Like other

location fields, the location is taught by moving the mouse pointer to the Here or Teach Button and clicking the mouse. If Teach was selected the standard Aim teach pendant routines are used to define the location. Approach and depart fields must also be defined for this location.

The Quick-Change database has several fields that are used for Input Signals to operate the change hand mechanism. The list of possible inputs include: *Tool-On-Stand*, *Tool Attached*, *Total Tool Signatures*, *First Signature Bit*. The *Tool-On-Stand* signal determines if the end-effector is in the nest, this input signal is required for operation of the software. The optional *Tool Attached* signal allows the software to check for the presence of the end-effector on the robot arm. The optional *Total Tool Signatures* and *First Signature Bit* allow the system to be configured to check which Hand Number is attached to the robot. The *Total Tool Signatures* field defines the number of bits required to sense the largest Hand Number available. The *First Signature Bit* defines the first input signal of the consecutive signals to define the Hand Number. The optional *Hand Number* field allows the user to define hand numbers to each end-effector. This field is compared to the sensed number when the tool is picked up to assure the proper tool is available.

The available Quick-Change outputs fields are *Acquire Tool*, *Release Tool*, *Gripper Open*, *Gripper Closed*. The *Acquire Tool* and *Release Tool* output signals operate the change hand mechanism to pick-up and release the end-effector. The *Gripper Open* and *Gripper Closed* signals are both turned off when the end-effector is picked-up or released to assure no open air lines.

The *Acquire* and *Release Delays* are set for the end-effector pick-up mechanism actuation time during the pick-up and release cycles. These values are to be entered in seconds.

The I/O reset routines are used to set default I/O settings for the quick-change end-effectors. Specify the optional programs in the *pickup routine* and *putdown routine* fields in the Quick-Change database. The pickup routine is called after acquiring a new end-effector and the putdown routine is called before returning an end-effector that is attached to the robot. For dispensing applications, the putdown routine might be used to make sure that the dispensing gun is turned off before placing the end-effector back into its nest. The I/O routines must conform to the calling sequence described below:

Quick-Change IO Debug

The Quick-Change software includes an additional menu for debugging I/O and to allow manual operation of the change hand mechanism. This menu can be found in the IO pull-down menu under the name QC CONTROLS. The QC CONTROLS menu also shows the binary inputs for the Hand Number in a LED type format with the least significant bit to the right side.

Table B-2 Quick-Change Database Fields

name	String (15 characters)
	A standard AIM name that uniquely identifies this tool for the tool changing utility. This name is referenced in the change_hand routine.
update date	Date
	The date and time when this record was last modified. This field is automatically set to the current date whenever information is changed by an operator.
tool-on-stand signal	Integer

The number of the input signals indicating whether or not the corresponding quick-change end-effector is in its nest. If the value is negative, the logic is inverted. If the signal number is set to zero, no switch is assumed.

hand number Integer

Optional tool signature value that uniquely identifies an end-effector. This enables the tool acquisition and return routines to check that the proper tool is attached to the arm before continuing the Change_Hand sequence.

tool attach Integer

Optional tool attach database field specifies an input signal number that senses whether the tool is attached to the manipulator's wrist.

pickup routine String (15 characters)

A standard name that specifies the name of the subroutine that is called to set default end-effector I/O settings after acquiring an end-effector from its nest.

putdown routine String (15 characters)

A standard name that specifies the name of the subroutine that is called to set default end-effector I/O settings before returning an end-effector to its nest.

total signatures Integer

Optional total signatures database field specifies the number of bits required corresponding to the number of tools to be used in the system.

first signature Integer

Optional first signature database field specifies the first input signal number relative to the number of signatures specified in the total signatures field.

acquire tool Integer

The acquire tool database field specifies the output signal that is activated to lock the tool to the quick-change wrist.

release tool Integer

The release tool database field specifies the output signal that is to be activated to release the tool from the manipulator's wrist.

acquire delay Real

The acquire delay database field specifies the amount of time to be delayed after the acquire signal has been activated before departing from that position.

release delay Real

The release delay database field specifies the amount of time after the release signal has been activated before departing from the nest position.

gripper open Integer

Optional gripper open database field specifies a gripper signal to be turned off before releasing or acquiring the tool.

gripper closed Integer

Optional gripper closed database field specifies a gripper signal to be turned off before releasing or acquiring a tool.

Table B-3 QUICK-CHANGE Database Record Definition

#	Field Name	Data Type	Size	Sort	Array	User
0	name	string	15	-1		
1	update date	date	4			
2	device	integer	2			
3	location name	string	15			
4	[location]	integer	2			
5	tool-on-stand signal	integer	2			
6	hand number	integer	2			
7	pickup routine	string	15			
8	put-down routine	string	15			
9	tool attach	integer	2			
10	total signatures	integer	2			
11	first signature	integer	2			
12	acquire tool	integer	2			
13	release tool	integer	2			
14	acquire delay	real	4			
15	release delay	real	4			
16	gripper open	integer	2			
17	gripper closed	integer	2			

Table B-4 Quick-Change Database Variable Name

Variable Name	Interpretation
qc.db	Quick-Change database number
cc.name	Field number for <i>name</i>
cc.update	Field number for <i>update date</i>
cc.device	Field number for <i>device</i>
qc.on.stand	Field number for <i>tool-on-stand signal</i>
qc.hand.num	Field number for <i>hand number</i>
qc.get.rtn	Field number for <i>pickup routine</i>
qc.put.rtn	Field number for <i>put down routine</i>
qc.tool.attch	Field number for <i>tool attach</i>
qc.total.sig	Field number for <i>total signatures</i>
qc.first.sig	Field number for <i>first signature</i>
qc.acquire.tool	Field number for <i>acquire tool</i>
qc.release.tool	Field number for <i>release tool</i>
qc.acquire.del	Field number for <i>acquire delay</i>
qc.release.del	Field number for <i>release delay</i>
qc.grip.open	Field number for <i>grripper open</i>
qc.grip.close	Field number for <i>grripper closed</i>
qc.loc.name	Field number for <i>location record name</i>
qc.location	Field number for <i>location record number</i>

B.3 Quick-Change Operation

This section describes the operation of the Quick-Change Module. First, the **Change_Hand** statement is reviewed. Next, details about the database menu and field inputs are discussed. Finally, an example using the Quick-Change Software is reviewed.

The CHANGE_HAND Statement

The statement performs automatic tool exchanges. The statement's syntax is:

```
CHANGE_HAND({{APPROACH -path-} OLD.TOOL -tool- {DEPART -path-}}  
{{APPROACH -path-} NEW.TOOL -tool- {{DEPART -path-}}  
{IF SIG ON -constant-}
```

The statement performs the following series of steps:

1. If an **old** tool is specified and is attached to the robot, place the tool in its nest. First, approach the old tool nest using the optional approach path. Next, move to the nest location and detach the hand. Finally, move away from the nest along the depart path if a depart path is defined.
2. If a **new** tool is specified and the hand is not already attached, pick up a hand at its nest location. First, approach the nest following the optional approach path. Next, move to the nest location and attach the hand. After picking up the tool, depart along the transit path if a depart path is specified.
3. If the optional IF SIG ON is specified, the statement will be executed if the specified signal is on. If the signal is off the statement will be by-passed.
4. The -path- arguments reference records in the Path database, specifying standard AIM paths. The Quick-Change argument is linked to the Quick-Change database records which contains the tool nest locations.

Note that the **CHANGE_HAND** statement can be programmed to simply return an end-effector, acquire an end-effector, or perform an entire quick-change task. If tool signature and nest presence signals are incorporated with the hardware, the **CHANGE_HAND** statement uses these signals to verify that the correct end-effector is attached to the robot and that the correct tool is in a tool nest before the robot exchanges tools. If an error condition in the workcell exists (e.g. an end-effector is not released into its nest), the robot stops and the operator is notified of the error. This prevents potential crashes.

B.4 Quick-Change Example

To review the topics discussed in this chapter, consider an application that requires 3 different end-effectors. This example will show the database requirements for 2 of the 3 tools, and show the sequence statements to pick-up tool 1, release tool 2, and finally release Tool #1 and pickup Tool #2. For this example, we are assuming all the optional tool inputs are being used, also the provided sample pickup and putdown routines are used.

Before editing the Quick-Change database, determine the required signal numbers to be used. For this example, we are using the following signals:

Table B-5 Change_Hand Example Tool I/O

Inputs	Signals	Comments
Tool-On-Stand signal (gripper #1)	1001	
Tool-On-Stand signal (gripper #2)	1002	
Tool Attached signal	1003	
First Signature Bit Signal	1004	Two bits are required for 3 tools
Outputs		
Aquire Tool signal	-2031	Internal signal for Adept arm valves
Release Tool signal	2031	
Gripper Open signal	9	
Gripper Closed signal	10	

Quick-Change Database

The Quick-Change database records required for 2 of the 3 end-effectors are shown in Figure B-2 and Figure B-3. This record is created during database editing, or automatically by the AIM linker during sequence start-up if the tool is specified in the sequence database. Important fields used by the **CHANGE_HAND** statement include:

- Location field. This is where the tool is stored when not attached to the robot. The robot executes a complete approach and depart sequence to acquire and replace the tool

- Pickup routine. This program is executed after the end-effector is picked up from its nest. The program might set the tool signal low so that the tool is not actuated when the tool is first attached to the robot. System customizers typically will write this program; the required calling sequence is described in Appendix B.
- Hand number. If there are multiple quick-change end-effectors in the workcell and tool signature signals are used, this parameter uniquely identifies this tool. The hand number is used to verify that the robot picks up the correct tool.
- I/O signal definition. Both end-effector Quick-Change records show the signals that were described in the previous section.

Sequence Statements

The following **CHANGE_HAND** statements are used for the application described above. The first sequence statement will pick-up tool 1 from the tool stand.

```
CHANGE_HAND NEW.TOOL end_effector_1
```

The second sequence statement will place tool 2 back into the tool stand.

```
CHANGE_HAND OLD.TOOL end_effector_2
```

The third sequence statement will place tool 1 back and pick-up tool 2 from the tool stand.

```
CHANGE_HAND OLD.TOOL end_effector_1  
NEW_TOOL end_effector_2
```

Quick-Change Tool (test)

Go Seek Edit Help

end_effector_1

Device: 1

25-Feb-97 16:52

1 of 2

Quick-Change Information

Hand number: 1

Pickup routine: ch.set.io

Tool-on-stand signal: 1001

Putdown routine: ch.set.io

Quick-Change Tool I/O

Tool Attached: 1003

Total Tool Signatures: 2

First Signature Bit: 1004

Acquire Tool: -2031

Release Tool: 2031

Gripper Open: 9

Gripper Close: 10

Delays

Acquire Tool Delay: 0.50

Release Tool Delay: 0.50

Tool Location: tool_1

Figure B-2
Quick-Change Database Record End-Effector 1

The screenshot shows a window titled "Quick-Change Tool (test)" with a menu bar containing "Go", "Seek", "Edit", and "Help". The main area displays the following information:

- Top left: An icon of a gripper and the text "end_effector_2" in a blue box, with "2 of 2" below it.
- Top right: "Device: 1" and a timestamp "25-Feb-97 16:51" in a box.
- Quick-Change Information** section:
 - Hand number: 2
 - Pickup routine: ch.set.io
 - Tool-on-stand signal: 1002
 - Putdown routine: ch.set.io
- Quick-Change Tool I/O** section:
 - Tool Attached: 1003
 - Total Tool Signatures: 2
 - First Signature Bit: 1004
 - Outputs** sub-section:
 - Acquire Tool: -2031
 - Release Tool: 2031
 - Gripper Open: 9
 - Gripper Close: 10
 - Delays** sub-section:
 - Acquire Tool Delay: 0.50
 - Release Tool Delay: 0.50
- Bottom: "Tool Location: tool_2" in a blue box.

Figure B-3
Quick-Change Database Record End-Effector 2

B.5 Quick-Change Routines

The routines listed in Table B-6 are documented in this chapter and can be used by AIM system customizers. Programs marked with a "P" are protected programs which cannot be edited or modified. Edits to unprotected files should be made to the commented ".V2" versions of the files. The files should then be squeezed to their ".SQU" versions for loading with AIM.

Table B-6
Documented Routines

Program Name	Autoloaded File	Commented File	Purpose
ch.reset.io	RUN_CH.SQU	RUN_CH.V2	Sample tool.change I/O program
rn.ch.get	RUN_CH.SQU	RUN_CH.V2	Tool acquire runtime primitive
rn.ch.put	RUN_CH.SQU	RUN_CH.V2	Tool replace runtime primitive

Each routine documented in this section is presented on a separate page, in alphabetical order. The dictionary page for each routine contains the following sections:

Calling Sequence

The format of the V CALL instruction for the routine is shown.

Function

This is a brief statement of the routine's function.

Usage Considerations

This section points out any special considerations associated with the routine.

Input Parameters

Each of the input parameters in the calling sequence is described in detail. For parameters that have a restricted range of values, the allowable range is specified.

Output Parameters

Each of the output parameters in the calling sequence is described in detail.

Global Variables

Global variables accessed by the routine are described.

Details

A complete description of the routine and its use is given.

Related Routines

Other AIM routines that are related to the routine's function are listed.

Calling Sequence

ch.set.io(error)

Function

Sample I/O reset routine specified in the modified Tool database for use with the CHANGE_HAND statement. The program sets output signals to default values before replacing or after acquiring an end-effector.

Input Parameters

None.

Output Parameters

<i>error</i>	Real variable that receives the standard AIM operator response code.
--------------	--

Details

Used in conjunction with the CHANGE_HAND statement, this routine is called after acquiring a tool (if specified as a "pickup routine") or before replacing a tool (if defined as a "putdown routine") to set output signals to their default values. This can be used, for example, to prevent air lines from blowing when a tool is detached.

Calling Sequence

rn.ch.get(app.path.rec, ref, dep.path.rec, error)

Function

Runtime primitive routine for acquiring a hand from its stand. The CHANGE_HAND statement uses this primitive.

Usage Considerations

Assumes the appropriate Tool database record is open.

Input Parameters

<i>app.path.rec</i>	Real value, variable, or expression that defines the Path database number for the optional path to follow while approaching the tool nest location.
<i>ref</i>	Transformation variable, function, or compound transformation specifying a reference frame. If the tool nest locations are defined frame relative, they are defined with respect to <i>ref</i> .
<i>dep.path.rec</i>	Real value, variable, or expression that defines the Path database record number for the optional path to follow while departing from the assembly location.

Output Parameters

<i>error</i>	Real variable that receives the standard AIM operator error response code.
--------------	--

Details

This routine acquires a tool from its tool nest. The program performs the following steps:

1. If there is an approach path specified, begin moving along it.
2. Move to the approach location.
3. If tool signature bits and nest presence signals are used, check to make sure the end-effector is in its nest and verify that there is no hand attached to the robot.
4. Move to the tool acquire location and activate the hand acquire signals.
5. Depart from the pickup location.
6. Make sure the the correct tool is attached to the robot and the tool nest is empty.
7. Execute the "pickup routine" to initialize gripper I/O.
8. If a departure path is specified, move along it.

Related Routines

rn.ch.put

Calling Sequence

rn.ch.put(app.path.rec, ref, dep.path.rec, error)

Function

Runtime primitive to return a tool to its nest. The **CHANGE_HAND** statement uses this primitive.

Usage Considerations

Assumes the appropriate Tool database record is open.

Input Parameters

<i>app.path.rec</i>	Real value, variable, or expression that defines the Path database record number for the optional path to follow while approaching the tool nest location.
<i>ref</i>	Transformation variable, function, or compound transformation specifying a reference frame. If the tool nest locations are defined frame relative, they are defined with respect to <i>ref</i> .
<i>dep.path.rec</i>	Real value, variable, or expression that defines the Path database record number for the optional path to follow while departing from the assembly location.

Output Parameters

<i>error</i>	Real variable that receives the standard AIM operator error response code.
--------------	--

Details

This program returns an end-effector to its tool nest. The approach and depart heights used by this routine are the depart and approach heights, respectively, specified in Tool database. This prevents the robot crashes in the workcell.

The program performs the following steps:

1. If no hand is attached to the arm, or the hand is already in its nest, exit.
2. If an approach path is specified, move along it.
3. Move to the approach location.
4. If tool signature and nest presence signals are used, make sure the correct tool is attached to the arm and that no tool is sitting in the nest.
5. Execute the user specified "putdown routine." This sets the tool's default I/O signals.
6. Move to the tool nest location and activate the detach signals.
7. Depart and verify that there is no tool attached to the arm and that the hand is in its tool nest.
8. If a departure path is specified, begin moving along it.

Related Routines

rn.ch.get

Event I/O Manager

C

C.1 Introduction and Overview	124
C.2 Event Monitor Editing	124
C.3 Event Initialization Database	126
C.4 Running the Event Monitor	126

C.1 Introduction and Overview

Provided with the AIM Utilities software is a utility to allow the user to monitor I/O events that occur. This software will allow you to look for the event to occur and monitor the time during the occurrence. Time limits can be set and an alarm will be generated when the event exceeds the limits. This software has a logging feature which will allow the user to specify a description and a file name.

C.2 Event Monitor Editing

This section describes the menus used for adding events to the event monitor software package. You can reach this menu by selecting the **EDIT** pulldown and the **Event** option on the pulldown list.

Edit ➔ Event

The figure below is the Event monitor database page.

The screenshot shows the 'Event *Global*' window with a menu bar (Go, Seek, Edit, Help). The main area contains configuration fields for an event named 'alarm_1' (1 of 4). The 'Enable' checkbox is checked (marked with 1). The 'Auto RST' checkbox is unchecked. The 'INPUT to MONITOR' section has 'Start' (2040, marked with 2) and 'Stop' (2034) fields. The 'TIME RANGE' section has 'Min' (0.50, marked with 3) and 'Max' (3.00) fields. The 'OUTPUT RESULT' section (marked with 4) shows 'Alarm' (35) and 'Result' (3.02). An 'Event Timeout' field is present. The 'Des' field (marked with 5) contains 'Etch Conveyor Sync Alarm', and the 'File' field contains 'sync.log'. A 'Log' checkbox is at the bottom right.

Figure C-1 Event Monitor Database

① **Enable Selection:**

This check box will allow the event to be monitored when the Event Task is running in AIM.

Auto RST:

This check box will reset the alarm after the stop input has been set. This can reduce operator responses.

② **Start Input:**

The start input field is the input signal number to monitor for the event process to start. This will begin the timer to check for the proper event to occur.

Stop Input:

The stop input field will monitor the event until the input signal specified has turned into a true state. When this occurs the event monitoring will stop until the next start process has begun.

③ *Min Time Range:*

This field is used to set the minimum time the event process can occur. Events that occur in a shorter time will be flagged with an alarm.

Max Time Range:

This field is used to set the maximum time the event can occur. Events that occur in a longer period of time will be flagged with an alarm.

④ *Alarm Output:*

This output signal number will be turned on when a event exceeds the timing parameters.

Result Displays:

There are two displays that are also shown based on the last event occurrence. The first display shows the actual event time that occurred. The second display shows the state message based on the last event.

⑤ *Event Alarm Logging:*

Alarm data can be logged to a specified file with a description statement provided in this database. The check box needs to be turned on to enable this feature.

C.3 Event Initialization Database

The event software provides an initialization database for setting default parameters. Currently only 2 items exist in this database. You can reach this menu by selecting the **SETUP** pulldown and the **Initialization Data** option. Then you must select the **EVTINLDB** option from the resulting selection window.

Setup ➡ Initialization Data ➡ evtini.db

The following 2 records will appear:

1. Log Delimiter - This determines what characters get placed between event text data in the log file that is specified. Currently the default is 9 which places a TAB between text data.
2. Statement_EVT - This record loads the statement database called **statevt.db**. Currently no additional statements are provided with this software package.

C.4 Running the Event Monitor

The event monitor is a server that runs in task 18 as provided with this software package. This server needs to be executed by the user of the event monitor to be operational.

HPGL Graphics Module **D**

D.1 Introduction and Overview	128
-------------------------------------	-----

D.1 Introduction and Overview

The HPGL Plug-In allows the user to import Corel graphics files into a AIM menu static graphic area. Below is a set of instructions provided with this utility to load a HPGL file.

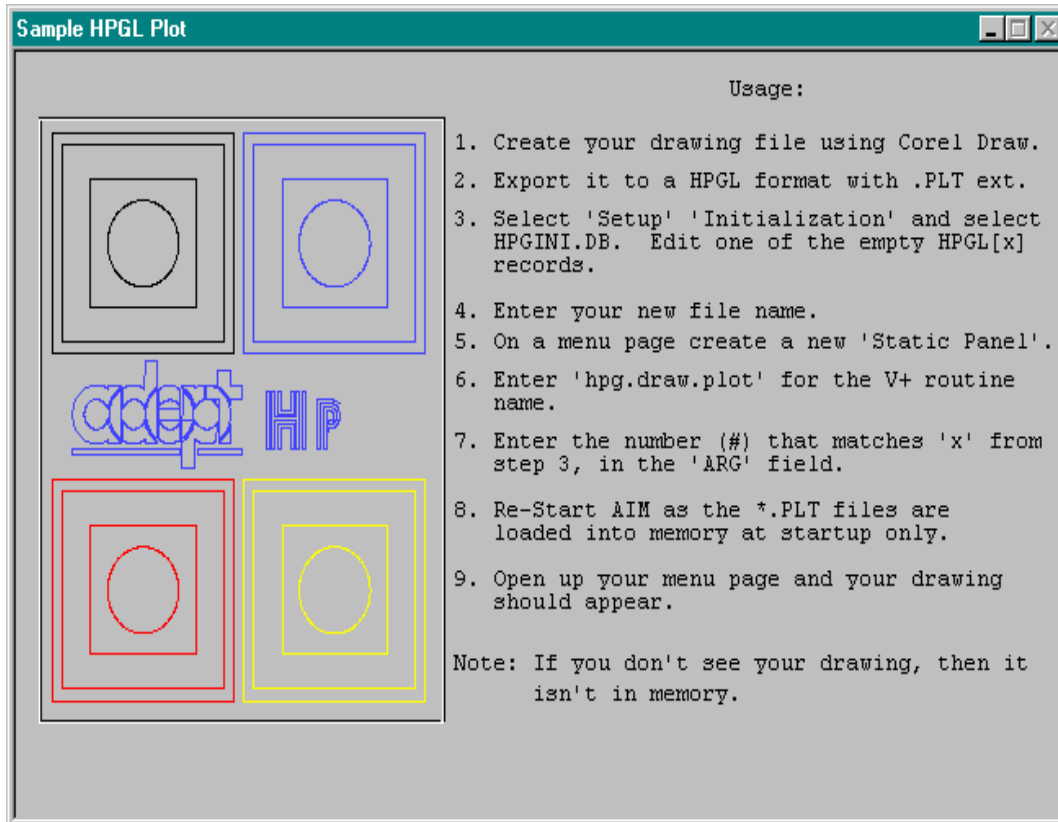


Figure D-1HPGL Instructions

ConnectWare Module

E

E.1 Introduction and Overview	130
E.2 Sequence Statements	130
DDE_CONNECT	130
DDE_DISCONNECT	130
DDE_READ	131
DDE_WRITE	131

E.1 Introduction and Overview

The ConnectWare is a fully integrated software package designed specifically for use with Adept Controllers. This software module allows network communications between multiple Adept Controllers. This software uses the DDE interface that is supplied with the AIM product. Generally, the PC or other device has a DDE server for the communication. This package transfers AIM Variable data via sequence instructions for the communication. This software is for use with Aim revision 3.1 and higher.

E.2 Sequence Statements

This section describes the sequence statements that are provided with the ConnectWare Plug-In software package. First, the statements and arguments are documented.

DDE_CONNECT

This statement will connect the controller to another Adept controller based on its IP address. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
DDE_CONNECTIP --string_var--HANDLE--variable--STATUS--variable--
```

The statement performs the following steps:

1. The **IP** argument specifies the IP address of the other controller.
2. The **Handle** argument returns back a logical unit number that can be used by the sequence for future communication.
3. The **Status** argument returns back the status of the network connection

DDE_DISCONNECT

This statement will disconnect the communication between the two controllers. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
DDE_DISCONNECTHANDLE--variable--
```

The statement performs the following steps:

1. The **Handle** argument allows the user to specify the Logical Unit Number of the device you wish to disconnect from.

DDE_READ

This statement will read the specified data from the other controller that has been attached via the logical unit number returned by the connection statement. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
DDE_READHANDLE--variable--FROM_VARIABLE--string_var--  
IN_MODULE --string_var--PUT_IN_VAR--string_var--STATUS --variable--
```

The statement performs the following steps:

1. The **Handle** argument allows the user to specify the Logical Unit Number of the device you wish to read from.
2. The **From_Variable** argument specifies the variable name in the other Adept controller to read from.
3. The **In_Module** argument specifies the database module in the other Adept Controller to read from.
4. The **Put_In_Var** argument specifies where the data is written to in the variables database.
5. The **Status** variable returns success if the communication worked.

DDE_WRITE

This statement will write the specified data to the variable database in the other other Adept controller. The statement's syntax is as follows, where the braces ({ . . . }) define optional clauses:

```
DDE_WRITEHANDLE--variable--SEND --string_var--  
TO_VARIABLE--string_var--IN_MOD--string_var--STATUS --variable--
```

The statement performs the following steps:

1. The **Handle** argument allows the user to specify the Logical Unit Number of the device you wish to write to.
2. The **Send** argument specifies the variable that will be used to extract the data for transfer.
3. The **To_Variable** argument specifies the variable in the other Adept controller the data will be written to.
4. The **In_Mod** argument specifies the database module in the other Adept controller where the other variable database resides.
5. The **Status** variable returns success if the communication worked.

TCP-IP V+ Server



F.1 Introduction and Overview	134
TCP Message Format	134
F.2 TCP Debug Menu	135
F.3 TCP Initialization Database	136

F.1 Introduction and Overview

This software module allows a PC or an Adept MV TCP CLIENT to retrieve information from the controller that is running this application. This allow some communication to occur when following the format shown below.

TCP Message Format

The TCp message must be constructed as shown in the format

{x | string}

Where 'x' is the type of message that is to occur based on the list below.

1. x = 's' - evaluate a string expression

Example: '{s | \$ai.ctl[1]}'

2. x = 'r' - evaluate a real variable

Example: '{r | ai.ctl[1]}'

3. x = 'f' - evaluate a function

Example: '{f | switch(power)}'

Example: '{f | bits(1001,8)}'

4. x = 'o' - perform a function

Example: '{f | x=100}'

5. x = 'aim' - read an AIM variable

Example: '{aim | test\var_1}

module<---^ ^----->variable

The return message is the evaluated value in 'ASCII' format without any formatting characters. The server will echo the original message following the results.

Example: Request '{f | switch(power)}'

Result '{f | switch(power)}{-1}'

F.2 TCP Debug Menu

Provided with this software package is a debug panel to allow the user to type in messages and view the response back from the other controller or device.

To view the TCP Debug panel, select the Setup pulldown and the TCP Debug option.

Setup ➔ TCP Debug

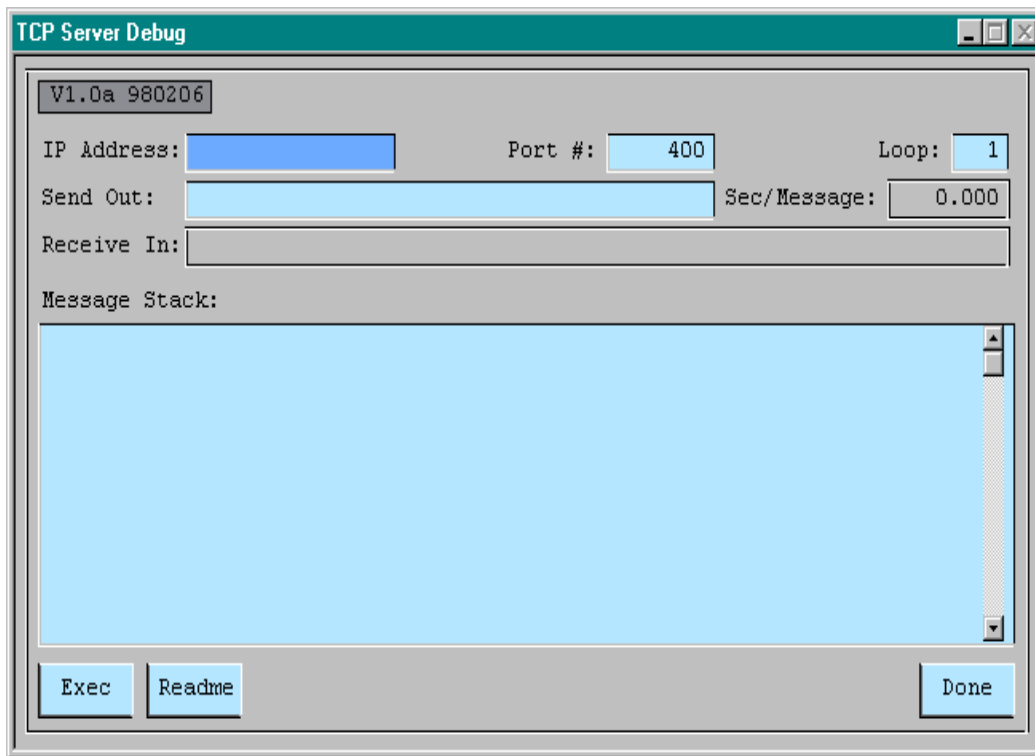


Figure F-1TCP Debug Window

F.3 TCP Initialization Database

Provided with this software package is a initialization database to allow the user to configure to operation of the TCP software package. Listed below are the items available in the database provided with this software.

1. **Statements_TCP**- This record in the database loads a statement database for customized statements for TCP. Currently no statements are available.
2. **TCP Clients Allowed** - This is the maximum number of simultaneous connections to the TCP Server. Maximum is 15 -- Default is 5
3. **TCP Port Number** - This the TCP port number used for the communication between the devices. The current default is 400.
4. **TCP Task** - This the task number the server will run in. Currently defaulted to 18.

Index

A

ARM_CONFIG 56
ATTACH_ROBOT 56

C

CALIBRATE 57
CLEAR_MESSAGE 57

D

Database
 Quickch.db 104
DETACH_ROBOT 57
DISKCOPY utility 104
DISPENSE 56
DISTANCE_FROM 57

G

GET_BITS 58
GET_PARM 58
GET_STATE 59
GET_SWITCH 59
GET_TIME 60

H

HNDSHK_WAIT 56, 57, 58, 59, 60,
 61, 62, 63, 64, 65, 66,
 67, 68, 69, 130, 131

M

MCP_CHECK 61
MCP_WRITE 61

O

OPEN_MENU 62

P

Part Type Database 29

R

RETRACT_Z 64
RETURN_LOC 65

S

SET_BITS 65
SET_OPRMODES 66
SET_PARM 66

SET_SWITCH 67

Statements

 CHANGE_HAND 104
 DISPENSE 130, 131
STATUS_MESSAGE 67

T

TASK_MODE 68
TYPE_MON 69

W

WAIT_FOR_IO 69

Adept User's Manual Comment Form

We have provided this form to allow you to make comments about this manual, to point out any mistakes you may find, or to offer suggestions about information you want to see added to the manual. We review and revise user's manuals on a regular basis, and any comments or feedback you send us will be given serious consideration. Thank you for your input.

NAME _____ DATE _____

COMPANY _____

ADDRESS _____

PHONE _____

MANUAL TITLE: _____

PART NUMBER and REV level: _____

COMMENTS:

MAIL TO: Adept Technology, Inc.
Technical Publications Dept.
11133 Kenwood Rd
Cincinnati OH 45242

FAX: (513) 792-0274



adept
technology, inc.

150 Rose Orchard Way
San Jose, CA 95134
408•432•0888

RDASG-C0002, Rev. B.