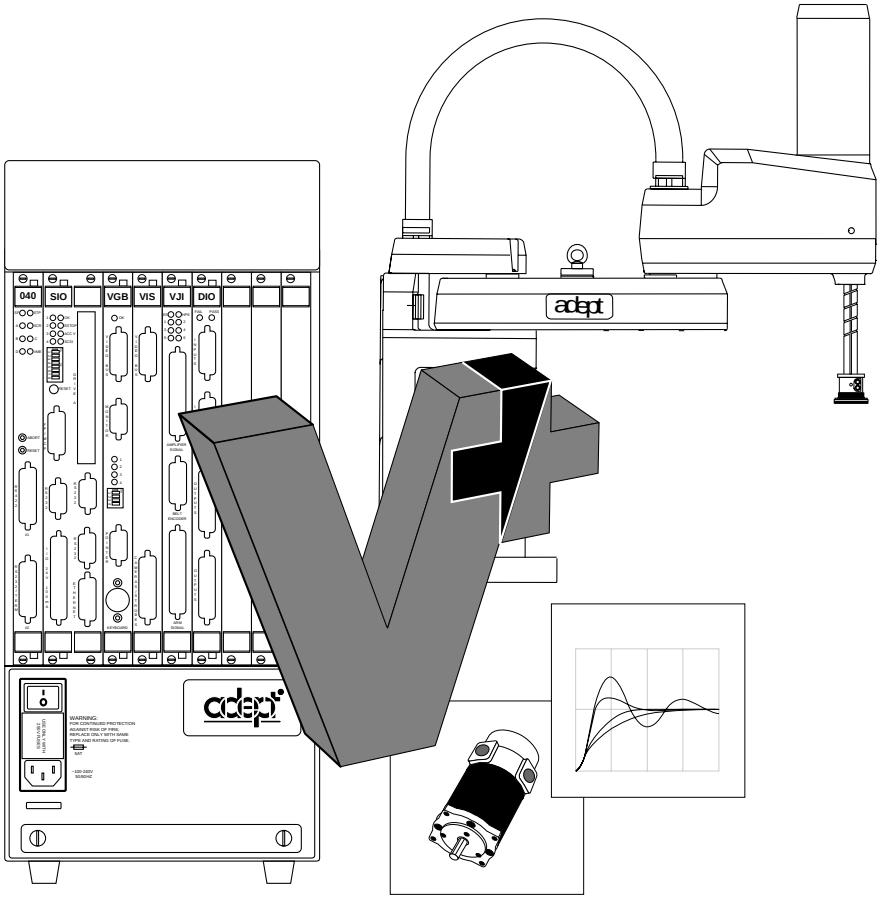
V+ Version 13.x

Release Notes

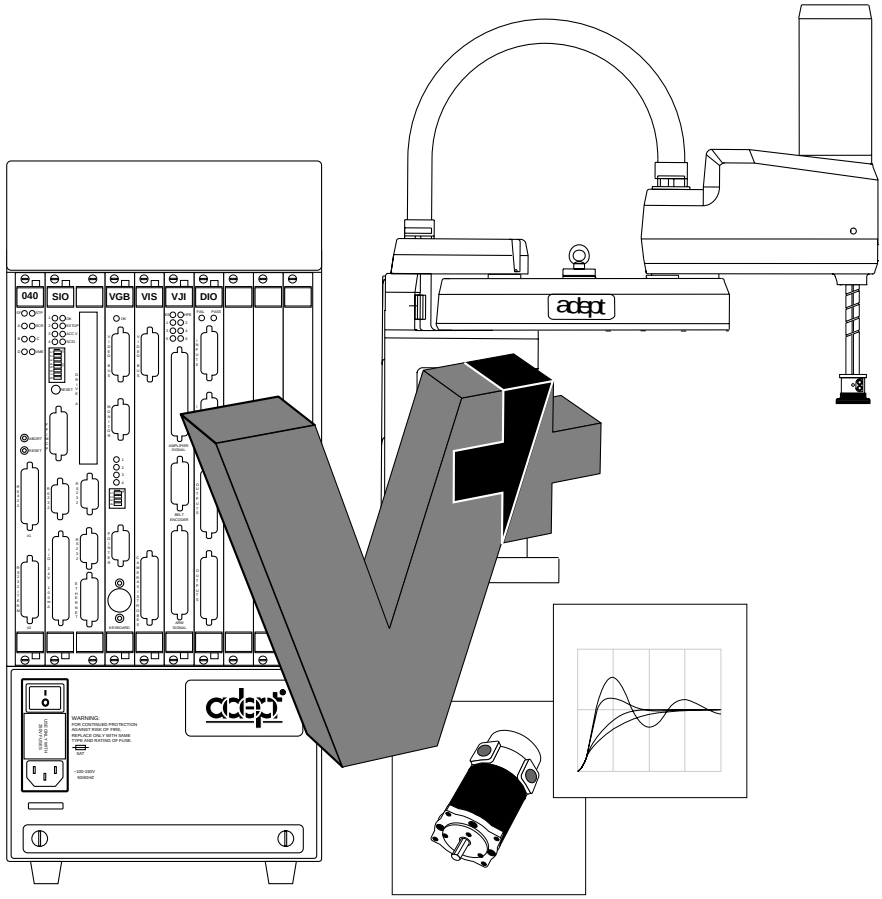
(Describes V+ Version 13.1)



V+ Version 13.x

Release Notes

(Describes V+ Version 13.1)



00963-11120, Rev. A
April 1999

adept technology, inc.

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

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 © 1998 - 1999 by Adept Technology, Inc. All rights reserved.

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

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

AdeptOne-XL, AdeptThree-XL, HyperDrive, Adept 550, Adept 550 CleanRoom, Adept 1850, Adept 1850XP, Adept Cobra 600, Adept Cobra 800, Adept Flexfeeder 250, Adept MV, Adept MV4, AdeptVision, AIM, VisionWare, AdeptMotion, MotionWare, PalletWare, FlexFeedWare, AdeptGEM, AdeptNet, AdeptFTP, AdeptNFS, AdeptTCP/IP, AdeptForce, AdeptModules, AdeptWindows, AdeptWindows PC, AdeptWindows DDE, AdeptWindows Offline Editor, Adept Digital Workcell, 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 11

Related Manuals 12

Standard Manuals 12

Other Adept Product Manuals 13

Optional V+ Developer's Manuals 14

How Can I Get Help? 14

Conventions 15

Notes, Cautions, and Warnings 15

Typographic Conventions 16

Keyboard Conventions 17

Selecting, Choosing, Pressing, and Performing 17

Abbreviations 18

Hypertext Links in Online Manuals 18

Links to Cross-References 18

Links to Related Manuals 18

Links to Related Keywords 19

Installing V+ 19

Summary of Changes 19

Commands and Keywords 19

2 V+ Version 13.1 21

New Functionality 22

Support for Device B Floppy Drive 22

Support for Four-Digit Years 22

Adept Digital Workcell Simulation Mode 23

New System Messages 23

Enhancements to V+ Configuration Data 26

New Vision System Task 27

Description of System Tasks 27

Other Enhancements to V+ 29

Corrections to the Manuals 30
Corrections to the V+ Operating System Reference Guide . . . 30
Corrections to the V+ Language Reference Guide 30
Changes to V+ Keywords 31

3 AdeptVision Version 13.1 105
Summary of AdeptVision Changes 106
ObjectFinder Changes 106
Keyword Changes 106
ObjectFinder Enhancements 107
Overview 107
Concurrency With Other Vision Tools 107
Position and Orientation Offset 107
Position and Orientation Limits 107
Binary and Edge Images 107
Reading ObjectFinder Model Data 108
Details on Concurrent Use 108
Details on Position and Orientation Offsets 110
Details on Position and Orientation Limits 110
Details on Binary Images 111
Details on Edge Images 113
Details on Reading ObjectFinder Model Data 115
Changes to Vision Keywords 116

4 Adept Utility Programs 129
Changes to the Utility Programs 130
CONFIG_C 134
Menu Structure 137
Detailed Descriptions of Program Operations 141
Setting the IP Address of the Adept Controller 155
Software Configurable Switch Settings 156
User Interface 156
AdeptWindows PC Setup 157
V+ Auto-Start 158
Boot Sequence 158

Auto-Boot 158
DDT Serial I/O on Auxiliary CPU 159
DeviceNet Configuration Setup 159
Scanning the DeviceNet for Setup Information 159
DeviceNet Statements in V+ Configuration Data 162
SPEC 168
Menu Structure 171
Descriptions of Selected Program Operations 181
Motion Acceleration and Deceleration Percentages 187
Options for the Frequency Response Tests 187
Adding a Description to a SPEC-Generated Data File 189
Static Friction Feedforward Compensation 190
Navigation Hotkeys 192
Plot Color Preferences 192
Programmable ESTOP Delay-time Parameters 195
Custom Point-to-Point Motion Routine 196

5 Advanced Servo Library 199

Introduction 200
Changes to the Advanced Servo Library 200
New Opcodes 201
Corrected Documentation Errors. 202
New ASL Routines 204

Index 209

List of Figures

Figure 2-1 ID Command Sample Output 66
Figure 4-1 Example DeviceNet Scanning Results 160
Figure 4-2 Typical DeviceNet Configuration for Digital Input and Output. 162
Figure 4-3 DeviceNet Bit Mapping to V+ Signals 166
Figure 4-4 Example of DeviceNet Bit Mapping in V+ 166
Figure 4-5 Initial SPEC Screen. 175
Figure 4-6 Robot Options and Motor Configuration Screen 176
Figure 4-7 Hardware Diagnostics Screen 177
Figure 4-8 Robot Specifications Screen 178
Figure 4-9 SPEC Program Preference Screen 180
Figure 4-10 User-Defined Obstacle Definition Screen 183
Figure 4-11 Sample Collision Detection Options for Box Shape 184
Figure 4-12 The Robot/Motor Calibration Menu. 185
Figure 4-13 Change Homing Configuration Submenu 186
Figure 4-14 Motor Tuning Test Configuration Menu (Frequency Response Test Selected) 188
Figure 4-15 Adding a Description to a SPEC Data File 189
Figure 4-16 Static Friction Feedforward Menu 190
Figure 4-17 Sample Menu with Hotkeys 192
Figure 4-18 Sample Plot Color Preferences 193
Figure 4-19 Sample Plot Color Preferences Menu 194
Figure 4-20 Sample Plot Color Preferences With List of Colors 195

List of Tables

Table 1-1 Standard Manuals 12
Table 1-2 Other Adept Product Manuals 13
Table 1-3 Optional V+ Developer's Manuals 14
Table 1-4 Changes to V+ and AdeptVision Keywords 20
Table 2-1 Description of System Tasks 27
Table 2-2 System Task Priorities 28
Table 2-3 FSET Graphics Window Attributes 47
Table 2-4 FSET Serial-Line Attributes 57
Table 2-5 FSET Attributes for AdeptNet 58
Table 4-1 Changed Utility Disk Files 130
Table 4-2 DeviceNet Scanner Results 161
Table 4-3 Interpretation of DeviceNet Scanning Results 161
Table 4-4 Frequency Response Test Menu Items 188
Table 4-5 Static Friction Feedforward Menu Items 191
Table 5-1 New SPEC Servo Configuration Parameter Names 201

1 Introduction

These release notes describe the changes to the V+ Operating System and Language from version 13.0 that have not yet been described in related manuals. It also covers corrections made to the *V+ Operating System Reference Guide* and the *V+ Operating System User's Guide*.

This chapter provides an introduction to the release notes and covers the following topics:

Related Manuals 12
- Standard Manuals 12
- Other Adept Product Manuals 13
- Optional V+ Developer's Manuals 14

How Can I Get Help? 14

Conventions 15
- Notes, Cautions, and Warnings 15
- Typographic Conventions 16
- Keyboard Conventions 17
- Selecting, Choosing, Pressing, and Performing 17
- Abbreviations 18
- Hypertext Links in Online Manuals 18
  - Links to Cross-References 18
  - Links to Related Manuals 18
  - Links to Related Keywords 19

Installing V+ 19

Summary of Changes 19
- Commands and Keywords 19

Related Manuals

Adept products come with a set of documentation that is appropriate for the products you have ordered. In addition, there are optional manuals available if you are going to be programming the Adept system. These release notes refer to both the standard and optional manuals. The following sections give brief descriptions of the contents and organization of the Adept documentation set.

Standard Manuals

In addition to these release notes, the following manuals are shipped with each Adept MV controller:

Table 1-1. Standard Manuals

Manual	Material Covered
Adept MV Controller User's Guide	Details of the installation, configuration, and maintenance of your Adept controller.
V+ Operating System User's Guide	Loading, storing, and executing programs with the V+ operating system.
Instructions for Adept Utility Programs	The collection of Adept utility programs for configuring and calibrating various features of your Adept system.

Other Adept Product Manuals

When you order an AdeptWindows, AdeptVision VME, AdeptMotion VME, or AdeptForce VME option, any AIM software product, or an Adept robot, you receive manuals that cover those products. Also, optional hardware such as the Manual Control Pendant comes with a manual. A partial list follows.

Table 1-2. Other Adept Product Manuals

Manual	Material Covered
AdeptVision User's Guide	Concepts and strategies for programming the AdeptVision VXL system. (See also the optional *AdeptVision Reference Guide* in **Table 1-3**.)
AdeptMotion VME Developer's Guide	Installation, configuration, and tuning of an AdeptMotion VME system.
AdeptForce VME User's Guide	Installation, operation, and programming of the AdeptForce VME product.
Manual Control Pendant User's Guide	Basic use and programming of the manual control pendant.
AdeptNet User's Guide	Use and programming of the AdeptNet products.
AdeptWindows User's Guide	Installation and use of the AdeptWindows software.

Optional V+ Developer's Manuals

If you are programming V+ applications, you should order the optional V+ developer's manuals (first three in the list below). These manuals contain a complete description of the commands, instructions, functions, and other features available in the V+ language and operating system. These manuals are essential for advanced applications programming.

If you are programming vision applications, you should order the ***AdeptVision Reference Guide*** (in addition to the V+ developer's manuals).

Table 1-3. Optional V+ Developer's Manuals

Manual	Material Covered
V+ Operating System Reference Guide	Descriptions of the V+ operating system commands (known as monitor commands).
V+ Language User's Guide	Programming principles for creating programs in the V+ high-level language.
V+ Language Reference Guide	Complete descriptions of the keywords in the basic V+ language system.
AdeptVision Reference Guide	Descriptions of the additional V+ keywords available with the AdeptVision VXL option.

How Can I Get Help?

Refer to the ***How to Get Help Resource Guide*** (Adept P/N 00961-00700) for details on getting assistance with your Adept software or hardware.

You can obtain this document through Adept On Demand. The phone numbers are:

(800) 474-8889 (toll free)
(503) 207-4023 (toll call)

Please request document number 1020.

Conventions

This section discusses:

- Notes, cautions, and warnings
- Typographic conventions
- Keyboard conventions
- Selecting, choosing, and pressing items
- Abbreviations

Notes, Cautions, and Warnings

Three levels of special notation are used in this manual. In descending order of importance, they are:



WARNING: If the actions indicated in a *warning* are not complied with, injury or major equipment damage could result. A warning typically describes the potential hazard, its possible effect, and the measures that must be taken to reduce the hazard.



CAUTION: If the action specified in a *caution* is not complied with, damage to your equipment or loss of data could result.

NOTE: A *note* provides supplementary information, emphasizes or supplements a point or procedure, or gives a tip for easier operation.

Typographic Conventions

The following typographic conventions are used throughout this manual:

This	Represents
ALL CAPITALS	V+ file names, directory names, commands, keywords, and attributes; also acronyms and keyboard key names.
`monospace`	Screen displays, code examples, non-placeholder terms in formal syntax definitions, and case-sensitive words required for a UNIX-like setting associated with the AdeptNet product.
`italic monospace`	Placeholders for information that you provide in formal syntax definitions.
bold	When you need to type or enter an instruction, you type the boldface line exactly as it appears. For example, if you are asked to type **execute 1 a.diskcopy**, you type all the bold characters exactly as they are printed. What you type is shown in lowercase letters unless it must be typed in uppercase letters to work properly. You may always substitute a currently valid shortcut form when typing a V+ command. In order for the V+ system to process your typing, you must conclude your entry by pressing the ENTER, or RETURN, key. Bold type is used for lowercase names such as subroutine names, variable names, and program names; for example, **a.diskcopy**. Bold type also is used for window items that you choose and window items that do not have initial capital letters in all principal words.
italic	Placeholders that you must provide in typed input.

This	Represents
Initial Capitals	The name of an object such as a window, screen, menu, button, dialog box, or dialog box component. Examples are the Display menu and the Task Profiler window. The *logical* names of physical function keys or buttons use regular font and follow the interface's capitalization, which usually has initial capital letters in all principal words. Examples are the Go To key in the SEE editor and the CMD1 button on the manual control pendant.

Keyboard Conventions

Key combinations appear in the following format:

Notation	Meaning
KEY1+KEY2	A plus sign (+) between keys means that you must press the keys at the same time. For example, "Press CTRL+Z" means that you press CTRL and hold it down while you press Z.

Selecting, Choosing, Pressing, and Performing

In a context using windows, the terms *select*, *choose*, and *press* have different and specific meanings. Selecting an item usually means marking or highlighting it, as in selecting a radio button. Selecting alone does not initiate an action.

Choosing an item carries out an action. For example, choosing a menu item might open a window or carry out a command. You can also initiate an action by choosing a command button (a push button or a standard button). You often need to select an item before you can choose it.

Often you can use a combination of keyboard and mouse techniques for selecting and choosing.

Pressing refers to *physical* keys and buttons. For example, you press the SAVE key and the Program Start button. By contrast, you select or choose a window button.

Performing refers to carrying out a sequence of steps that are defined in a menu-instruction line.

Abbreviations

The following abbreviations may appear in this manual:

Abbreviation	Meaning
CE	European Certification
CIP	Controller Interface Panel
DAC	Digital-to-Analog Converter
HTR	Hold To Run; an MCP switch
KB	Kilobyte (1024 8-bit bytes)
MB	Megabyte (1,048,576 8-bit bytes)
MCP	Manual Control Pendant
MMSP	Manual Mode Safety Package

Hypertext Links in Online Manuals

The online version of this manual contains links to cross-references and to related manuals. This section describes how to identify and use these links.

> **NOTE:** To view the online version of this manual, you must install Acrobat Reader version 3.0 or later on a PC, Macintosh, or UNIX system.

Links to Cross-References

Links to cross-references are displayed using turquoise bold text. When you click on one of these links, the cross-referenced information is displayed. To return to the original information, select the "<<" (go to previous view) icon from the Acrobat Reader toolbar.

Links to Related Manuals

Links to related manuals are displayed using turquoise italic text. When you click on one of these links, the related manual is opened and the title page is displayed along with the bookmark list. You can then select the desired destination within the manual from the bookmark list. To return to the original manual, click the go to previous view icon (<<) from the Acrobat Reader toolbar.

Links to Related Keywords

Links to related keywords are displayed using turquoise bold text. When you click on one of these in the standard reference guides, the cross-referenced information is displayed. However, since this manual contains only new or enhanced keywords, if the related keyword is not in this manual, when you click the hypertext link, the related manual is opened and the title page is displayed along with the bookmark list. You can then select the desired destination within the manual from the bookmark list. To return to the original manual, click the go to previous view icon (<<) from the Acrobat Reader toolbar.

Installing V+

You should use the CONFIG_C utility to install your new V+ system. For installation instructions, refer to the description of the CONFIG_C utility in the manual titled ***Instructions for Adept Utility Programs***.

Summary of Changes

This section provides a list of the software changes made to V+ after version 13.0 was released. Each of these items is described in detail later in the release notes.

- **Support for Device B Floppy Drive**
- **Support for Four-Digit Years**
- **Adept Digital Workcell Simulation Mode**
- **New System Messages**
- **Enhancements to V+ Configuration Data**
- **New Vision System Task**
- **Other Enhancements to V+**
- **ObjectFinder Enhancements**

Commands and Keywords

This section summarizes in table form the changes to:

- V+ monitor commands
- V+ programming keywords

- V+ keywords for the AdeptVision product

In the table, the term *new* indicates a command or keyword that was added to the corresponding version. The term *enhanced* implies compatibility with previous releases so that you can ignore the change if you wish. By contrast, the term *altered* implies incompatibility with previous releases so that you need to investigate the change further. The term *manual* indicates a correction or addition to the appropriate user's or reference guide.

Table 1-4. Changes to V+ and AdeptVision Keywords

Command	Type	V+ 13.1
V+ Monitor Commands		
DIRECTORY	monitor command	manual
EXECUTE	monitor command	manual
HERE	monitor command	enhanced
ID	monitor command	manual
INSTALL	monitor command	manual
LOAD	monitor command	manual
MDIRECTORY	monitor command	manual
NET	monitor command	manual
PING	monitor command	manual
POINT	monitor command	enhanced
PRIME	monitor command	manual
TIME	monitor command	enhanced
XSTEP	monitor command	manual
V+ Programming Keywords		
ADW	system switch	new
AMOVE	program instruction	manual
FSET	program instruction	enhanced, manual
ID	real-valued function	enhanced
IF	program instruction	manual
TIME	program instruction	enhanced
TIME	real-valued function	enhanced
$TIME	string function	enhanced
$TIME4	string function	new
AdeptVision Keywords		
VSHOW.MODEL	program instruction	enhanced
VTRAIN.FINDER	program instruction	enhanced

V+ Version 13.1 2

New Functionality 22
- Support for Device B Floppy Drive 22
- Support for Four-Digit Years 22
- Adept Digital Workcell Simulation Mode 23
- New System Messages 23
- Enhancements to V+ Configuration Data 26
- New Vision System Task 27
  - Description of System Tasks 27
- Other Enhancements to V+ 29

Corrections to the Manuals 30
- Corrections to the V+ Operating System Reference Guide . . 30
- Corrections to the V+ Language Reference Guide 30

Changes to V+ Keywords 31

New Functionality

V+ 13.1 and later includes the new features and enhancements described in the following sections.

Support for Device B Floppy Drive

V+ version 13.1 supports the floppy-drive option for the AWC processor. The optional floppy, which is referenced as drive “B”, is accessed in the same manner as the "D" drive (Compact Flash), and the “A” floppy drive and “C” hard drive in the SIO module.

To enable application programs to determine whether or not the B floppy drive is present in the controller, the ID real-valued function has been enhanced. The following bit is now used in the value returned by ID(11,1).

Bit 8 Device B floppy board present (mask = 128)

This bit is set if a device B floppy interface board is present in the controller.

NOTE: This bit indicates only that the interface board is present. It does not indicate that the floppy drive is installed.

See the description of the ID real-valued function in the ***V+ Language Reference Guide*** for more details on ID values that identify the current system configuration.

Support for Four-Digit Years

This section describes new V+ functionality that was added in V+ version 13.1 to support four-digit years.

- A date and time string function that outputs four-digit years was added. The syntax is:

  ```
  $TIME4(date, time)
  ```

  This function is similar to $TIME (date, time), except that $TIME4 outputs the date with a four-digit year (dd-mmm-yyyy hh:mm:ss).

- The V+ startup message was changed to always display a four-digit year.

- The TIME monitor command was changed to accept both two- and four-digit year values, and to always display a four-digit year. The syntax is:

  ```
  TIME dd-mmm-yy hh:mm:ss
  ```

  or

  ```
  TIME dd-mmm-yyyy hh:mm:ss
  ```

- The TIME( ) real-valued function was changed to accept both two- and four-digit year values, and to return correct values for years up to 9999.
- The $TIME( ) string function was changed to return an error year (**) if a date value later than year 2079 is passed as input.
- The TIME monitor command and the TIME program instruction were changed to reject years later than 2079, since that is the limit of the hardware clock.

Adept Digital Workcell Simulation Mode

This is a new feature of V+ that permits a workcell (including the robot, conveyor belt, and other equipment connected to an Adept controller) to be simulated by a PC running the Adept Digital Workcell software. That software simulates the robot workcell by displaying its operation on the computer screen, and by responding to the actual V+ commands to the robot and other equipment.

When operating in this mode, the V+ system does not control the physical robot(s) and other equipment connected to the Adept controller. Instead, all robot motion commands, as well as control signals to other workcell equipment, are sent to the Adept Digital Workcell software running on a PC, which must be on the same Ethernet network as the Adept controller.

Use of Adept Digital Workcell simulation mode by V+ is controlled with the ADW system switch. Use of that switch, however, is different from the use of all other V+ system switches. Unlike other system switches, there is no functional effect when the switch is enabled or disabled. Instead, a message is displayed telling the user that the Adept controller must be turned off and back on for the change to take effect.

For complete information about the Adept Digital Workcell system and its use, see the ***Adept Digital Workcell User's Guide***.

New System Messages

The following system messages were added in V+ 13.1:

Adept Digital Workcell Simulation mode

Explanation: This message is displayed during V+ initialization (i.e., during the booting process), and as part of the output displayed by the ID monitor command, to indicate that the V+ system is operating in Adept Digital Workcell simulation mode.

User Action: No response is required if the V+ system is intended to be used in Adept Digital Workcell simulation mode. Otherwise, the command DISABLE ADW should be entered and the Adept controller should be rebooted.

Connecting to Adept Digital Workcell system via Ethernet (^C to quit)

Explanation: This message is displayed during V+ initialization (i.e., during the booting process) to indicate that the V+ system is operating in Adept Digital Workcell simulation mode, and that the V+ system is waiting for a response from the PC that is running the Adept Digital Workcell software.

User Action: No response is needed if the connection completes successfully. If the connection does not complete, the user can enter CTRL+C to cancel waiting for the connection. Then, the V+ system reports failures of all the robot servos and all the installed licenses are disabled (except those needed to use the AdeptWindows PC user interface). In that case, the user should either try again to start up the Adept controller (after making sure that the Adept Digital Workcell software is running on the PC), or enter the V+ monitor command DISABLE ADW and restart the Adept controller.

***Inconsistent model mode setting* (–763)**

Explanation: Inconsistent modes (binary with backlight, binary without backlight, or grayscale) exist between different vision models if the error occurs during planning, or between the planned model(s) and the object finder if the error occurs during recognition.

User Action: During planning, combine only models trained with the same mode (either all grayscale or all binary with the same backlight setting). Retrain some models if the wrong mode setting has been used by mistake. During recognition, set the object finder mode setting to be similar to that of the planned models. Modes are set using the VTRAIN.FINDER instruction and the V.BACKLIGHT[ ] system switch.

***Not supported in simulated vision mode* (–764)**

Explanation: An attempt has been made to execute a vision command, instruction, or function while the V+ system is operating in Adept Digital Workcell vision simulation mode. Vision operations are not supported in that mode, which is used when the V+ system is operating in Adept Digital Workcell simulation mode and no vision hardware is installed in the Adept controller.

User Action: Do not attempt to use vision operations when the V+ system is operating in Adept Digital Workcell simulation mode and no vision hardware is installed. If you must use vision operations, install the proper vision hardware.

***Interrupted multi-segment motion* (–902)**

Explanation: Currently, the only multiple-segment motions are MOVEF and MOVESF. For these motion instructions, this error is signaled if the DEPART motion has been initiated, but neither the APPROACH nor the final move to the destination is performed. For example, this situation might occur if an envelope error is detected during the DEPART or the APPROACH motion segments.

User Action: Correct the problem that caused the robot motion to terminate prematurely and reexecute or skip the multiple-segment motion.

Now reboot the controller to disable Adept Digital Workcell simulation mode.

Explanation: The V+ monitor command DISABLE ADW has been entered. This message is telling the user that the Adept controller must be rebooted to actually discontinue Adept Digital Workcell simulation mode.

User Action: If Adept Digital Workcell simulation mode is to be terminated, turn off power to the Adept controller, and turn power back on to reboot the V+ system. Otherwise, enter the V+ monitor command ENABLE ADW to cancel the effect of the previous DISABLE command.

Now reboot the controller to enable Adept Digital Workcell simulation mode.

Explanation: The V+ monitor command ENABLE ADW has been entered. This message is telling the user that the Adept controller must be rebooted to initiate Adept Digital Workcell simulation mode.

User Action: If Adept Digital Workcell simulation mode is to be used, turn off power to the Adept controller, start up the Adept Digital Workcell software on the PC, and turn on power to the Adept controller.

Otherwise, enter the V+ monitor command DISABLE ADW to cancel the effect of the previous ENABLE command.

Okay to restart the servos (Y/N)? (12)

Explanation: The user has entered the command ENABLE ADW while in Adept Digital Workcell simulation mode. In response to this command, V+ will restart the servos to account for any robot configuration changes that have made in the Adept Digital Workcell system. V+ is asking for confirmation before restarting the servos.

User Action: If ENABLE ADW was entered on purpose to restart the servos, enter "Y". Otherwise enter "N". See the ***Adept Digital Workcell User's Guide*** for information about restarting the servos.

Push, release, hold, MCP enable switch (58)

Explanation: The user is attempting to enable HIGH POWER while in manual mode. This operation requires that the MCP enable switch be tested. HIGH POWER cannot be enabled unless the user toggles the switch to test it.

User Action: Push, then release, then hold the enable switch within a limited time. Alternately, you can change from manual mode to automatic mode.

Enhancements to V+ Configuration Data

The following new system configuration statements were added to the .SYSTEM section for V+ 13.1 and later. See **page 153** for descriptions of these new statements.

- PASSIVE_ESTOP = "IGNORE_ENABLE" or "TRACK_ENABLE"
- STARTUP_MESSAGE = "Any custom message"

Also in the .SYSTEM section, the statement

NETWORK = "/BUFFERS n"

has no effect with V+ 13.1 and later. That is, V+ ignores the statement if it is present. (CONFIG_C version 13.1A and later completely suppresses that statement if it is encountered in configuration data for V+ 13.1 or later.)

New Vision System Task

A new vision system task (Vision Analysis #2) was added in V+ 13.1. The vision task priorities were also changed. See **Table 2-1** and **Table 2-2** for details. (The previous versions of these tables are located in the ***V+ Language User's Guide***.)

Description of System Tasks

The system tasks and their functions are shown in **Table 2-1**.

Table 2-1. Description of System Tasks

Task	Function
Trajectory Generator	Compute the series of set points that make up a robot motion
Terminal/Graphics	Refresh the terminal or graphics monitor display
Monitor	Service user requests entered at the monitor window (monitor commands and responses to system prompts)
DDCMP	Handle implementation of DDCMP protocols for serial lines configured as DDCMP lines
Kermit	Handle implementation of Kermit protocols for serial lines configured as Kermit lines
Pendant	Handle manual control pendant I/O
Disk Driver	Handle requests for I/O to the hard and floppy disk drives and the Compact Flash
Serial I/O	Service serial I/O ports
Pipes Driver	Allow a V+ task to service I/O requests like a standard I/O driver
NFS Driver	Allow access of remote files on network file servers using the Network File Services protocol
TCP Driver	Handle the TCP network communications protocol on Ethernet
Vision Communications	Communicate between V+ and vision software
Vision Analysis	Evaluate vision commands

Table 2-1. Description of System Tasks (Continued)

Vision Analysis #2	Vision tool analysis
Servo Communications	Communicate with the servo interrupt routines or the motion-control hardware
Cat 3 Timer	Handle timing and sequencing when robot power is enabled in systems with the Cat 3 option enabled

Table 2-2. System Task Priorities

System Task	**Time Slice**															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Trajectory Generator	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63
Terminal/ Graphics	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	58
Monitor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56
DDCMP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42
Kermit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52
Disk Driver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39	48
Serial I/O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	44
Pipes Driver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43	0
NFS Driver	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	40
Vision Comm	14	14	14	14	14	14	14	14	14	14	14	14	14	0	0	0
Vision Analysis	12	12	12	12	12	12	12	12	12	12	12	12	12	0	0	0
Vision Analysis #2	13	13	13	13	13	13	13	13	13	13	13	13	13	0	0	0

Table 2-2. System Task Priorities (Continued)

System Task	Time Slice															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Servo Comm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	0
Cat 3 Timer	0	45	0	45	0	45	0	45	0	45	0	45	0	45	0	0

Other Enhancements to V+

The following miscellaneous enhancements were made in V+ version 13.1:

- Added support for the AWC Gen-II board.
- Added support for adjusting the delay between assertion of ESTOP and the actual removal of robot power. See **page 195** for details. (Requires the AWC Gen-II board.)
- Added support for high-speed encoder latching with all Adept motion-interface boards. (Requires AWC Gen-II board.)
- Added support for a new Ethernet buffer scheme, which provides faster update of the Vision window when using the AdeptWindows PC interface. (Requires DDT version 4.3 or later on the AWC Gen-II board.)
- Enabled the FSET program instruction's /SPECIAL NOSELECTABLE attribute for the AdeptWindows PC interface. This prevents a window from being selected from the pull-down menu under the Adept icon (the window name is dimmed in the pull-down menu). See the description of the FSET program instruction on **page 45** for details.
- If the controller NVRAM data becomes invalid, the AdeptWindows, AdeptTCP/IP, AdeptNFS, and AdeptFTP licenses are automatically enabled so that the system can be restored. (Requires an AWC board and V+ version 13.1 or later.)
- Enhanced the General SCARA (4x4 Coupled) robot module (SCC) to include motor limit checking if the standard robot option bit is enabled. This modification does not affect the compatibility of this module with the previous version released with V+ version 13.0.

Corrections to the Manuals

The following sections contain correction information for the V+ version 13.0 manual set.

Corrections to the V+ Operating System Reference Guide

Corrections have been made to the following V+ monitor commands. The new descriptions of these keywords are included in these release notes.

- **DIRECTORY**
- **EXECUTE**
- **ID**
- **INSTALL**
- **LOAD**
- **MDIRECTORY**
- **NET**
- **PING**
- **PRIME**
- **XSTEP**

Corrections to the V+ Language Reference Guide

The following corrections are documented in these release notes:

- AMOVE Program Instruction—the description of this instruction was not included in the reference guide. See **page 36**.
- FSET Program Instruction—in the /SPEED attributes section, for the serial ports on the Adept SIO board, a baud rate of 19200 is incompatible with a baud rate of 7200 or 38400 on a pair-wise basis. See **page 45** for details.
- IF... GOTO Program Instruction—the correct syntax is:

  ```
  IF logical_expr GOTO label
  ```

  See **page 73** for details.

Changes to V+ Keywords

This section presents detailed descriptions of any changed V+ system keywords. See **"Changes to Vision Keywords" on page 116** for keywords related to the AdeptVision option.

The V+ system includes the following types of keywords:

- Monitor commands
- Program instructions
- Functions
- System parameters
- System switches

The keywords are presented in alphabetical order, with the description for each keyword starting on a new page. Note that some keywords may belong to more than one keyword type. The keyword is shown at the top of each page. The sections described on pages 32 and 33 are included for each keyword, as appropriate.

Syntax

This section presents the syntax of the keyword. The keyword is shown in uppercase and the arguments are shown in lowercase. The keyword must be entered exactly as shown. Parentheses must be placed exactly as shown. Required keywords, parameters, and marks such as equal signs and parentheses are shown in bold type; optional keywords, parameters, and marks are shown in regular type. In the example:

KEYWORD req.param1 = req.param2 OPT.KEYWORD opt.param

KEYWORD	must be entered exactly as shown. In the SEE editor, you can abbreviate keywords to a length that uniquely identifies the keyword. The SEE editor automatically expands the keyword to its full length.
req.param1	must be replaced with a value, variable, or expression
=	the equal sign must be entered
req.param2	must be replaced with a value, variable, or expression
OPT.KEYWORD	can be omitted but must be entered exactly as shown if used
opt.param	may be replaced with a value, variable, or expression but assumes a default value if not used

The keyword type (program instruction, function, and so on) is shown at the top of the page.

An abbreviated syntax is shown for some keywords. This is done when the abbreviated form is the most commonly used variation of the complete syntax.

Function

This section gives a brief description of the keyword.

Usage Considerations

This section lists any restrictions on the keyword's use. If specific hardware or other options are required, they are listed here.

Parameters

The requirements for input and output parameters are explained in this section. If a parameter is optional, that is noted here. When an instruction line is entered, optional parameters do not have to be specified and the system will assume a default. Unspecified parameters at the end of an argument list can be ignored. Unspecified parameters in the middle of an argument list must be accounted for by commas. For example, the following keyword has four parameters—the first and third are used, and the second and fourth are left unspecified:

```
SAMPLE.INST var_1,,"test"
```

String and numeric input parameters can be constant values (3.32, "part_1", etc.), variable names (see Chapter 1 of the ***V+ Language User's Guide*** for the requirements of a variable name), or expressions that combine constants and variables. The data type of the constant, variable, or expression must agree with the type expected by the keyword. String variables must be preceded by a "$". Precision-point variables must be preceded by a "#". Belt variables must be preceded by a "%". String constants must be enclosed in quotes. Real and integer constants can be used without modification. (V+ keywords cannot be used as variable names—see Appendix A of the ***V+ Language Reference Guide*** for a quick-reference list of keywords.)

Details

This section describes the function of the keyword in detail.

Examples

Examples of correctly formed command or instruction lines are presented in this section.

Related Keywords

Additional keywords that are similar or are frequently used in conjunction with this instruction are listed here.

Syntax

```
... ADW
```

Function

Determines whether the V+ robot-control system controls the physical robot(s) (ADW disabled) or interacts with the Adept Digital Workcell simulation product running on a remote computer (ADW enabled).

Usage Considerations

When the setting of the ADW switch is changed, the change does not take effect until after the V+ system is rebooted.

The ADW switch should be disabled (and the V+ system rebooted if necessary) for normal operation of the robot(s) connected to the Adept controller.

The SWITCH command displays (and the SWITCH function returns) the setting of this switch when the V+ system was initially booted from disk. That is, the indicated setting does not change after the switch is subsequently enabled or disabled during a V+ session. Thus, the reported setting indicates the mode that the V+ system is operating in.

Details

When the V+ system is booted after this switch had been *disabled*, the V+ system controls the robot(s) and other equipment connected to the Adept controller in the normal manner.

When the V+ system is booted after this switch had been *enabled*, the V+ system does not control the robot(s) and other equipment connected to the Adept controller. Instead, all robot motion commands, as well as control signals to other workcell equipment, are sent to the Adept Digital Workcell simulation software running on a PC, which must be on the same Ethernet network as the Adept controller.

For complete information about the Adept Digital Workcell system and its use, see the ***Adept Digital Workcell User's Guide.***

If Adept Digital Workcell simulation mode is already active when the monitor command ENABLE ADW is processed, the process to restart the robot servos is initiated. The servos must be restarted if the Adept Digital Workcell robot configuration is changed on the remote computer. Normally, the servos are automatically restarted by the remote computer when required. However, the restart cannot be done while there is any active V+ task. Thus, a restart request from the remote computer might fail, and a user-invoked restart might be needed.

This switch is initially set to the state stored in the controller NVRAM. That state was set when the ADW switch was enabled or disabled during a previous V+ session.

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

Syntax

```
AMOVE  axis = position
```

Function

Position an **extra** robot axis during the next joint-interpolated or straight-line motion to a transformation location.

Usage Considerations

No robot motion occurs when the AMOVE instruction is executed: The requested motion occurs when the **next** robot motion is performed.

If the next motion is a DRIVE, DELAY, or motion to a precision point, the AMOVE instruction is ignored.

The AMOVE instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the AMOVE instruction will cause an error.

The AMOVE instruction uses the basic program speed rather than the rotational speed to compute motion times during Cartesian motions.

Parameter

axis	Real value, variable, or expression (interpreted as an integer) that specifies the number of the **extra** robot axis to be moved. This value can be 1, 2, ..., **n,** where **n** is the total number of **extra** axes that the robot contains.
position	Real value, variable, or expression that specifies the (absolute) position desired for the extra axis. This value is interpreted in the units used to measure the axis position. That is, the position for a rotary axis must be specified in degrees, and the position for a linear axis must be in millimeters.

Details

This instruction positions an extra robot axis during the next joint-interpolated or straight-line motion. However, this instruction has no effect if the next motion is caused by a DRIVE or DELAY instruction or is a motion to a location specified by a precision point. That is, this instruction is **canceled** by those cases.

An **extra** axis is a redundant or independent axis that is an integral part of a robot device. For example, if a robot has two colinear Z axes, one of the two axes is designated as the redundant axis. Or if the robot is mounted on a slide axis (for example, to allow the robot to be positioned at different work stations), the slide axis can be included as an independent axis in the geometric model for the robot.

An **extra** axis can be positioned directly by executing a DRIVE instruction, but then the **extra** axis moves by itself. That is, the other robot axes do not move while the **extra** axis is moving.

There are two ways to make the **extra** axis move along with all the other robot axes:

- Perform a motion to a location specified by a precision point.
- Execute an AMOVE instruction to specify the desired destination position for the **extra** axis, and then execute a motion to a destination specified as a transformation value.

When the **extra** axis is moved with the other robot axes, all the axes are moved so they reach their destination positions at the same time.

Examples

`AMOVE 2 = 100.0`	Position the second extra axis of the robot to (absolute) position 100 during the next motion of the robot—unless a DRIVE or DELAY instruction is encountered first, or the location specified in the next motion instruction is a precision point.

Related Keywords

APPRO and **APPROS**	(program instructions)
DELAY	(program instruction)
DEPART and **DEPARTS**	(program instructions)
DRIVE	(program instruction)
MOVE and **MOVES**	(program instructions)
MOVET and **MOVEST**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

```
DIRECTORY switch wildcard_spec
```

Function

Display the names of some or all of the programs in the system memory.

Usage Considerations

This command lists the programs resident in system memory. It does not list the files on a disk drive. The FDIRECTORY command lists the files on a disk drive.

Parameters

`switch` Optional qualifier whose possible values are:

Switch Value	Purpose
/S	Suppress protected programs
/?	Display only nonexecutable programs
/M	Display only modified programs

`wildcard_spec` Optional character string that can include wildcards using either the "?" or "*" character. Both wildcards operate identically and can match 0, 1, or multiple characters.

Details

The following information can be displayed for each program listed:

- A question mark ("?") is shown at the left if the program cannot be executed. This usually indicates that the program contains a programming error.
- For any program name that is displayed, if the copy in memory has been modified since last being loaded or stored, an `M` is displayed before the program name.
- If the program has restricted access, a code letter for the type of restriction is shown at the left.
  - A `P` is shown if the program is *protected*. That means the program cannot be displayed, edited, or stored.
  - An `R` is shown if the program is *read-only*. That means the program cannot be edited or stored.

- The name of the program is displayed.
- If the program is not protected (see above), the entire .PROGRAM statement for the program is displayed. Thus, any parameters required by the program are displayed, as is any comment on the .PROGRAM line.
- If the program is protected (see above), the .PROGRAM statement is truncated after the program name.

Related Keywords

FDIRECTORY (monitor command)
MDIRECTORY (monitor command)

Syntax

```
EXECUTE /C task_num program(param_list), cycles, step,
          priority[i]
```

Function

Begin execution of a control program.

Usage Considerations

No program can already be active as the specified program task.

The `priority` array is normally used only during experimental tuning of the tasks (see below).

Parameters

`/C` Optional qualifier that conditionally attaches the selected robot. The qualifier has an effect only when starting the execution of task 0.

`task_num` Optional integer specifying which program task is to be activated. (See the ***V+ Operating System User's Guide*** for information on tasks.)

`program` Optional name of the program to be executed. If the name is omitted, the last program name specified in an EXECUTE command or instruction (or PRIME command) for the selected task is used.



WARNING: Entering an EXECUTE command with no program specified could result in unexpected motion of the robot, since the previous program is executed again.

`param_list` Optional list of constants, variables, or expressions separated by commas, which must correspond in type and number to the arguments in the .PROGRAM statement for the program specified. If no arguments are required by the program, the list is blank, and the parentheses may be omitted.

Program parameters may be omitted as desired, using commas to skip omitted parameters. No commas are required if parameters are omitted at the end of the list. Omitted parameters are passed to the called program as "undefined" and can be detected with the DEFINED real-valued function.

Automatic variables (and subroutine arguments) cannot be passed by reference in an EXECUTE instruction. They must be passed by value (see the description of the CALL instruction).

The parameters are evaluated in the context of the new task that is started (see below).

`cycles` Optional real value, variable, or expression (interpreted as an integer) that specifies the number of program execution cycles to be performed. If omitted, the cycle count is assumed to be 1. For unlimited cycles, specify any negative value. The maximum loop count value allowed is 32767.

`step` Optional real value, variable, or expression (interpreted as an integer) that specifies the step at which program execution is to begin. If omitted, program execution begins at the first executable statement in the program (that is, after the initial blank and comment lines, and all the AUTO, GLOBAL, and LOCAL instructions).

`priority[]` Optional array of real values (interpreted as integers) that are used by V+ to override the default execution priority of the task. The array contains 16 elements, which specify the program priority in each of 16 one-millisecond time slices. If specified, the elements must be in the range –1 to 64, inclusive. (See the ***V+ Language User's Guide*** for the details of task scheduling.)

`i` Optional real value, variable, or expression (interpreted as an integer) that specifies the index value of the first element to be used. If omitted, the value 0 is assumed.

Details

This command initiates execution of the specified control program. The program will be executed `cycles` times, starting at the specified program step. If no program is specified, the system reexecutes the last program executed by the selected program task.

Note that there is an EXECUTE program instruction, as well as the monitor command. Thus, one program can initiate execution of other, related programs. (After a program initiates execution of another program, the initiating program can use the STATUS and ERROR real-valued functions to monitor the status of the other program.)

If the task number is not specified, the EXECUTE command accesses task number 0 if the system is not in DEBUG mode; the current debug task is assumed if the system is in DEBUG mode. The EXECUTE instruction always accesses task #0 if the task number is omitted.

The optional `/C` qualifier has an effect only when starting execution of task 0. When `/C` is not specified, an EXECUTE command for task 0 fails if the robot cannot be attached; attachment requires that the robot is calibrated and that arm power is enabled (or that the DRY.RUN switch is enabled). When `/C` is specified, an EXECUTE command for task 0 attempts to attach the robot but allows execution to continue without any indication of error if the robot cannot be attached.

Certain default conditions are assumed whenever program execution is initiated. In effect, these are equivalent to the following program instructions:

```
DURATION 0 ALWAYS

FINE 100 ALWAYS

LOCK 0

MULTIPLE ALWAYS

NULL ALWAYS

SPEED 100,100 ALWAYS

SELECT ROBOT = 1
```

Also, the robot configuration is saved for subsequent motions.

An execution cycle is terminated when a STOP instruction is executed, a RETURN instruction is executed in the top-level program, or the last defined step of the program is encountered. The value of `cycles` can range from –32768 to 32767. The program is executed one time if `cycles` is omitted or has the value 0 or 1. Any negative value for `cycles` causes the program to be executed continuously until a HALT instruction is executed, an error occurs, or the user (or another program) aborts execution of the program.

> **NOTE:** Each time an execution cycle is initiated, the execution parameters are reset to their default values. This includes motion speed, robot configuration, and servo modes. However, the robot currently selected is not changed.

If `step` is specified, the program begins execution at that step for the first pass. Successive cycles **always** begin at the first executable step of the program.

The `priority` values override the default execution configuration for the specified program task. The specified priorities remain in effect only until the next time an EXECUTE instruction is issued for the program task. The acceptable values are:

–1	do not run in this slice even if no other task is ready to run
0	run in this slice only when no other task is ready to run
1-64	run in this slice according to the specified priority (higher priority tasks may lock out lower priority tasks for the duration of the time slice—64 is the highest priority)

When setting priorities, remember:

1-31	are normal user task priorities
32-62	are used by V+ device drivers and system tasks
63	is used by the trajectory generator (do not use 63 or 64 unless you have very short task execution times, or jerky robot motions may result)

Example

Initiate execution (as task #0) of the program named "assembly", with execution to continue indefinitely (that is, until execution is aborted, a HALT instruction is executed, or a run-time error occurs).

```
EXECUTE assembly,-1
```

Initiate execution, with program task #2, of the program named "test". The parameter values 1 and 2 are passed to the program.

```
EXECUTE 2 test(1,2)
```

Initiate execution of the last program executed by program task #0 (or by the current debug task). No parameters are passed to the program.

```
EXECUTE
```

Related Keywords

ABORT	(monitor command and program instruction)
CALL	(program instruction)
CYCLE.END	(monitor command and program instruction)
KILL	(monitor command and program instruction)
PRIME	(monitor command)

PROCEED	(monitor command)
RETRY	(monitor command)
SSTEP	(monitor command)
STATUS	(monitor command and real-valued function)
XSTEP	(monitor command)

Syntax

```
FSET (logical_unit) attribute_list
```

Function

Set or modify attributes of a graphics window, serial line, or network device related to AdeptNet.

Usage Considerations

If a window has been referenced, it must have been opened already with an FOPEN instruction. If a serial line is referenced, it must have been attached already with an ATTACH instruction.

The use of this instruction with NFS or TCP network devices applies only to systems fitted with the AdeptNet option and with the appropriate license(s).

As with all V+ I/O instructions, the IOSTAT real-valued function should be used after each FSET instruction to determine the success of the FSET request.

Parameters

`logical_unit`	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
`attribute_list`	List of string constants, variables, and expressions; real values, variables, and expressions; and format specifiers used to define the characteristics of the window. See the description of the FOPEN instruction for detailed information on this parameter.

Details

Using FSET With Windows

This instruction sets attributes for a window that already exists and is already open for access.

> **NOTE:** No interlocks exist to prevent multiple program tasks from changing the attributes of a window.

The argument list for this instruction has exactly the same format as that for the FOPEN instruction, except that no window name is included for this instruction. Refer to the description of the FOPEN instruction for information on how window attributes are specified.

The window attributes described below can be specified with either the FSET or FOPEN instructions (unless noted otherwise). For some attributes, after the attribute has been specified, its effect may be modified later in one of the following ways:

- The following keywords set attributes that can be changed only by another FSET instruction.

/BORDER	/LUT	/SPECIAL
/EVENT	/MARGINS	/TERMINAL
/FONT	/MENU	/UPDATE
/FONT_HDR	/CURSOR	/V_ARROWINC
/H_ARROWINC	/NOUPDATE	/V_RANGE
/H_RANGE	/POINTER	

- The following keywords set attributes that can be changed by another FSET instruction or by pointer device activity.

/ABS_POSITION	/NODISPLAY	/SIZE
/DISPLAY	/POSITION	/STACK
/H_HANDLE	/SELECT	/V_HANDLE
/H_SCROLL	/SHOW	/V_SCROLL

- The following keywords are related to the pull-down menu and do not actually remain as attributes of the window.

/IGNORE	/PULLDOWN

This information is provided for each attribute described in **Table 2-3**:

- Keyword that identifies the attribute

  Some attributes can be either on or off. For each such attribute there are two corresponding keywords. The on keyword directly relates to the attribute (for example, /DISPLAY); the off keyword has a NO prefix (for example, /NODISPLAY).

- Description of the effect of the attribute

  For attributes with an on/off character, the on condition is described.

- Description of any arguments required for the attribute
- Default attribute setting

Table 2-3. FSET Graphics Window Attributes

Attribute:	**/ABS_POSITION**
Explanation:	Position the window at an absolute location on the screen rather than at a location relative to the parent window.
Arguments:	The X and Y offsets from the upper left corner of the monitor.
Default:	None
Attribute:	**/BORDER**
Explanation:	Enable a border around all or part of the window. The border is a line (one pixel wide) inside the window boundary, and it will obscure anything in that area. The border is always drawn with color 1 (black).
Arguments:	One or more of the following keywords. • ALL = Display all borders • NONE= Display no borders • TOP = Display top border • NOTOP = Suppress top border • BOTTOM = Display bottom border • NOBOTTOM = Suppress bottom border • LEFT = Display left border • NOLEFT = Suppress left border • RIGHT = Display right border • NORIGHT = Suppress right border
Default:	/BORDER ALL
Attribute:	**/CURSOR**
Explanation:	Set the mouse cursor to a loaded icon.

Table 2-3. FSET Graphics Window Attributes (Continued)

Arguments:	The index of the icon element to use, followed by the name of the icon (in single quotes)
Default:	None
Attribute:	**/DISPLAY and /NODISPLAY**
Explanation:	Display the window. Top-level windows that are not displayed may be displayed by selecting their names from the Adept pull-down menu at the top of the screen.
Arguments:	None
Default:	/DISPLAY
Attribute:	**/EVENT**
Explanation:	Enables or disables event processing for the window. If you want to receive events, you must explicitly use this attribute with FSET or FOPEN each time you open a window. The details of event processing, and descriptions of the events that can be specified with this keyword, are described with the GETEVENT instruction. If the window is opened for write-only access (see the /WRITEONLY attribute), this keyword causes the error File already opened.
Arguments:	One or more keywords (see GETEVENT)
Default:	/EVENT NONE
Attribute:	**/FONT**
Explanation:	Selects a character font to be used for all text output in the window, except for output from the GTYPE instruction. The new font will take effect immediately, with the following effect on the placement of the next character output to the window: (1) The new character cell will have its left edge at the same pixel column as before the font change. (2) Its baseline will remain at the same pixel row. (3) If the top of the character cell extends above the text top margin, the character position will be bumped down so that no part of the cell is outside the text margins. (4) A similar rule applies if the right edge of the character cell extends beyond the text right margin. Font number 1 is the standard font—it has character cells 8 pixels wide and 15 pixels high; there are 3 pixels below the baseline; and uppercase letters are 9 pixels high.
Arguments:	Font number (the value must be greater than or equal to one)
Default:	/FONT 1

Table 2-3. FSET Graphics Window Attributes (Continued)

Attribute:	**/FONT_HDR**
Explanation:	This attribute is ignored by the AdeptWindows PC interface. Specify the font number to use for the title, menu bars, and pull-down menus of the window. (Changes the title bar immediately, changes the menu bars at the next FSET /MENU... instruction.)
Arguments:	Font number
Default:	/FONT 1
Attribute:	**/H_ARROWINC**
Explanation:	Sets the change in the position of the scroll handle in the horizontal scroll bar that will be caused by clicking on the arrow buttons in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see the GETEVENT instruction].)
Arguments:	Number of units (see /H_RANGE) to move the handle for each click on an arrow button (the value must be greater than or equal to zero)
Default:	Pixel width of the current font cell
Attribute:	**/H_HANDLE**
Explanation:	Sets the displayed position of the scroll handle in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see GETEVENT].)
Arguments:	Relative position along bar (note that this is **not** a pixel count; 0 = leftmost end; rightmost end may be set with /H_RANGE)
Default:	Current scroll position
Attribute:	**/H_RANGE**
Explanation:	Sets the value of the rightmost position of the scroll handle in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see GETEVENT].)
Arguments:	Value associated with the rightmost position of the scroll handle (value must be greater than or equal to 0; if n handle positions are desired, use the value n–1)
Default:	Total number of pixels that the window can be scrolled (**not** the width of the window)

Table 2-3. FSET Graphics Window Attributes (Continued)

Attribute:	**/H_SCROLL**
Explanation:	Sets the horizontal scroll position of the displayed contents of the window. (This keyword is ignored if it is specified with FOPEN and the window already exists.)
Arguments:	Horizontal offset (in pixels) in the bitmap to the first column to display (the value must be greater than or equal to zero)
Default:	/H_SCROLL 0
Attribute:	**/IGNORE**
Explanation:	Specifies pull-down menu items that cannot be selected. These items are dimmed in the pull-down menu display. This keyword must be specified in the same FSET instruction as the /PULLDOWN keyword.
Arguments:	List of numbers in the range 1 to 30, which correspond to rows in the pull-down menu. Numbers greater than the number of pull-down items are ignored.
Default:	No pull-down items are ignored
Attribute:	**/LUT**
Explanation:	This attribute is ignored by the AdeptWindows PC interface. Sets the red, green, and blue (RGB) values of a color. The specified color is actually altered for the entire graphics system. Thus, this keyword must be used with care. Note, for example, that changing color values may cause all the color descriptions in this manual to become invalid.
Arguments:	The color to be changed (in the range 0 to 15), followed by the red, green, and blue values for the color (each in the range 0 to 255). A value of –1 may be specified for the red, green, or blue value to return that value to its default. For instance, specifying /LUT 15 127 127 127 will change the default white (color #15) in the system to a gray, and specifying /LUT 15 –1 –1 –1 will return the system to its default color.
Default:	System default colors (see GCOLOR)
Attribute:	**/MARGINS**
Explanation:	Sets the text margins for the window. This keyword also sets the text cursor position to the top left corner of the text window defined by the new margins. The text scrolling window is reset to its maximum size.
Arguments:	Coordinates of the left, top, right, and bottom edges of the text window (if a value is negative, the corresponding margin is not modified)

Table 2-3. FSET Graphics Window Attributes (Continued)

Default:	• X coordinate of left margin = 8 • Y coordinate of top margin = 8 • X coordinate of right margin = (right_edge_of_window – 8) • Y coordinate of bottom margin = (left_edge_of_window – 8)
Attribute:	**/MENU**
Explanation:	Specifies up to 20 items to appear in the menu bar of the window. Each menu string is displayed in a box in the menu bar, left to right in the order specified. A space is added to each side of each string. If the strings are too long, the excess falls off the end of the menu bar without any error indication. The strings are written in the current font defined for the menu bar. (This keyword is ignored if the window does not have a menu bar.)
Arguments:	List of up to 20 strings, each one enclosed in single quotes. Each string delimited by single-quote characters must be separated from the next string by at least one space. The correct way to clear the menu bar is to specify no strings, not a null string (that is, with /MENU, not with /MENU ' ').
Default:	/MENU (no items)
Attribute:	**/POINTER**
Explanation:	Specifies the point of reference for position information obtained from the mouse cursor. This keyword takes effect with the next event that occurs—it does not change any events that have already occurred but are waiting to be read.
Arguments:	• DISPLAY = Return pointer device positions relative to the upper left-hand corner of the window region displayed on the screen. • SCREEN = Return pointer device positions relative to the upper left-hand corner of the entire screen. • WINDOW = Return pointer device positions relative to the upper left-hand corner of the allocated window, even if the corner is scrolled out of the displayed region.
Default:	/POINTER WINDOW

Table 2-3. FSET Graphics Window Attributes (Continued)

Attribute:	**/POSITION**
Explanation:	Sets the displayed position of the window relative to its parent. The arguments define the position of this window's upper left-hand corner relative to the upper left-hand corner of the parent's allocated window. If this is a top-level window, the position is relative to the upper left-hand corner of the screen. (This keyword may not be specified for control-bar windows.) This keyword is ignored if it is specified with FOPEN and the window already exists. If this keyword is used with FSET, the value specified overrides any mouse drag that may have been done by the user. This keyword may also specify the position of the reference point of an icon relative to the top left corner of the icon (see the /ICON keyword for FOPEN).
Arguments:	X position (in pixels) relative to parent (always rounded up to be an even number) Y position (in pixels) relative to parent
Default:	/POSITION 0 0
Attribute:	**/PULLDOWN**
Explanation:	Outputs text into a pull-down window, and displays the pull-down window under the specified menu-bar item if that item is currently selected (highlighted). (This keyword may not be specified with FOPEN.) All the strings specified are displayed, left justified, stacked vertically, in a window the width of the longest string (plus a space on each side of the string). For VGB-equipped systems, with the standard font, the pull-down window may have a maximum area equivalent to 25 lines of text, 25 characters wide. There can be a maximum of 30 lines of text. If there are more than 25 lines, the maximum pull-down will be less than 25 characters wide; conversely, if there are fewer than 25 lines, the pull-down can be more than 25 characters wide. Each line of text will be truncated on the right if necessary. For systems using the AdeptWindows PC interface, the limit is determined by the FSET attribute list—the maximum length is 512 characters. See the /IGNORE keyword for information on how to disable pull-down items. Only one pull-down window can be present in the entire system at any one time.
Arguments:	Number of the menu-bar item with which this pull-down list is associated (1 [leftmost item] to N [rightmost item]) followed by a list of strings (each in single quotes) separated by spaces

Table 2-3. FSET Graphics Window Attributes (Continued)

Default:	No pull-down menu specified
Attribute:	**/SELECT**
Explanation:	Specifies that a window is selected for receiving input. The effect of this keyword is equivalent to clicking the mouse on the window.
Arguments:	None
Default:	The window is not selected
Attribute:	**/SHOW**
Explanation:	This attribute is ignored by the AdeptWindows PC interface. Forces a window and all its ancestors to be displayed. (This keyword differs from /DISPLAY—that keyword does not display a window if any one of its ancestors is not displayed.)
Arguments:	None
Default:	None
Attribute:	**/SIZE**
Explanation:	Sets the size of the window in the display. The size specified overrides any mouse drag that may have been done by the user. This keyword may not be specified for control-bar windows. (This keyword is ignored if it is specified with FOPEN and the window already exists.) The X value is always rounded up to be an even number. For all windows, X and Y values less than 8 are set to 8. The values of X and Y are clipped to the full size of the window as required. This keyword may also specify the size of an icon or font (see the /ICON or /DEFFONT keyword for the FOPEN instruction).
Arguments:	• X size (in pixels) • Y size (in pixels)
Default:	Display the whole window (see /MAXSIZE described in the FOPEN instruction)
Attribute:	**/SPECIAL**
Explanation:	Except for /SPECIAL NOSELECTABLE, this attribute is ignored by the AdeptWindows PC interface. Sets special attributes for the window.

Table 2-3. FSET Graphics Window Attributes (Continued)

Arguments:	One or more of the following keywords. • DESELECT = Allow deselection • NODESELECT = Don't allow deselection • POSITION = Allow moving by dragging title bar • NOPOSITION = Don't allow dragging of title bar • SELECTABLE = Allow window to be selected from pull-down menu under Adept icon • NOSELECTABLE = Don't allow window to be selected from pull-down menu under Adept icon. (Window name is dimmed in pull-down menu.) • SIZE = Allow resize with sizing icon • NOSIZE = Don't allow resize with sizing icon
Default:	/SPECIAL DESELECT POSITION SELECTABLE SIZE
Attribute:	**/STACK**
Explanation:	Sets the position of the window in its window stack, which determines the layering of the windows in the display. This keyword may not be specified for control-bar windows. No child of a window can be stacked below that window's control bars.
Arguments:	Code value for desired position: • 1 Move to top of parent's stack • –1 Move to bottom of parent's stack
Default:	/STACK 1
Attribute:	**/TERMINAL**
Explanation:	Sets characteristics of the terminal emulator used to output text to the window.

Table 2-3. FSET Graphics Window Attributes (Continued)

Arguments:	One or more of the following keywords. • CURSOR = Display the text cursor • NOCURSOR = Do not display text cursor • OVERSTRIKE = Force designated overstrike characters to overstrike the previous character • NOOVERSTRIKE = Defeat overstriking • WRAP = Wrap lines of text that are too long • NOWRAP = Truncate long lines of text
Default:	/TERMINAL CURSOR OVERSTRIKE WRAP
Attribute:	**/TITLE**
Explanation:	Set the window title, which is the text that is displayed in the title bar. The title has no other meaning to the V+ system.
Arguments:	String enclosed in single quotes. The string will be truncated to 40 characters.
Default:	Same as the window name
Attribute:	**/UPDATE and /NOUPDATE**
Explanation:	In normal operation, the screen display is updated every time new graphics appear in any displayed window. The /NOUPDATE keyword stops this process for the window currently open, thereby hiding all new graphics from the screen display until normal operation resumes. The /UPDATE keyword will force the screen display to be updated and normal operation to resume. (This keyword may not be specified with the FOPEN instruction.)
Arguments:	None
Default:	/UPDATE
Attribute:	**/V_ARROWINC**
Explanation:	Sets the change in the position of the scroll handle in the vertical scroll bar that will be caused by clicking on the arrow buttons in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL is enabled [see the GETEVENT instruction].)
Arguments:	Number of units (see V_RANGE) to move the handle for each click on an arrow button (the value must be greater than or equal to zero)

Table 2-3. FSET Graphics Window Attributes (Continued)

Default:	Pixel height of the current font cell
Attribute:	**/V_HANDLE**
Explanation:	Sets the displayed position of the scroll handle in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL is enabled [see GETEVENT].)
Arguments:	Relative position along bar (note that this is **not** a pixel count; 0 = topmost end; bottommost end may be set with /V_RANGE)
Default:	Current scroll position
Attribute:	**/V_RANGE**
Explanation:	Sets the value of the bottommost position of the scroll handle in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL event is enabled [see GETEVENT].)
Arguments:	Value associated with bottommost position of the scroll handle (the value must be greater than or equal to 0; if n handle positions are desired, use the value n–1)
Default:	Total number of pixels that the window can be scrolled (not the height of the window)
Attribute:	**/V_SCROLL**
Explanation:	Sets the vertical scroll position of the displayed contents of the window. (This keyword is ignored if it is specified with FOPEN and the window already exists.)
Arguments:	Vertical offset (in pixels) in the bitmap to the first row to display (the value must be greater than or equal to zero)
Default:	/V_SCROLL 0

Using FSET With Serial Lines

The following specifications can be used as arguments to directly ATTACH a serial line:

LOCAL.SERIAL:n
: Local serial line n on the local CPU. For Adept CPU boards, n = 1, 2, 3, or 4.

SERIAL:n
: Global serial line n on the Adept SIO board. For Adept SIO board, n = 1, 2, 3, or 4.

As a convenience, the following synonyms may be used:

KERMIT The serial line currently configured for Kermit protocol.

MONITOR The serial line currently configured for use as the monitor terminal.

The keywords listed in **Table 2-4** may appear in the keyword list string.

Table 2-4. FSET Serial-Line Attributes

Attribute	Argument	Description
/PARITY	NONE	No parity generation
	EVEN	Use even parity
	ODD	Use odd parity
/STOP_BITS	1 or 2	Use 1 or 2 stop bits per byte
/BYTE_LENGTH	7 or 8	Use 7 or 8 bits per byte
/FLOW	NONE	No flow control
	XON_XOFF	Detect and generate XON/XOFF (turn off modem)
	MODEM	Use modem control RTS/CTS (turn off XON_XOFF). Available only for LOCAL.SERIAL 2 and 3, and SERIAL 1, 2, and 3.
/DTR	OFF	Turn off the DTR modem signal. Available only for LOCAL.SERIAL 2 and 3, and SERIAL 1, 2, and 3.
	ON	Turn on the DTR modem signal. Available only for LOCAL.SERIAL 2 and 3, and SERIAL 1, 2, and 3.
/MULTIDROP	OFF	Do not use multidrop mode
	ON	Use multidrop mode (Valid only for LOCAL.SERIAL:1 on Adept CPUs)
/FLUSH	OFF	Disable recognition of CTRL+O for flushing output
	ON	Enable recognition of CTRL+O for flushing output

Table 2-4. FSET Serial-Line Attributes (Continued)

Attribute	Argument	Description
/SPEED	110, 300, 600, 1200, 2400, 4800, 7200, 9600, 19200, 38400	Select the indicated baud rate. For the Adept SIO board, a baud rate of 19200 is incompatible with a baud rate of 7200 or 38400 on a pair-wise basis. The pairs are: SERIAL:1 and SERIAL:4 SERIAL:2 and SERIAL:3

The drivers for KERMIT, DDCMP, and NETWORK do not support all the modes indicated by the keywords; they ignore those that are not supported.

Using FSET With NFS and TCP

The following AdeptNet devices may be referenced with the FSET instruction:

NFS Network File System

TCP Transmission Control Protocol

You can use the attributes listed in **Table 2-5** when accessing these devices with the FSET instruction.

Table 2-5. FSET Attributes for AdeptNet

Attribute	Description
/ADDRESS	IP address. (Applies only to the TCP device.)
/MOUNT	Defines the name to be used for an NFS server remote disk. (Applies only to the NFS device.)
/NODE	Node name.
/PATH	Path for NFS server remote disk. (Applies only to the NFS device.)

You may define new nodes on the network using the FSET program instruction to access a logical unit that has been attached to the TCP device. The string used with the FSET instruction has the same format as that used with the NODE statement in the V+ configuration file (see the later example).

You may also define new remote mounts, using the FSET program instruction to access a logical unit that has been attached to the NFS device. The string used with the FSET instruction has the same format as that used with the MOUNT statement in the V+ configuration file (see the later example).

Examples

Graphics

The following FSET instruction causes a pull-down menu to be displayed under the menu-bar item just selected by the user (assuming that the array element event[2] has been set by a GETEVENT instruction that detected a menu event):

```
FSET (lun) "/PULLDOWN", event[2], $menu[event[2]]
```

This instruction requires that string array elements $menu[n] be defined for each of the menu items displayed on the menu bar. The following line shows how a pull-down list with four items could be defined for the first item on the menu bar. (Note that the menu items can have different lengths, they can consist of multiple words, and there is a space between the single quotes delineating items.)

```
$menu[1] = "'Item #1' 'Second Item' 'One more item' 'Last
item'"
```

The following instruction takes control of the vertical scroll bar away from the window manager. The vertical scroll bar is set to return values from 0 to 10. The initial position of the scroll-bar handle is at the top (zero) and each time the up-arrow or down-arrow icon is clicked, the value of the scroll-bar handle position changes by 2 units.

```
FSET (lun) "/EVENT GRAB_V_SCROLL/V_RANGE 10/V_HANDLE
0/V_ARROWINC 2"
```

The following example attaches serial line 2 and sets the baud rate to 2400:

```
ATTACH (slun, 4) "SERIAL:2"

FSET (slun) "/SPEED 2400"
```

AdeptNet

Define a new node called SERVER2 with the IP address 172.16.200.102:

```
ATTACH (lun, 4) "TCP"
FSET (lun) "/NODE 'SERVER2' /ADDRESS 172 16 200 102"
```

Define a new NFS mount with the disk name DISK2 to access the exported directory `/a` of the node (server) called SERVER1 (which must already be defined):

```
ATTACH (lun, 4) "NFS"
FSET (lun) "/MOUNT 'DISK2' /NODE 'SERVER1' /PATH '/a'"
```

Related Keywords

FOPEN (program instruction)
IOSTAT (real-valued function)

Syntax

HERE `@task:program` **`loc_variable`**

Function

Define the value of a transformation or precision-point variable to be equal to the current robot location.

Usage Considerations

If no task is specified, the HERE monitor command returns information for the robot selected by the V+ monitor (with the SELECT command). If a task is specified, the command returns the location of the robot selected by that task (with the SELECT instruction).

Interactive modification of the location value is not available when the V+ system is operating in Adept Digital Workcell simulation mode.

If the V+ system is not configured to control a robot, use of the HERE command will cause an error.

Parameters

`@task:program` These optional parameters specify the context for the location variable. The location variable will be treated as though it is referenced from the specified context. If no context is specified, the location variable will be considered global (if the V+ program debugger is not in use).

`loc_variable` Transformation, precision point, or a compound transformation that ends with a transformation variable.

Details

This command defines the value of a transformation or precision-point variable to be equal to the current robot location.[1]

After the new value of the variable has been set, its components are displayed and may be modified by typing new values after the query "Change?" appears. Component values must be separated by commas; values that are not being changed may be omitted. Changes are asked for repeatedly until no change is made.

[1] Normally, the robot location is determined by reading the instantaneous values of the joint encoders. However, if the robot has either backlash or linearity compensation enabled, the commanded robot location is used instead.

If a precision point is being defined, all its joint variables are displayed. Otherwise, a transformation is defined, and its value is displayed as X, Y, Z, y, p, r—where X, Y, Z specify the position of the tool point (for example, a point centrally located between the fingertips) in World coordinates, and y, p, r (yaw, pitch, and roll, respectively) specify the orientation of the tool.

If a compound transformation is specified, only the rightmost element of the compound transformation will be given a value by this command. An error message results if any other transformation in the compound transformation is not already defined.

Examples

Define the transformation "place" to be equal to the current robot location.

```
HERE place
```

Assign the current location of the robot to the precision point "#pick", which is treated as a local variable in the program "test".

```
HERE @test #pick
```

Related Keywords

HERE (program instruction and transformation function)

POINT (monitor command)

SELECT (monitor command and real-valued function)

WHERE (monitor command)

Syntax

```
ID device
```

Function

Display identity information about components of the system.

Parameter

`device` Optional integer value from 1 to 4 that determines which system component should be identified (see below). If this parameter is omitted, V+ performs an ID 1 followed by an ID 3 and ID 4.

Details

When the ID command is issued with no device parameter, the following information is displayed:

- If a user-defined startup message has been defined, that message is displayed.
- If the system is operating in Adept Digital Workcell simulation mode, a message is displayed to indicate that mode.
- The output from the following ID commands is displayed: ID 1, ID 3, ID 4.

The ID information for each device code is described below.

The option words contain coded information about the system (formatted as hexadecimal values). This information is useful to Adept support personnel when troubleshooting your system.

> **NOTE:** For detailed information on the option words, see appendix C of the ***V+ Language Reference Guide.***

Output for ID 1:

```
Software: version.revision opt1-opt2 edit_message
```

`version` is the V+ version number, which is the same value that is returned by the function ID(3).

`revision` is the software revision number, which is the same value that is returned by the function ID(4).

`opt1` is option word 1, which is the same value that is returned by the function ID(5).

`opt2` is option word 2, which is the same value that is returned by the function ID(6).

`edit_message` is the identifying string returned by the function $ID(–1).

`Controller: model-serial options`

`model` is the controller model number, which is the same value that is returned by the function ID(1).

`serial` is the controller serial number, which is the same value that is returned by the function ID(2).

`options` is the option word, which is the same value that is returned by the function ID(8).

`Processor n: version.revision type-software memMb`

`n` is the CPU number, which is the same value that is returned by the function ID(1,4).

`version` is the CPU hardware version code, which is the same value that is returned by the function ID(3,4).

`revision` is the CPU hardware revision code, which is the same value that is returned by the function ID(4,4).

`type` is the CPU type, which is the same value that is returned by the function ID(5,4).

0 = Heurikon 68040 (obsolete)
1 = Adept 68030 (obsolete)
2 = Motorola 68040 (obsolete)
3 = Motorola 68060 (obsolete)
4 = Adept AWC 68040
5 = Adept AWC 68060

`software` is the active system software components on this processor, which is the same value that is returned by ID(6,4).

Bit 1 = V+
Bit 2 = Vision
Bit 3 = Servo

`mem` is the RAM size, in megabytes, which is the same value that is returned by the function ID(7,4) divided by 1024.

(Repeated for all processors seen)

```
Robot n: model-serial opt1-opt2 module
```

`n` is the robot number, which is the value that must be used to reference this robot in the ID functions mentioned below and in the SELECT monitor command and program instruction.

`model` is the robot model number, which is the same value that is returned by the function ID(1,10+`n`).

`serial` is the robot serial number, which is the same value that is returned by the function ID(2,10+`n`).

`opt1` is option word 1, which is the same value that is returned by the function ID(8,10+`n`).

`opt2` is option word 2, which is the same value that is returned by the function ID(11,10+`n`).

`module` is the kinematic device module number, which is the same value that is returned by the function ID(5,10+`n`).

(Repeated for all robots seen)

Output for ID 2:

```
Manual Control Pendant: version
```

`version` is the Manual Control Pendant (MCP) version number

Output for ID 3:

```
Vision n: version.revision model options memMb edit_message
```

`n` is the number of the vision system, which is the value that must be used to reference this vision system in the ID functions mentioned below and in the SELECT monitor command and program instruction.

`version` is the vision software version number, which is the same value that is returned by the function ID(3,3).

`revision` is the vision software revision number, which is the same value that is returned by the function ID(4,3).

`model` is the vision model number, which is the same value that is returned by the function ID(1,3).

`options` is the vision option word, which is the same value that is returned by the function ID(5,3).

`mem` is the RAM size, in megabytes, which is the same value that is returned by the expression ID(6,3)/1024

`edit_message` is the identifying string that is returned by the function $ID(–3) .

Output for ID 4:

```
Servo: version.revision edit_message
```

`version` is the servo software version number, which is the same value that is returned by the function ID(3,5).

`revision` is the servo software revision number, which is the same value that is returned by the function ID(4,5).

`edit_message` is the identifying string that is returned by the function $ID(–4).

Example

Figure 2-1 is a sample display from the command ID:

```
Configured as Category 3 Robot System per ISO 10218 and EN954

Software:    13.0 887-51E0  (Edit E, 22-Jun-1998, Preliminary Release)
Controller:  3405-217 0
Processor 1: 0.0 4-7 8MB
Robot 1:     862-106 0-15B 5
..
```

Figure 2-1. ID Command Sample Output

Please have the information for your system available if you call Adept customer service concerning your system.

Related Keyword

ID (real-valued function)

Syntax

```
ID (component, device, board)
```

Function

Return values that identify the configuration of the current system.

Parameters

`component` Real value, variable, or expression (interpreted as an integer) whose value determines which component of identification information is returned.

`device` Optional real value, variable, or expression (interpreted as an integer) whose value selects the device to be identified. Device #1 (the basic system) is assumed if this parameter is omitted.

`board` Optional integer specifying the CPU of interest when `device` = 4. Board #1 (the main CPU) is assumed if this parameter is omitted.

Details

The ID function enables a program to access the information displayed by the ID monitor command. The values of the components are the same as the fields displayed by that command.

The function returns the value 0 for devices that do not exist.

Device number 1 refers to the **basic system**. The acceptable values for the **`component`** parameter, and the corresponding values returned, are:

component	Result of ID (component, 1)
1	Model designation of the system controller
2	Serial number of the system controller
3	Version number of the V+ software in use
4	Revision number of the V+ software in use
5	First option word for the V+ system (*)
6	Second option word for the V+ system (*)
7	Size of the system program memory (in kilobytes, K [1 K = 1024 8-bit bytes])
8	Controller option word (*)

component	Result of ID (component, 1)
9	Controller product-type value
10	(Value for internal use by Adept)
11	Controller hardware configuration (see below)

*The system and controller option words are described in Appendix C in the ***V+ Language Reference Guide***.

The value for the controller hardware configuration [returned by ID(11,1)] should be interpreted as a bit field, as follows:

Bit 1 (LSB) Graphics system processor (mask = 1)

This bit is set if the system processor in the controller includes hardware components for graphics output by the V+ system.

Bit 2 V+ monitor outputs to graphics monitor (mask = 2)

This bit is set if the system processor supports graphics output and the configuration switches on the system processor are set to direct the V+ monitor output to the graphics monitor. This bit is 0 if the V+ monitor output is directed to a terminal connected to the system controller.

Bit 3 Not used

Bit 4 SIO board present (mask = 8)

This bit is set if the SIO board is present in the controller.

Bit 5 AdeptWindows (mask = 16)

This bit is set if the controller is using AdeptWindows.

Bit 6 i960 hardware as an I/O coprocessor (mask = 32)

When this bit is set, it indicates the system has functional i960 hardware as an I/O coprocessor.

Bit 7 i960 hardware and software is running (mask = 64)

When this bit is set, it indicates the system has functional i960 hardware and its software is running.

Bit 8 Device B floppy board present (mask = 128)

This bit is set if a device B floppy interface board is present in the controller.

NOTE: This bit indicates only that the interface board is present. It does not indicate that the floppy drive is installed.

Device number 2 refers to the manual control pendant. The **component** parameter must have the value 1. The value returned is the version number of the pendant software.

Device number 3 refers to the AdeptVision system. (If the vision system is not installed, all values are returned as zero.) The acceptable values for the component parameter, and the corresponding values returned, are listed below. (If the system has multiple vision systems, the information returned is for the vision system that is currently selected. The number of the vision system that is currently selected can be obtained with the function SELECT(VISION).)

component	Result of ID (component, 3)
1	Model designation of the vision system
2	Serial number of the vision system
3	Version number of the vision software in use
4	Revision number of the vision software in use
5	Option word for the vision system (see Appendix C in the *V+ Language Reference Guide*)
6	Size of the vision system RAM memory (in K)

Device number 4 refers to the system CPUs. (If the indicated board does not exist, all values are returned as –1.)

component	Result of ID (component, 4, board)
1	Number of the CPU
2	Not currently used
3	The CPU board version number
4	The CPU board revision number
5	CPU type: 1 = 68030; 0, 2 = 68040; 3= 68060; 4 = AWC 68040; 5 = AWC 68060

component	Result of ID (component, 4, board)
6	Software modules active on the board: Bit field where bit 1 represents the V+ operating system, bit 2 represents the vision software, and bit 3 represents the servo/motion software
7	Size of the random access memory (RAM) on the CPU board (in K)

Device number 5 returns servo information. (If no servo is present, all values are returned as zero.)

component	Result of ID (component, 5)
3	Returns the servo version number
4	Returns the servo revision number

Device numbers 6 and 7 are invalid and return the value 0, as do any other specified devices that do not exist. For valid devices, an *Invalid argument* error message is reported if the requested component is not valid.

Device number 8 returns information for the currently selected robot. The acceptable values for the **component** parameter are the same as for a robot.

The values returned for device number 9 are for Adept's internal use.

Device number 10 refers to the external encoders connected to the robot controller. The acceptable values for the **`component`** parameter are the same as for a robot.

Device numbers 11, 12, ... refer to robot number 1, 2, ..., respectively, for each robot connected to the controller. That is, a device number equal to 10+*r* refers to robot number *r*, which can range from 1 to the value returned by the function SELECT(ROBOT, –1). The number of the robot that is currently selected can be obtained with the function SELECT(ROBOT).The acceptable values for the **`component`** parameter, and the corresponding values returned, are listed below.

component	Result of ID (component, 10+*r*)
1	Model designation of the robot
2	Serial number of the robot
3	Number of motors configured for the robot. This normally is equal to the number of configured joints.

component	Result of ID (component, 10+*r*)
4	Value interpreted as bit flags for the robot joints that are enabled: bit 1 for joint 1, and so on. (The value is zero if the robot does not have joints that can be disabled selectively. For example, this value is defined for the X/Y/Z/Theta robot but is zero for the 4/5-axis SCARA module.)
5	Robot control-module identification number
6	Number for the robot control-module qualifier
7	Number of robot joints configured for use
8	Robot option word (*)
9	Robot product-type value
10	Number of Cartesian axes used during motion planning. This value specifies how many values must be defined in the robot load-module arrays that specify the maximum Cartesian velocities and accelerations.
11	Second robot option word (*)
12	Information on the robot module. Currently, only bit 1 (mask 1) is defined. If set, this bit indicates that the specified robot is an Adept robot.

*The robot option words are described in Appendix C in the ***V+ Language Reference Guide***.

Device number 50 refers to the currently selected force sensor. Device numbers 51 through 66 refer to force sensors numbered 1 to 16, respectively.

See the documentation for the SELECT program instruction for an explanation of selecting among multiple force sensors. The acceptable values for the **`component`** parameter, and the corresponding values returned, are listed below.

component	Result of ID (component, 50)
1	Model number of force sensor (0 if no force sensor is connected)
2	Serial number of force sensor (0 if no force sensor is connected)
3	Version number of VME Force Interface (VFI) firmware
4	Version number of force-sensing software

component	Result of ID (component, 50)
5	Option word for force system
6	Size of the data collection buffer (in K)

Related Keywords

ID	(monitor command; see the *V+ Operating System Reference Guide*)
$ID	(string function; see the *V+ Language Reference Guide*)
SELECT	(monitor command, program instruction, and real-valued function; see the *V+ Language Reference Guide*)

Syntax

```
IF logical_expr GOTO label
```

Function

Branch to the specified step label if the value of the logical expression is TRUE (nonzero).

Usage Considerations

In general, it is a better programming practice to use the IF ... THEN control structure rather than this instruction.

Parameters

`logical_expr` Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (zero).

`label` Program step label of a step in the current program.

Details

If the value of the expression is nonzero, program execution branches and begins executing the statement with a label matching the one specified. If the value of the expression is zero, the next instruction is executed as usual.

If the specified statement label is not defined, the program is not executable. Any attempt to branch to an undefined label is identified when the SEE editor is exited and when the program is loaded into memory from a disk file.

Example

The most common use for IF...GOTO is as an exit-on-error instruction. The following code checks each I/O operation and branches to a label whenever an I/O error occurs:

```
ATTACH(dlun, 4) "DISK"
IF IOSTAT(dlun) < 0 GOTO 100

FOPENW(dlun) "my_file"
IF IOSTAT(dlun) < 0 GOTO 100

...

FCLOSE(dlun) "my_file"
IF IOSTAT(dlun) < 0 GOTO 100
```

```
     DETACH(dlun)
100  IF IOSTAT(dlun) < 0 THEN
          TYPE $ERROR(IOSTAT(dlun))
     END
```

Related Keywords

GOTO	(program instruction)
IF ... THEN	(program instruction)

Syntax

INSTALL password op

Function

Install or remove software options available to Adept systems.

Usage Considerations

You must have received the authorization password from Adept. INSTALL can be used only on CPU #1 in multiple CPU systems.

Parameters

password	A 15-character string assigned by Adept
op	Optional integer indicating the desired operation: 0 = install option (default) 1 = remove option

Details

When you purchase additional software options from Adept, the software is delivered with a software license and authorization code that enables the software for a particular controller. If the option is not enabled, the software option will not be available for use.

The password is keyed both to the software option and the serial number of your controller. The password cannot be used on any controller other than the one for which you purchased the software option.

> **NOTE:** If you replace and initialize the AWC board in your controller, you must either (a) move the socketed NVRAM device from your old AWC board to the new one, or (b) reinstall all the software options. To reinstall the software options, issue an INSTALL command for each option after V+ is running on the new AWC board.

Example

If you purchase a software option from Adept and the password provided with the option is 4EX5-23GH8-AY3F, the following command will enable the software option:

```
INSTALL 4EX5-23GH8-AY3F
```

NOTE: Some options, such as AIM software, have additional software disks that must be copied to the hard drive. Other options, such as AdeptVision VME, are already resident in the V+ system and need only to be enabled.

Related Keyword

INSTALL (program instruction)

Syntax

LOAD qualifier **file_spec**

Function

Load the contents of the specified disk file into the system memory.

Parameters

`qualifier` Optional qualifier whose possible values are shown below.

Switch Value	Purpose
/Q	Suppress the listing of program names to the monitor screen when the file is loaded.
/S	Squeeze programs as they are loaded into memory in much the same way that the SQUEEZE utility program operates on program files. All in-line comments and full-line comments are deleted with the exception of full-line comments that begin with the character sequence ";*".
/R	Force all of the loaded programs to be in read-only mode except when other considerations dictate that a more secure mode be utilized.



CAUTION: When a file is loaded with the /S switch, you should also use /R to prevent others from mistakenly editing the squeezed version of a file and then saving back their changes and possibly overwriting the unsqueezed version.

NOTE: When a file is loaded in read-only mode, only the first program in the file is listed on the monitor screen (if /Q was not specified to suppress all program names).

file_spec File specification for the disk file from which programs and variables are to be loaded. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension.

The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command).

If no filename extension is specified, the extension ".V2" is appended to the name given if a local disk or NFS file is to be accessed.

Details

This command loads the contents of the specified disk file into the system memory. The disk file can contain programs and/or variables, but it must have the format produced by the V+ STORE_ commands.

If an attempt is made to load a program that has the same name as a program already in memory, an error message is displayed, and the new program is *not* loaded. The currently loaded program must be DELETEd, or memory must be ZEROed, before a new program of the same name can be LOADed.

If a location variable, real variable, or string variable already exists in memory that has the same name as one contained in the disk file, the previous variable is deleted and replaced by the information on the diskette *without* warning.

If a program is being loaded into the system and there is a line that V+ cannot process, an error message appears on the monitor screen, and the line is made into a "bad line", marked with a question mark. You can then use one of the V+ editors to modify the program after it is completely read into memory. This is useful, for example, when you load a program that was composed off-line.

When a LOAD command loads programs into memory, all the programs are entered in a program module with the same name as the first program read from the disk file. The program module is created if it does not already exist. The programs loaded are entered into the module in the order they are read from disk. (See the description of the MODULE command for an explanation of program modules.)

The autostart feature available with Adept V+ systems provides a means of having a LOAD command issued automatically when the robot system is powered on and V+ is loaded from disk. See "Program Autoload" in the ***V+ Operating System User's Guide*** for details.

If the file is in special, binary program format, the LOAD command automatically applies the special handling the file requires.

> **NOTE:** Some Adept software options are restricted to the controller(s) the software license was purchased for. These files cannot be loaded by any other controller. If you have trouble loading software for which you have purchased a license, make sure you have correctly executed an INSTALL command for the license.

Examples

```
LOAD C:pallet
```

Loads the contents of the file PALLET.V2 from disk C (the optional hard drive).

```
LOAD f3.pg
```

Loads all the programs contained in file F3.PG into the system memory.

Related Keywords

DEFAULT	(monitor command)
MODULE	(monitor command)
STORE	(monitor command)
STOREL	(monitor command)
STOREM	(monitor command)
STOREP	(monitor command)
STORER	(monitor command)
STORES	(monitor command)

Syntax

```
MDIRECTORY /M module_name
```

Function

Display the names of all the program modules in the system memory or the names of the programs in a specified program module.

Parameters

`/M`	Optional qualifier that specifies that only modified modules or modified programs within a module are listed.
`module_name`	Optional name of a program module in memory. All the modules in memory are listed if this parameter is omitted. If this parameter is specified, all the programs in the named module are listed.

Details

This command can be used to obtain information about the program modules currently defined in the system memory. Program modules are automatically created by the LOAD command. The MODULE command can be used to create, expand, or rearrange a program module.

When the command parameters are omitted, the MDIRECTORY command lists the names of all program modules currently in memory.

For any module or program name that is displayed, if the copy in memory has been modified since last being loaded or stored, an "M" appears before the program or module name.

When the `module_name` parameter is specified, the MDIRECTORY command acts like a DIRECTORY command, except that only programs in the specified module are listed—in the sequence they follow in the module. (See DIRECTORY for details of the information displayed.) The order of programs in a module can be changed with the MODULE monitor command.

Examples

Display the names of all of the program modules in memory.

```
MDIRECTORY
```

Display the names of all of the programs in the program module named **main.package**.

```
MDIRECTORY main.package
```

Related Keywords

DELETEM	(monitor command)
DIRECTORY	(monitor command)
LOAD	(monitor command)
MODULE	(monitor command)
STOREM	(monitor command)

Syntax

NET mode

Function

Display status information about the AdeptNet option. Also display details about the remote mounts that are currently defined in the V+ system.

Parameter

mode	Additional error-count information is displayed if this optional parameter is specified with a value other than 0.

Details

This command displays information about the network service protocols that are currently installed and are available. These include AdeptTCP/IP, AdeptNFS, and AdeptFTP. The command also displays the local IP address (i.e., the address of the Adept controller).

There can be any one of four states (three in the case of AdeptFTP):

State message	Explanation
`Up`	For AdeptTCP/IP and AdeptNFS, this indicates that the network has initialized successfully and is running.
`Option installed`	The software option is installed, but the protocol is not currently active. (In V+ versions prior to V+ 11.2G1 the message `Initializing` was used.)
`Hardware not installed`	The Ethernet cable is not connected.
`Option not installed`	The AdeptNet option (FTP, NFS, TCP/IP) is not installed.

If NFS or TCP is in the Up state, additional information is also displayed as shown in the following example.

Example

With the AdeptNFS option installed, and with a connection to an NFS server established, typing **net** produces the following display on the monitor screen:

```
TCP/IP:  Up
   NFS:  Up
   FTP:  Option not installed
```

These messages indicate that the TCP/IP and NFS layers of the network are fully functioning and that the FTP option is not installed.

In addition, if TCP is in the Up state, the following appears:

```
Local IP address                    address

Packets transmitted                 count
Packets received                    count
Transmission errors                 count
Reception errors                    count
Missed packets                      count
```

where `address` is a dotted-decimal IP address and `count` is a number. The following additional output is displayed only if `mode` is specified with a nonzero value.

```
Reception framing error              count
Reception CRC error                  count
Reception overflow:                  count
Reception buffer error               count
Transmission late collision          count
Transmission loss of carrier         count
Transmission retry error             count
Transmission buffer error/underflow count
```

The number of available TCP connections is displayed.

In addition, if NFS is in the Up state, the current mount definitions appear in this format:

```
Mount               Node                          Path
XA                  SERVER1                       A:\
```

This indicates that the name XA is allocated to a remote server disk. The server is the node called SERVER1. The path of the remote server disk is given in server-native syntax as “A:\”.

Related Keywords

NETWORK (real-valued function)

PING (monitor command)

Syntax

```
PING node
```

Function

Test the network connection to a node.

Usage Considerations

This command is relevant only to Adept controllers with the AdeptNet option.

Parameter

node Name or dotted-decimal IP address of the network node with which communication will be attempted. If a node name is used, it must have been defined in the V+ configuration file or by an FSET command or instruction.

Details

This command tests the network connection to a named or addressed node. If the node responds, the command displays `Success`. If the node does not respond within 5 seconds, the command displays `Node not reachable`.

Examples

To determine if a node named `server2` is successfully connected, type

```
ping server2
```

The `Success` response indicates that the connection was successful.

The response `Node not reachable` indicates that the connection was not successful.

To determine if a node whose IP address is 172.16.200.1 is connected, type

```
ping 172.16.200.1
```

Related Keywords

FSET (monitor command)

NET (monitor command)

Syntax

```
POINT @task:program loc_variable = loc_value
```

Function

Set the location variable on the left equal to the value on the right and allow interactive editing.

Usage Considerations

Interactive modification of the location value is not available when the V+ system is operating in Adept Digital Workcell simulation mode.

Parameters

`@task:program` These optional parameters specify the context for all of the variables referenced by the command. The variables will be treated as though they are referenced from the specified context. If no context is specified, the variables will be considered global (if the V+ program debugger is not in use).

`loc_variable` Transformation or precision-point variable, or a compound transformation that ends with a transformation variable.

`loc_value` Optional location variable or function (or compound transformation) of the same type as the location variable on the left.

Details

Sets the value of the location variable equal to the value on the right, providing the variable and value are the same type of location representation. That is, they must both be transformations or precision points.

If no "`loc_value`" is specified and the variable was previously defined, its value is unchanged and the user is given the opportunity to make changes to its components as described below. If the value is not specified and the variable is not defined, the variable is set equal to the value of the null transformation (0, 0, 0, 0, 0, 0).

The value of the variable is displayed and its components may be modified by typing new values after the query "Change?" appears. Component values must be separated by commas; values that are not being changed may be omitted. Changes are asked for repeatedly until no change is made.

If a precision point is being defined, all its joint variables are displayed. Otherwise, a transformation is being defined, and its value is displayed as X, Y, Z, y, p, r, where X, Y, Z specify the position of the tool point (for example, a point centrally located between the fingertips) in World coordinates, and y, p, r (yaw, pitch, and roll, respectively) specify the orientation of the tool.

If a compound transformation is specified on the left, only the rightmost element of the compound transformation will be given a value by this command. An error message results if any other transformation in the compound transformation is not already defined.

Examples

Set the value of transformation "pick1" equal to that of transformation "pick", and allow for changes.

```
POINT pick1 = pick
```

Prepare for definition or modification of the precision point "#park".

```
POINT #park
```

Related Keywords

HERE (monitor command)

SET (program instruction)

Syntax

```
PRIME task_number program(param_list), cycles, step,
          priority[i]
```

Function

Prepare a program for execution, but do not actually start execution.

Usage Considerations

PRIME resets the execution stack for the selected program execution task and cancels the context of any program that is paused for that task.

A PRIME command cannot be processed while the selected program task is already active.

This command can be used only when the CIP keyswitch is set to AUTOMATIC mode.

The `priority` array is normally used only during experimental tuning of the tasks.

Parameters

`task_number` Optional integer number that specifies which system program task is to be activated.

`program` Optional name of the program to be executed. If the name is omitted, the last program name specified in an EXECUTE command or instruction (or PRIME command for the selected task) is used.

`param_list` Optional list of constants, variables, or expressions separated by commas, which must correspond in type and number to the arguments in the .PROGRAM statement for the program specified. If no arguments are required by the program, the list is blank, and the parentheses may be omitted.

Program parameters may be omitted as desired, using commas to skip omitted parameters. No commas are required if parameters are omitted at the end of the list. Omitted parameters are passed to the called program as "undefined" and can be detected with the DEFINED real-valued function.

`cycles` Optional real value, variable, or expression (interpreted as an integer) that specifies the number of program execution cycles to be performed. If omitted, the cycle count is assumed to be 1. For

unlimited cycles, specify any negative value. The maximum loop count value allowed is 32767.

`step` Optional real value, variable, or expression (interpreted as an integer) that specifies the step at which program execution is to begin. If omitted, program execution begins at the first executable statement in the program (that is, after the initial blank and comment lines, and all the AUTO, GLOBAL, and LOCAL instructions).

`priority[]` Optional array of real values (interpreted as integers) that are used by V+ to override the default execution priority within the task. The array contains 16 elements, which specify the program priority in each of 16 one-millisecond time slices. If specified, the elements must be in the range –1 to 64, inclusive. (See the ***V+ Language User's Guide*** for the details of task scheduling.)

`i` Optional real value, variable, or expression (interpreted as an integer) that specifies the index value of the first element to be used. If omitted, the value 0 is assumed.

Details

This command prepares a program for execution. It can be considered as being the same as the EXECUTE command, except that PRIME does not actually start program execution.

After a program is primed, execution can be started with the PROCEED command.

The `priority` values override the default execution configuration for the specified program task. The specified priorities remain in effect only until the next time an EXECUTE instruction is issued for the program task. The acceptable values are:

–1	do not run in this slice even if no other task is ready to run
0	run in this task only when no other task is ready to run
1-64	run in this task according to the specified priority. 64 is the highest priority; higher priority tasks may lock out lower priority tasks for the duration of the time slice.

When setting priorities, remember:

1-31	are normal user task priorities
32-62	are used by V+ device drivers and system tasks

63	is used by the trajectory generator. Do not use 63 or 64 unless you have very short task execution times, or jerky robot motions may result.

See the EXECUTE monitor command for additional details.

Related Keywords

EXECUTE	(monitor command and program instruction)
STOREP	(monitor command)
XSTEP	(monitor command)

Syntax

```
TIME dd-mmm-yy hh:mm:ss

TIME dd-mmm-yyyy hh:mm:ss
```

Function

Set or display the date and time.

Parameter

`dd-mmm-yy hh:mm:ss`	Optional date and time specification (see below). If omitted, the time is displayed. If specified, the system clock is changed and all of the elements (except ":ss") must be included.
or	
`dd-mmm-yyyy hh:mm:ss`	

Details

The system clock is maintained automatically and should be changed only when its values are incorrect (e.g., the controller is moved to a different time zone).

The system clock is used in the following situations:

- The date and time are displayed when the V+ system is booted from disk.
- Whenever a new disk file is created, the date and time are recorded with the file name. The FDIRECTORY command displays the dates and times for files.
- The date and time are appended to the message indicating that an application program has terminated execution.
- The date and time are displayed by the TIME monitor command.
- The date and time are available to an application program by use of the $TIME() and $TIME4() string functions.

The individual elements of the date and time specification are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively
yyyy	The year (1980 to 2079)

hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if ":ss" omitted)

Example

Set the date and time to June 23, 1999, at 4:10 p.m.

```
TIME 23-JUN-99 16:10
```

Display the current date and time:

```
TIME
```

Related Keywords

TIME	(program instruction and real-valued function)
$TIME	(string function)
$TIME4	(string function)

Syntax

```
TIME time_string
```

Function

Set the date and time.

Parameter

`time_string` String expression whose value specifies the date and time to be set. The value of the string must have one of these formats (see below):

dd-mmm-yy hh:mm:ss	dd-mmm-yyyy hh:mm:ss
dd-mmm-yy hh:mm	dd-mmm-yyyy hh:mm

Details

The system clock is set equal to the value of the string expression.

The system clock is maintained automatically and should be changed only when its setting is incorrect (e.g., the controller is moved to a different time zone).

The system clock is used in the following situations:

- The date and time are displayed when the V+ system is booted from disk.
- Whenever a new disk file is created, the date and time are recorded with the file name. (The FDIRECTORY command displays the dates and times for files.)
- The date and time are appended to the message indicating that an application program has terminated execution.
- The date and time are displayed by the TIME monitor command.
- The date and time are available to an application program by use of the $TIME() and $TIME4() string functions.

The individual elements of the date and time specification are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
yyyy	The year (1980 to 2079)

hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

Example

```
TIME "23-JUN-99 16:10:25"
```

Related Keywords

TIME	(monitor command and real-valued function)
$TIME	(string function)
$TIME4	(string function)

Syntax

```
TIME (string, select)
```

Function

Return an integer value representing either the date or the time specified in the given string parameter.

Parameters

`string` Optional string variable, constant, or expression that specifies the date and time in the format described below. (See below for details.)

`select` Real value, variable, or expression (interpreted as an integer) that selects the value to be returned. An error results if **`select`** is not one of the following:

select	Returned	Defined ss
1	date	(year-1980)*512 + month*32 + day
2	time	hour*2048 + minute*32 + second/2
3	seconds	time past the minute

Details

This function can be used to encode the date and time into compact (unsigned 16-bit) integer formats. After the integer date and time values are obtained, they can be arithmetically compared to other date and time values to determine before and after conditions.

> **NOTE:** You should not try to manipulate the encoded integer values to perform date or time arithmetic. For example, you should **not** attempt to add days to an encoded date value.

If the `string` parameter is supplied, both the date and the time must be specified in the string. The value of the string must have one of the following formats:

dd-mmm-yy hh:mm:ss	dd-mmm-yyyy hh:mm:ss
dd-mmm-yy hh:mm	dd-mmm-yyyy hh:mm

(The function returns the value –1 if the input string does not have an acceptable format [see the example below].)

The individual date and time elements are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
yyyy	The year (1980 to 2079)
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

If the `string` parameter is not supplied and the **`select`** parameter is 1, the current date of the system clock is returned. In addition, the current time of the system clock is stored in the (internal) administrative data for the program task. If the `string` parameter is not supplied and the **`select`** parameter is 2 or 3, the selected time value is returned for the system-clock time *previously* saved.

Example

The following program segment shows how the TIME real-valued function can be used to make sure a valid date and time are entered by the user after a prompt:

```
PROMPT "Enter the date and time (dd-mmm-yy hh:mm:ss): ", $time
WHILE (TIME($time,1)== –1) DO            ;Make sure it's valid
   TYPE /B,"    Could not interpret date/time."
   PROMPT "Try again (dd-mmm-yy hh:mm:ss): ", $time
END
TIME $time                               ;Set system time
```

Related Keywords

TIME	(monitor command and program instruction)
$TIME	(string function)
$TIME4	(string function)

Syntax

```
$TIME (date, time)
```

Function

Return a string value containing either the current system date and time or the specified date and time.

Parameters

`date` Optional integer value representing the year, month, and day (see below). The value is interpreted as follows (month ranges from 1 to 12):

```
date = (year-1980)*512 + month*32 + day
```

`time` Optional integer value representing the hour, minutes, and seconds past midnight (see below). The value is interpreted as follows (hour ranges from 0 to 23):

```
time = hour*2048 + minute*32 + second/2
```

NOTE: This function always returns a string containing both the date and the time. That can result in an erroneous date string if the `date` parameter is omitted when the `time` parameter is specified.

Details

If both the date and time parameters are omitted, this function returns the current system date and time in the format described below. (An empty string is returned if the system clock has not been initialized.)

If the date and time parameters are specified, their values are converted to an ASCII string in the format described below, and the string is returned. This operation is used to decode the output values generated by the TIME real-valued function.

The date and time are output in the format dd-mmm-yy hh:mm:ss, in which the individual elements are defined as follows:

dd The day of the month (1 to 31)

mmm The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)

yy The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.

hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59)

NOTE: The $TIME() function converts passed arguments (instead of the system time) when either the date or the time parameter is supplied. However, the function always tries to generate a string representation of **both** parameters. You get the date 01-Jan-80 if you do not provide a date value. The time substring will be 00:00:00 if you do not specify a time value. The following expressions can be used to return only the date and the time, respectively:

```
$date = $MID($TIME(date,),1,9)
$time = $MID($TIME(,time),11,8)
```

Related Keywords

TIME	(monitor command, program instruction, and real-valued function)
$TIME4	(string function)

Syntax

```
$TIME4 (date, time)
```

Function

Return a string value containing either the current system date and time or the specified date and time.

Parameters

`date` Optional integer value representing the year, month, and day (see below). The value is interpreted as follows (month ranges from 1 to 12):

```
date = (year-1980)*512 + month*32 + day
```

`time` Optional integer value representing the hour, minutes, and seconds past midnight (see below). The value is interpreted as follows (hour ranges from 0 to 23):

```
time = hour*2048 + minute*32 + second/2
```

NOTE: This function always returns a string containing both the date and the time. That can result in an erroneous date string if the `date` parameter is omitted when the `time` parameter is specified.

Details

If both the date and time parameters are omitted, this function returns the current system date and time in the format described below. (An empty string is returned if the system clock has not been initialized.)

If the date and time parameters are specified, their values are converted to an ASCII string in the format described below, and the string is returned. This operation is used to decode the output values generated by the TIME real-valued function.

The date and time are output in the format dd-mmm-yyyy hh:mm:ss, in which the individual elements are defined as follows:

dd The day of the month (1 to 31)

mmm The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)

yyyy The year (1980 to 2079)

hh The hour of the day (0 to 23)

mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59)

NOTE: The $TIME4() function converts passed arguments (instead of the system time) when either the date or the time parameter is supplied. However, the function always tries to generate a string representation of **both** parameters. You get the date 01-Jan-80 if you do not provide a date value. The time substring will be 00:00:00 if you do not specify a time value. The following expressions can be used to return only the date and the time, respectively:

```
$date = $MID($TIME4(date,),1,11)
$time = $MID($TIME4(,time),13,8)
```

Related Keywords

TIME	(monitor command, program instruction, and real-valued function)
$TIME	(string function)

Syntax

```
XSTEP task_number program(param_list), cycles, step,
      priority[i]
```

Function

Execute a single step of a program.

Usage Considerations

XSTEP can be used to single-step any of the system program tasks, independent of the execution status of other system tasks.

When using the program debugger, you can press CTRL+X to generate an XSTEP command for the task being debugged. (See the ***V+ Language User's Guide*** for details.)

The `priority` array is normally used only during experimental tuning of the tasks.

Parameters

`task_number`	Optional integer that specifies which system program task is to be executed. If no task number is specified: Task 0 is assumed if the system is not in DEBUG mode; the current debug task is assumed if the system is in DEBUG mode.
`program`	Optional name of the application program to be executed.[1]
`param_list`	Optional parameter list for the program. (See the description of the EXECUTE command and instruction for details.)
`cycles`	Optional real value, variable, or expression (interpreted as an integer) that specifies the number of program execution cycles to be performed. (See the description of the EXECUTE command and instruction for details.)
`step`	Optional real value, variable, or expression (interpreted as an integer) that specifies the step at which program execution is to begin.
`priority[]`	Optional array of real values (interpreted as integers) that are used by V+ to override the default execution priority within the task. The array contains 16 elements, which specify the program

[1] XSTEP commands that do not include a program name do not affect the temporary time-slice and priority parameters.

priority in each of 16 one-millisecond time slices. If specified, the elements must be in the range –1 to 64, inclusive. (See the *V+ Language User's Guide* for the details of task scheduling.)

`i` Optional real value, variable, or expression (interpreted as an integer) that specifies the index value of the first element to be used. If omitted, the value 0 is assumed.

Details

The XSTEP command can be used to execute an application program one step at a time. This is frequently useful while a program is being developed and program errors are being found and corrected.

Three aspects of program execution must be kept in mind:

1. The system program task that is to be utilized
2. The program that is to be executed
3. Program execution is stopped after one instruction.

The optional `task_number` parameter specifies which of the system program tasks is to be activated.

If any of the program arguments (`program`, `param_list`, `cycles`, or `step`) are specified, program execution is initiated in the same manner as for the EXECUTE command. Unlike that command, however, the first (executable) program instruction is displayed on the system terminal and the manual control pendant but is *not* executed. XSTEP must then be entered again, without any program arguments, to execute that instruction.

If all the command arguments are omitted, the following actions are performed:

1. The displayed program instruction is executed
2. The next instruction to be executed is displayed on the system terminal and the manual control pendant
3. The program is again stopped

As with the PROCEED and RETRY commands, an XSTEP command with no arguments can be executed only after execution has stopped due to one of the following events:

- An ABORT command or instruction is processed
- Single-step execution of the preceding program instruction
- A PAUSE instruction is executed
- A breakpoint or watchpoint is encountered

- Occurrence of a nonfatal error during program execution

The `priority` values override the default execution configuration for the specified program task. The specified priorities remain in effect only until the next time an EXECUTE instruction is issued for the program task. The acceptable values are:

–1	do not run in this slice even if no other task is ready to run
0	run in this task only when no other task is ready to run
1-64	run in this task according to the specified priority (64 is the highest priority; higher priority tasks may lock out lower priority tasks for the duration of the time slice)

When setting priorities, remember:

1-31	are normal user task priorities
32-62	are used by V+ device drivers and system tasks
63	is used by the trajectory generator. Do not use 63 or 64 unless you have very short task execution times, or jerky robot motions may result.

Examples

Initiate execution of program “pack” for three cycles as task 0 (or the current debug task). The parameters “p2” and “17” are passed to the program. The first (executable) step of “pack” is displayed in anticipation of its execution with a subsequent XSTEP command (without parameters).

```
XSTEP pack(p2,17),3
```

Prepare the program “assembly” for execution as program task 0 (or the current debug task) starting at step number 23. If “XSTEP” is then typed, step 23 would be executed.

```
XSTEP assembly,,23
```

Execute the next step of the program executing as program task 2.

```
XSTEP 2
```

Execute the next step of the program that was executing as task 0 (or the current debug task).

```
XSTEP
```

Move the execution pointer to step number 45 in the program that was executing as task 0 (or the current debug task).

```
XSTEP ,,45
```

Related Keywords

DEBUG	(monitor command)
EXECUTE	(monitor command and program instruction)
PRIME	(monitor command)
PROCEED	(monitor command)
RETRY	(monitor command)
SSTEP	(monitor command)
STATUS	(monitor command)

AdeptVision Version 13.1 3

Summary of AdeptVision Changes 106
ObjectFinder Changes 106
Keyword Changes 106
ObjectFinder Enhancements 107
Overview 107
Concurrency With Other Vision Tools 107
Position and Orientation Offset 107
Position and Orientation Limits 107
Binary and Edge Images 107
Reading ObjectFinder Model Data 108
Details on Concurrent Use 108
Details on Position and Orientation Offsets 110
Details on Position and Orientation Limits 110
Details on Binary Images 111
Details on Edge Images 113
Details on Reading ObjectFinder Model Data 115
Changes to Vision Keywords 116

Summary of AdeptVision Changes

This section provides a summary of the software changes made to AdeptVision VXL after version 13.0 was released. Each of these items is described in detail later in this chapter.

ObjectFinder Changes

The ObjectFinder was enhanced to provide new features and better performance during refinement, learning, and execution. See **"ObjectFinder Enhancements" on page 107** for details. Also, see the ***AdeptVision User's Guide*** for details on the ObjectFinder.

Keyword Changes

The following AdeptVision keywords were changed:

VSHOW.MODEL (program instruction)
This instruction has been extended to allow users to query the data stored in ObjectFinder models. The intent is that any ObjectFinder model data that can be set using a VTRAIN.FINDER mode can be read back by V+ using the corresponding mode for the enhanced VSHOW.MODEL instruction.

VTRAIN.FINDER (program instruction)
Several modes for this instruction were enabled to support ObjectFinder enhancements. They are Mode 5 to support object instance offsets; Mode 14 to support position and orientation constraints; and Mode 15 to support ObjectFinder image types.

See **"Changes to Vision Keywords" on page 116** for details.

ObjectFinder Enhancements

This section describes the enhancements to the ObjectFinder for AdeptVision VXL that were made after version 13.0 was released.[1]

Overview

This section provides a summary of the ObjectFinder enhancements. Several of these topics are detailed in later sections.

Concurrency With Other Vision Tools

Starting with version 13.1, the ObjectFinder can run concurrently with other vision tools. This allows other tools to be applied to an ObjectFinder instance based on information obtained about that part model instance, such as the position and orientation of the model instance, while the ObjectFinder continues to search for other model instances. For example, an object can now be recognized by the finder, checked with other vision tools such as rulers, then picked by the robot while other object instances are being recognized.

Position and Orientation Offset

Starting with version 13.1, the ObjectFinder allows position and orientation offsets to be applied to recognized objects. The offsets do not affect the functionality of the finder. They simply provide a way for application programmers to specify the reference point and orientation for object instances.

Position and Orientation Limits

Starting with version 13.1, the user can specify limits on the range of positions and orientations of model instances. For example, the user can specify that a model instance can rotate only plus or minus 10 degrees about the nominal orientation at which the model was trained. This improves ObjectFinder performance and reliability in applications where the objects do not occur in the scene with the full range of positions and orientations.

Binary and Edge Images

Starting with version 13.1, the ObjectFinder can work with a binary image like the prototype finder. Since the ObjectFinder is much easier to train, this should eliminate any need to use the prototype finder.

[1] Except for the changes to the VSHOW.MODEL instruction, these changes to the ObjectFinder were also made in version 12.4.

Additionally, the ObjectFinder can work with edge images that are computed using vision instructions such as VEDGE, VCONVOLVE, or VMORPH, instead of using the edge detection algorithms that are internal to the ObjectFinder. This allows application programmers to specify the image processing algorithms used by the ObjectFinder, and allows the ObjectFinder to work on images that resemble edge images.

Reading ObjectFinder Model Data

Starting with version 13.1, the VSHOW.MODEL instruction returns information about ObjectFinder models. When the model name string is the name of an ObjectFinder model, the VSHOW.MODEL instruction returns values from the ObjectFinder model in a data array. The values returned depend on the instruction mode, and the mode numbers for returning ObjectFinder data have been selected to correspond to the VTRAIN.FINDER modes that modify the corresponding data.

Details on Concurrent Use

ObjectFinder concurrency works automatically, without any new instructions or changes to existing instructions, so existing V+ programs do not have to be changed to take advantage of it. The vision system automatically detects situations where concurrency should be used and executes the sequence of vision instructions in a separate system task while the ObjectFinder VFINDER instruction is executed in its own task. After the ObjectFinder places an object instance in the queue, it continues to process the image and attempts to recognize additional model instances. A V+ program could then, for example, execute a VLOCATE instruction followed by VFEATURE instructions to obtain information about the recognized instance. This information could be used to move the robot to pick up an object while the ObjectFinder is attempting to recognize additional objects.

The following vision instructions can be executed concurrently with the ObjectFinder VFINDER instruction: VRULERI, VFIND.LINE, VFIND.ARC, VFIND.POINT, VWINDOWI, VDEF.AOI, VGET.AOI, VDEF.TRANS, and VGET.TRANS.

Before ObjectFinder concurrency was implemented (up to and including AdeptVision version 12.3 and version 13.0), none of the vision tools listed above could be executed until the VFINDER instruction was completely finished and all the objects had been found in the image. For complex, cluttered images and/or complex parts, this could cause a significant slowdown in the part-consumption cycle time. For example, in an image with several objects, the first model instance could not be inspected until the entire image had been processed and all the

objects were found. With automatic concurrency, a part can be immediately inspected with any combination of the vision tools listed above as soon as the object has been recognized and placed in the queue. Inspection of the object can proceed concurrently while the ObjectFinder recognizes other parts.

The following V+ code provides an example of concurrent execution.

```
VPICTURE (cam) 2
VFINDER (cam, 1, -1, howmany) aoi_num
VLOCATE (cam, 0, 0) $name, loc

WHILE VFEATURE(1) DO
   DECOMPOSE xform[] = loc
   x = xform[0]
   y = xform[1]
   ang = xform[5]

   VRULERI (cam, 2) ruler[] = 1, x, y, r, ang

   ; Code to test the edge points

   VFIND.LINE (cam) line[] = 1, x, y, r, r/4, ang+90

   ; Code to test the line

   VFIND.ARC (cam, 4, , , 2) arc[] = 1, x, y, r, r/2, ang, ang+270

   ; Code to test the arc


   VWINDOWI (cam, 0) data[] = 1, x, y, r, r, ang

   ; Code to test the window statistics

   VLOCATE (cam, 0, 0) $name, loc

END
```

The vision tools VRULERI, FIND.LINE, VFIND.LINE, VFIND.ARC, and VWINDOWI are executed for each part found by the VLOCATE while the ObjectFinder VFINDER instruction is executing to recognize and queue other parts. The first VLOCATE instruction (before the DO loop) causes the instruction sequence to wait until the first object is recognized and placed in the queue, the second VLOCATE instruction (at the bottom of the loop) causes the instruction sequence to wait until at least one additional object is recognized and placed in the queue. The ObjectFinder is always allowed to run while the instruction sequence waits for an object.

NOTE: There is no way to disable concurrent execution. However, a VWAIT instruction can be placed after the VFINDER instruction to force the instruction stream to wait until the finder is finished.

Details on Position and Orientation Offsets

With AdeptVision version 13.1, the application programmer can specify position and orientation offsets for an ObjectFinder model. This is done using mode 5 of the VTRAIN.FINDER instruction. The offsets are added to the position and orientation of recognized instances before they are placed in the VLOCATE queue. This enhancement allows V+ programmers to use other vision tools, such as rulers, line finders, or blob finders, to determine a desired reference point and orientation for the model (overriding the values chosen by the ObjectFinder when the model is trained).

For example, let's consider how this would apply to a rectangular part. The ObjectFinder may set the reference point at the center of the rectangle. However, the programmer may need to set the reference point to be at a particular corner of the rectangle. The programmer can use line finders or rulers to calculate the location of the corner of the rectangle, use VSHOW to determine the default reference point chosen by the ObjectFinder for the model, calculate the offset required to place the reference point at the desired location, and assign this offset to the model using mode 5 of the VTRAIN.FINDER instruction. The offset will be added to each recognized instance of the rectangle so that the reported position and orientation will be relative to the reference point and orientation determined by the programmer.

Because there is no way to specify the reference frame for a model before the model has been trained, the reference frame must be specified indirectly as offsets. This allows the programmer to specify the desired reference frame as offsets in position and orientation from the reference frame computed automatically by the ObjectFinder.

See the description of **VTRAIN.FINDER** on **page 122** for more details.

Details on Position and Orientation Limits

In some applications, the object will not translate or rotate significantly from the position or orientation in which it was trained. Effective with version 13.1, the ObjectFinder allows the application programmer to specify the range of positions and orientations in which the object will occur in an image. This constraint can be specified using mode 14 of the VTRAIN.FINDER instruction. For example, an object may occur with an orientation that is plus or minus 30 degrees from the orientation in which it was trained. If the programmer specifies these limits with

the VTRAIN.FINDER instruction, the ObjectFinder will not attempt to recognize objects with orientations outside this range. This can improve recognition time and reduce false matches. See the description of **VTRAIN.FINDER** on **page 122** for more details.

Details on Binary Images

The ObjectFinder was originally designed to work with gray value images, applying its own image processing algorithms to the image to compute features for training and recognition. In contrast, the prototype finder was designed to work with binary images and compute features from the boundary points of the binary regions and holes. There are situations where binary images are better for recognition than edges computed from gray-value images (e.g., binary images may be better when parts are very small). For that reason, the ObjectFinder in version 13.1 is enhanced to work with binary images using the same image processing algorithms that were used in the prototype finder. Since the ObjectFinder is easier to train, this enhancement should allow the ObjectFinder to replace the prototype finder in binary applications.

In binary-image processing mode, the ObjectFinder uses the edges of the boundary of the binary blobs extracted from the binary image specified in the AOI. The switches and parameters that define binary processing in the general vision system are also used by the ObjectFinder in binary mode, including V.BINARY, V.BACKLIGHT, and V.THRESHOLD. The following examples demonstrate the use of the switches and parameters for the binary-mode ObjectFinder. The program **bintrain( )**, shown below, can be used to train a model using binary edge processing.

```
.PROGRAM bintrain(cam, threshold, back, $modn)

        aoi = 2000

        cx = 307
        cy = 256
        width = 400
        height = 230

        VDEF.AOI aoi = 1, cx, cy, width, height, 0

        acq_ibr = aoi+11

        ; Set switches and parameters for binary image processing

        ENABLE V.BINARY[cam]
        PARAMETER V.THRESHOLD = threshold
        PARAMETER V.MAX.AREA = width * height

        ; Specify light (dark) object on dark (light) background
```

```
IF (back == 1) THEN
     ENABLE V.BACKLIGHT[cam]
ELSE
     DISABLE V.BACKLIGHT[cam]
END
VPICTURE (cam,, acq_ibr) 2

VWAIT

; Tell finder to use binary data as the input

VTRAIN.FINDER (cam, 15,, 0)

; Invoke finder with the usual instruction syntax

VTRAIN.FINDER (cam, 1, 2, 0, 75, 0) $modn, acq_ibr

; Reset the finder to default input mode (if desired)

VTRAIN.FINDER (cam, 15,, 1)
RETURN
```

The V.BINARY switch must be enabled so that the vision system performs binary image processing on the image acquired by the VPICTURE instruction. The VTRAIN.FINDER mode-15 instruction with argument 0 tells the finder to use the binary information. It is necessary to set the V.BINARY switch and tell the finder to use the binary information.

After training, the VPLAN.FINDER instruction must be used to generate a recognition plan before recognition can be done. The program **binfind( )**, shown below, can be used to recognize part instances using binary image processing.

```
.PROGRAM binfind(cam, threshold, back)

     AUTO times[2]

     aoi = 2000
     cx = 324
     cy = 244
     width = 612
     height = 440

     VDEF.AOI aoi = 1, cx, cy, width, height, 0
     acq_ibr = aoi+11

     ENABLE V.BINARY[cam]
     PARAMETER V.THRESHOLD = threshold
     PARAMETER V.MAX.AREA = width * height

     IF (back == 1) THEN
        ENABLE V.BACKLIGHT[cam]
     ELSE
```

```
    DISABLE V.BACKLIGHT[cam]
END

VPICTURE (cam,, acq_ibr) 2

VTRAIN.FINDER (cam, 15,, 0)

times[0] = 2
times[1] = 9000
times[2] = 9000

VFINDER (cam, 1, -1, -1, times[]) acq_ibr

VLOCATE (cam, 0, 0) $name, loc

WHILE VFEATURE(1) DO
    DECOMPOSE xform[] = loc
    x = xform[0]
    y = xform[1]
    ang = xform[5]

    TYPE "Found model ", $name, " at ", x, ", ", y, ", ", ang
    TYPE "Verification = ", VFEATURE(9), " percent"

    VLOCATE (cam, 0, 0) $name, loc
END
RETURN
```

Note the setup instructions used to specify certain options for binary processing such as:

- ENABLE V.BINARY
- PARAMETER V.THRESHOLD
- ENABLE V.BACKLIGHT

These instructions operate in the same manner with the ObjectFinder as with other binary image-processing applications, since the ObjectFinder uses the same algorithms. For example, the V.BACKLIGHT switch, when enabled, specifies that the scene contains dark objects on a light background. If disabled (the default), the scene contains light objects on a dark background.

See the description of **VTRAIN.FINDER** on **page 122** for more details.

Details on Edge Images

The ObjectFinder includes its own edge-detection algorithms for extracting features from grayscale images. These algorithms are more sophisticated than the edge-detection algorithms available to the application programmer and should be used in most cases. However, there are situations where the programmer may

need to use a custom edge-detection algorithm. Some images resemble edge images and may not require any edge processing. In version 13.1, the ObjectFinder has been enhanced to accept an edge image as its input and to use the edge image without further image processing.

In the edge mode, the ObjectFinder bypasses its own default edge-detection process and directly uses the edge image supplied in the instruction. All instructions, parameters, and switches that are relevant to an edge-detection instruction are also relevant to this mode.

The program **edgetrain( )**, shown below, can be used to train a model using an edge image computed using the VEDGE instruction.

```
.PROGRAM edgetrain(cam, etype, edgestr, $mod)
        AUTO $fmods[10]

        VDEF.AOI 2000 = 1, 308, 240, 500, 400, 0

        ; Specify the edge strength parameter

        PARAMETER V.EDGE.STRENGTH[cam] = edgestr

        ; Extract edges from the image (either Sobel or
        ; Cross-gradient)
        ; depending on the value of the supplied edge type argument

        VEDGE (cam, etype) 2000

        ; Specify edge image option for ObjectFinder

        VTRAIN.FINDER (cam, 15,, 2)

        ; Train model

        VTRAIN.FINDER (cam, 1, 2, 0, 75, 0) $mod, 2000

        ; Unset mode and restore default (gray image processing) mode

        VTRAIN.FINDER (cam, 15,, 1)
        RETURN
```

After training, the VPLAN.FINDER instruction must be used to generate a recognition plan before attempting recognition. The program **edgefind( )**, shown below, can be used to recognize part instances in an edge image supplied by the user.

```
.PROGRAM edgefind(cam, etype, strength)

       VDEF.AOI 2000 = 1, 308, 240, 500, 400, 0
```

```
; Set edge strength parameter.

PARAMETER V.EDGE.STRENGTH[cam] = strength

; Execute edge detection (either Sobel or Cross-gradient,
; depending on the value of the edge type argument)

VEDGE (cam, etype) 2000

; Set edge image processing option for ObjectFinder

VTRAIN.FINDER (cam, 15,, 2)

; Apply object recognition

VFINDER (cam, 1, -1, -1) 2000

VLOCATE (cam, 0, 0) $name, loc
WHILE VFEATURE(1) DO
    DECOMPOSE xform[] = loc
    x = xform[0]
    y = xform[1]
    ang = xform[5]

    TYPE "Found model ", $name, " at ", x, ", ", y, ", ", ang
    TYPE "Verification = ", VFEATURE(9), " percent"

    VLOCATE (cam, 0, 0) $name, loc
 END

; Unset edge image mode.

VTRAIN.FINDER (cam, 15,, 1)
RETURN
```

See the description of **VTRAIN.FINDER** on **page 122** for more details.

Details on Reading ObjectFinder Model Data

The VTRAIN.FINDER instruction is used to create and modify ObjectFinder models. However, there is also a need for a way to read the values that are stored in ObjectFinder models. The VSHOW.MODEL instruction has been used to display and receive information on correlation templates and OCR fonts. Starting with AdeptVision version 13.1, the instruction has been extended to return information about ObjectFinder models. When the model name string is the name of an ObjectFinder model, the VSHOW.MODEL instruction returns values from the ObjectFinder model in a data array. The values returned depend on the instruction mode, and the mode numbers for returning ObjectFinder data have been selected to correspond to the VTRAIN.FINDER modes that modify the corresponding data. Essentially, VSHOW.MODEL reads back whatever

information is written into the model by the corresponding mode of VTRAIN.FINDER. Further, the data is read back using the same units that were used when it was written. See the description of **VSHOW.MODEL** on **page 117** for more details.

In previous versions of AdeptVision, the only way to retrieve ObjectFinder model data was by using a VSHOW followed by use of the VFEATURE function. However, the VFEATURE array is a global data structure and there can be conflicts between uses of this global array. The VSHOW.MODEL instruction copies a selected portion of the ObjectFinder model data into a user-specified array that can be local to the V+ program.

For correlation templates and OCR fonts, the VSHOW.MODEL instruction is used to display the template or font. Since ObjectFinder models can be displayed using the VSHOW instruction, the VSHOW.MODEL instruction has not been enhanced to display ObjectFinder models.

Changes to Vision Keywords

This section presents detailed descriptions of changed V+ system keywords related to vision. These keywords are available only in systems with the AdeptVision VXL option.

The global system switch VISION enables and disables all vision processing. If this switch is DISABLEd, the system will behave as if the vision option is not installed. The VISION switch default value is ENABLEd.

The vision system includes the following types of keywords:

- Program instructions
- Functions
- Monitor commands
- System parameters
- System switches

The keywords are presented in alphabetical order, with the description for each keyword starting on a new page. Note that some keywords may belong to more than one keyword type. The keyword is shown at the top of each page. The sections described on pages **32** and **33** are included for each keyword, as appropriate.

Syntax

```
VSHOW.MODEL (mode) $chars, data[i] = $model_num

VSHOW.MODEL (mode) , data[i] = $model
```

Function

Display a model—either a correlation template or an Optical Character Recognition (OCR) font—and return information about it, or return information about all the defined templates or OCR fonts. Also, accept the name of an ObjectFinder model and return data from the model in the array provided as an argument.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

`mode` Optional. When the input argument is the name of a correlation template or OCR font, mode is interpreted as a bit field to control the behavior of this instruction. Possible values are 0 or 1, below. When the input argument is the name of an ObjectFinder model, mode is interpreted as an integer that specifies the finder model data that is returned in the data array.

- 0 Default value; the font(s) or template(s) is (are) displayed.
- 1 Nothing is displayed.
- 3 The instruction returns the model symmetry, add instance flag, and reject verify percent in the first three elements of the data array. See the Details section for more information.
- 5 The instruction returns the x, y, and z offsets and the rotation offsets about the x, y, and z axes. In current vision systems, only the x and y offset and the rotational offset about the z axis are meaningful. See the Details section for more information.
- 14 The instruction returns the position and orientation constraints. See the Details section for more information.

`$chars` Optional string variable to be assigned the set of characters in the font specified by the parameter **`$model_num`**. (This parameter is ignored if **`$model_num`** refers to all fonts or to a correlation template.)

`data[]` Optional array into which font or template information is to be placed. The amount of information depends on whether the model specified by **`$model_num`** (see below) is a font or a correlation

template. The information returned further depends on whether all models (that is, all fonts or all templates) or a single model is specified.

If a specific OCR font is referenced:

data[i+0] = Number of characters in the font
data[i+1] = Height of the font in pixels
data[i+2] = Font color: TRUE = black, FALSE = white
data[i+2+1] = Number of trained instances of the first character in `$chars`
...
data[i+2+n] = Number of trained instances of the nth character in `$chars`

If a specific correlation template is referenced:

data[i+0] = Width of the template in pixels
data[i+1] = Height of the template in pixels

If all fonts or templates are referenced:

data[i+0] = Number of fonts or templates defined
data[i+1] = Number of first font or template defined
...
data[i+n] = Number of nth font or template defined

`i` Optional array index that identifies the first element to be defined in `data[]`. The default is 0. If a multiple-dimension array is specified, only the rightmost index is incremented as the values are assigned.

`$model_num` String variable specifying a font, all fonts, a template, or all templates for display (if `mode` = 0) and return of information.

The string naming a font has the form "FONT_n", where "n" is the number (in the range 1 to 99) of a single font; "FONT_0" specifies all fonts.

Similarly, the name of a template has the form "TMPL_n", where "n" is the number (in the range 1 to 99) of a single template; "TMPL_0" specifies all templates.

`$model` The name of an ObjectFinder model that was trained using the VTRAIN.FINDER instruction.

Details

Getting Data on Correlation Templates

The VSHOW.MODEL instruction may be used to display and make queries about one or all correlation templates in vision memory. The vision system displays a specific template by copying the template from system memory into the frame store currently displayed. The template is centered in the frame store. If all templates are specified for display, a list of the identifying numbers of all known templates is displayed.

Getting Data on OCR Fonts

The VSHOW.MODEL instruction may also be used to display and make queries about one or all OCR fonts in vision memory. Fonts are displayed in the Vision display window. If all fonts are specified, a list of the identifying numbers of all known fonts is displayed. If a specific font is specified, the characters in the font are displayed along with the following information:

- The approximate height of the font as specified in the VDEF.FONT instruction
- The color of the font characters: black on white or white on black
- The base orientation
  This angle is set when the font is first trained (see the instruction VTRAIN.MODEL). When the font is subsequently trained or recognition/verification is performed (see VOCR), the features in the OCR window are rotated as necessary to match the base orientation.
- For each character in the font, the vision system displays the number of times that the character has been trained

The above information, except for the base orientation, is also returned in the `data[]` parameter of this instruction.

Getting Data on ObjectFinder Models

The VSHOW.MODEL instruction has been enhanced to accept the name of an ObjectFinder model and return data from the model in the array provided as an argument:

```
VSHOW.MODEL (mode) , data[] = $model
```

where the input argument $model is the name of an ObjectFinder model trained using the VTRAIN.FINDER instruction. The comma before the data array is required. In previous versions of the VSHOW.MODEL instruction the input argument was the name of a correlation template or OCR font and the mode was interpreted as a bit mask with the low bit indicating whether the template or font

should be displayed. When the input argument is the name of an ObjectFinder model, the mode is interpreted as an integer that specifies the finder model data that is returned in the data array. The same mode numbers are used for the VSHOW.MODEL instruction as those used in the VTRAIN.FINDER instruction for assigning values to the ObjectFinder model. So, for example, VTRAIN.FINDER (mode = 5) *assigns* the model offsets, while VSHOW.MODEL (mode = 5) *reads* the model offsets. Essentially, VSHOW.MODEL reads back whatever information is written into the model by the corresponding mode of VTRAIN.FINDER. Further, the data is read back using the same units that were used when it was written. See the description of the **VTRAIN.FINDER** program instruction on **page 122** for more details on the meaning of the ObjectFinder model parameters.

The VSHOW.MODEL modes associated with ObjectFinder data are:

Mode 3 — The instruction returns the model symmetry, adds instance flag, and reject, verify percent in the first three elements of the data array.

Mode 5 — The instruction returns the x, y, and z offsets and the rotation offsets about the x, y, and z axes. In current vision systems, only the x and y offset and the rotational offset about the z axis are meaningful.

Mode 14 — The instruction returns the position and orientation constraints. The first element in the data array is a bit mask that indicates which constraints have been specified. The six bits in right to left order, starting with the least significant bit, indicate, respectively, whether the x translation, y translation, z translation, x rotation, y rotation, and z rotation constraints have been set. The next six elements in the data array are the upper limits on x translation, y translation, z translation, x rotation, y rotation, and z rotation, respectively, relative to the nominal value in the model. The last six elements in the data array are the lower limits on x translation, y translation, z translation, x rotation, y rotation, and z rotation, respectively, relative to the nominal value in the model. The lower limits are always returned as positive numbers, since it is implicit that the range of a coordinate is the nominal value plus the upper limit down to the nominal value minus the lower limit. If only one side of the limit was provided, then the lower limit will be equal to the upper limit. If a particular constraint has not been set, then the values returned by mode 14 of VSHOW.MODEL are indeterminate.

Examples

Display a list of all defined templates:

```
VSHOW.MODEL () = "TMPL_0"
```

Get all the information about font "font.num", but do not display it:

```
VSHOW.MODEL (1) $str, data[] = "FONT_"+$ENCODE(/I0,font.num)
```

Related Keywords

VCORRELATE	(program instruction)
VDEF.FONT	(program instruction)
VOCR	(program instruction)
VTRAIN.MODEL	(program instruction)

Syntax

```
VTRAIN.FINDER (cam, mode, dmode, arg, arg2, arg3)
                    $model_name, ibr = value, value2, value3
```

Function

Initiate training of the specified ObjectFinder model.

Usage Considerations

This instruction requires that all virtual cameras be idle (that is, no VPICTURE or VWINDOW operation can be executing).

Training is sensitive to camera calibration. Before training begins, mount, focus, and securely fix your cameras and lenses. Then calibrate the camera(s) that will be used to train and recognize the prototypes. The same camera(s) with the same calibration and position used for training must also be used for recognition.

Adept recommends the following settings for the symmetry override parameter (mode 1, arg3):

- For most cases, allow model disambiguation for all models by leaving the symmetry override parameter at its default value (0).
- If a model has perfect n-way symmetry and the ObjectFinder is being programmed in V+, set the symmetry override parameter to (1) and adjust the orientation returned by the ObjectFinder using V+ code so that gripper motion is optimized.
- If the object is perfectly circularly symmetrical, then set the symmetry override parameter to –1 to disable disambiguation.

Parameters

`cam` Optional integer specifying the virtual camera (and associated calibration, switches, and parameters) to use when acquiring an image for defining a prototype. This camera will also be the default camera associated with the prototype (see the *AdeptVision User's Guide* for details on associating cameras with a prototype).

`mode` Optional integer specifying the operating mode for this instruction. See below for details.

`dmode` Optional real-valued expression that specifies the display mode to use when displaying the border of the window:
–1 No draw
0 Erase

1 Default. Draw solid
2 Complement
3 Draw dashed
4 Complement dashed

`arg, arg2, arg3` Integers whose meaning depends on the value of the mode parameter. See below for details.

`$model_name` String containing the name for the model (up to 15 characters).

`ibr` Integer value specifying the image buffer region for training a model. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The image buffer region's AOI must specify a nonrotated rectangular shape. For ObjectFinder, the AOI width is reduced, if necessary, to a multiple of four pixels.

`value` Floating-point expressions whose meaning depends on the value of the mode parameter. See below for details.

Description of the mode parameter:

This section describes the meanings of the different values used with the mode parameter, and the meanings of other input parameters used with a particular mode. If an input parameter is not shown, it cannot be used for that mode.

Mode 1: Train first instance (new model)
- arg Hierarchical level (0 - 2) is the result of the subsampled image
  - 0 Full resolution
  - 1 One-half resolution
  - 2 One-quarter resolution
- arg2 Verify percent
- arg3 Symmetry override
  - 0 Enable model disambiguation (default)
  - 1 Disable model disambiguation
  - n>1 Disable model disambiguation and specify that the model has n-way symmetry. The angle of a model instance is reported by the ObjectFinder modulo 360/n degrees.
  - –1 Disable disambiguation and specify that the model has circular symmetry. The angle of a model instance reported by the ObjectFinder is always zero degrees.

Mode 2: Display features only
- arg Hierarchical level (0 - 2) is the result of the subsampled image
  - 0 Full resolution
  - 1 One-half resolution
  - 2 One-quarter resolution
- arg2 Edges only (0,1)
  - 0 Features are fit to the edges and the features are displayed
  - 1 Only the results of edge detection are displayed

Mode 3: Set ObjectFinder model parameters (after last instance)
To train new instances during multi-instance training, use VFINDER with mode=2.
arg Indicates the value:
1 Symmetry of model
2 Auto-add instances is okay (value=0 turns it OFF; value=non-0 turns it ON)
3 Verify percent to assign to model
value The value to assign to the designated parameter

NOTE: VisionWare uses the Mode 3/arg=2/value flag to control when to stop adding instances when in learning mode. It checks the convergence count in the model (using VSHOW/VFEATURE) and stops adding instances after the features in the model have converged sufficiently. See the ***AdeptVision Reference Guide*** for details on VSHOW/VFEATURE.

Mode 4: Feature weight setting (also feature deletion)
arg Feature number (1 to num_features)
(–1 means all features)
value New weight to assign
0 deletes the feature (deletion means that the feature is not used)
1 - 100 relative weighting for verification (the feature is still used to form feature-pairs).

Mode 5: Object instance offsets
value x coordinate of offset in pixels
value2 y coordinate of offset in pixels
value3 angular offset in degrees

Mode 6: Display a model feature
arg Number of the feature to display

Mode 7: Display a model feature in RED
arg Number of the feature to display

Mode 8: Accept the last-trained additional instance
arg Low weight cutoff (used for multi-instance training and is the threshold on feature weights [on a scale of 0 - 100]; feature weights below the low weight cutoff are set to zero)

Mode 9: Do not accept the additional instance last trained
arg Low weight cutoff

Mode 10: Clear all multi-instance training statistics. Leave the first instance.
arg Low weight cutoff

Mode 11: Apply low-weight filtering
arg Low weight cutoff

Mode 12: Undo the last accepted instance
arg Low weight cutoff

Mode 14: Position and orientation limits
- arg Coordinate being limited
  - 0 Remove all limits
  - 1 Limits specified for X (pixels)
  - 2 Limits specified for Y (pixels)
  - 6 Limits specified for rotation (degrees)
- value If value2 is not provided, then the limit is +value1 and –value1, otherwise the limit is +value1 and –value2.

Mode 15: ObjectFinder image type
- arg Indicates the type
  - 0 Run-length binary
  - 1 Gray-value image (default)
  - 2 Edge image

Details

When normal (initial) training is performed, a new model is created and assigned the given input values. Additionally, the ibr is processed to create edge pixels, and these are further processed to extract lines and arcs. The lines and arcs are paired up in various ways to form intrinsic "feature pairs". These can be later used to match with similar pairs formed from recognition images to make proposals of instances of the model. These lines, arcs, and pairs define the geometric portion of the model.

There are three system parameters that control the processing.

1. The system parameter V.EDGE.STRENGTH[vc] controls the threshold for edge points extracted during preprocessing.
2. The system parameter V.MAX.PIXEL.VAR[vc] controls the primary fitting of lines and arcs. A secondary operation performs a least-squares fitting to the edge points, which greatly improves the accuracy.
3. The system parameter V.MIN.LEN[vc] restricts the pairs by setting a minimum length for the features allowed to be used to make pairs.

The system switch V.FIT.ARCS[vc] should normally be enabled, but can be disabled to some advantage in certain difficult cases. If an object is mostly lines but contains some segments that occasionally fit as arcs due to noise in the image, you can force the system to fit only lines and avoid the additional computation time required to fit arcs.

Input Parameters Using Virtual Cameras

All switches and parameters should be at the default settings, except as noted here.

The following settings are required:

```
PARAMETER V.MIN.AREA[vc] = 4

PARAMETER V.MIN.HOLE.AREA[vc] = 4
```

The following settings are suggested:

`ENABLE V.FIT.ARCS[vc]`	(May be disabled if the image consists of mostly lines. See the description above for details.)
`V.MIN.LEN[vc] = 10`	(See the description above for details.)
`V.MAX.PIXEL.VAR[vc] = 2.5`	(See the description above for details.)
`V.EDGE.STRENGTH[vc] = 9`	(See the description above for details.)

Object Instance Offsets

Mode 5 of the VTRAIN.FINDER instruction can be used to specify position and orientation offsets:

```
VTRAIN.FINDER (cam, 5) $model_name = x, y, angle
```

The x and y coordinates of the position offset are specified in units of pixels. Since the x and y coordinates are specified as floating-point values, the offsets can be given with subpixel resolution. Currently, the offsets are stored in the finder model with an accuracy of 1/4 pixel. The angle offset is specified in degrees. The position and orientation offsets are added to the position and orientation of a recognized model in the image. The position offset is a vector from the model reference point chosen by the ObjectFinder at the time that the model is trained. The position offset is applied in the coordinate system of the model. After the position offset is applied, the orientation offset is added to the orientation of the model instance to rotate the reference frame by the specified amount. Any component of the position offset that is omitted defaults to zero. Likewise, if the orientation offset is omitted, it defaults to zero. The position and orientation offsets can be changed, at any time after the model has been changed, using VTRAIN.FINDER mode 5. The model does not have to be retrained for the new offsets to be used. The position and orientation offsets are saved when the model is saved using the VSTORE instruction.

ObjectFinder Image Type

Mode 15 of VTRAIN.FINDER is used to set the source type used for training and recognition. Versions of the ObjectFinder prior to 12.4 accepted a gray-value image and performed the appropriate edge-detection operations automatically before training or recognition. The ObjectFinder can accept a run-length binary image or an edge image as the source for training and recognition. The syntax of the instruction for setting the source is

```
VTRAIN.FINDER (cam, 15,, type)
```

where the type parameter has the following values (similar to the VRULER and VFIND.LINE instructions) that specify the type of source image for processing:

Type	Source
0	Run-length binary
1	Gray-scale image (default)
2	Edge image

For type equal 0, training and recognition are performed with features (line segments and circular arcs) computed from the boundaries of regions in the run-length binary image using the image processing algorithms that are used in the prototype finder. For type equal 1, training and recognition are performed with features that are computed from the gray-scale image using image processing algorithms that are built into the ObjectFinder. For type equal 2, training and recognition are performed with features that are computed from an edge image that is provided by the user.

> **NOTE:** The edge image should contain chains of edges that are one pixel wide.

The ObjectFinder normally subsamples the input image before extracting features (usually 1 level of subsampling). When the ObjectFinder input is a binary image or an edge image, training and recognition must be performed at level 0, unless the program itself has subsampled the input image.

Models trained with different modes (for example, gray scale or binary) or different backlight settings in binary mode (backlight on or off) should not be used in the same planning or recognition instruction. During recognition, the planned models should have the same mode as the one set for the ObjectFinder. If there is any difference in the mode or backlight settings during either planning or recognition, the error message *Inconsistent model mode setting* (error code –763) is returned. If this error code is returned for the recognition instruction (VFINDER), the mode for the ObjectFinder should be changed to the one set for the planned models. If this error code is returned for the VPLAN.FINDER instruction, then the list of models should be changed to include only those models that have the same mode (and backlight setting, if applicable). If a model has been trained with the wrong mode, then the model must be deleted and retrained with the correct mode.

Position and Orientation Limits

Mode 14 of the VTRAIN.FINDER instruction can be used to specify position and orientation limits for the model during recognition.

The instruction syntax is

```
VTRAIN.FINDER (, 14,, coordinate) $model = value1, value2
```

The coordinate parameter specifies the coordinate that will be constrained, as shown below. The translational limits are in units of (uncalibrated) pixels; the orientation limits are in units of degrees.

Coordinate	Meaning
0	Remove all limits
1	Specifies the limits in X
2	Specifies the limits in Y
6	Specifies the limits on orientation

Specifying value1 = 0 means no variation will be allowed in that coordinate. For example,

```
VTRAIN.FINDER (, 14,, 6) $model = 0
```

means no rotation of the model (from the orientation at which it was trained) will be allowed when making proposals.

If `value2` is not provided, then the range of allowed coordinates is plus or minus `value1` relative to the nominal value of the coordinate in the model. Otherwise, the upper limit of the range is nominal plus `value1`, and the lower limit of the range is nominal *minus* `value2`. When specifying both the upper and lower limits of the range of a coordinate, `value1` and `value2` are both entered as *positive* numbers. This scheme is used to ensure that the nominal value of a coordinate in the trained model is always in the interval of allowable values for the coordinate.

4 Adept Utility Programs

Changes to the Utility Programs 130

CONFIG_C 134

- Menu Structure 137
- Detailed Descriptions of Program Operations 141
- Setting the IP Address of the Adept Controller 155
- Software Configurable Switch Settings 156
  - User Interface 156
  - AdeptWindows PC Setup 157
  - V+ Auto-Start 158
  - Boot Sequence 158
  - Auto-Boot 158
  - DDT Serial I/O on Auxiliary CPU 159
- DeviceNet Configuration Setup 159
  - Scanning the DeviceNet for Setup Information . . 159
  - DeviceNet Statements in V+ Configuration Data . . 162

SPEC 168

- Menu Structure 171
- Descriptions of Selected Program Operations 181
- Motion Acceleration and Deceleration Percentages . . 187
- Options for the Frequency Response Tests 187
- Adding a Description to a SPEC-Generated Data File . . 189
- Static Friction Feedforward Compensation 190
- Navigation Hotkeys 192
- Plot Color Preferences 192
- Programmable ESTOP Delay-time Parameters 195
- Custom Point-to-Point Motion Routine 196

Changes to the Utility Programs

Table 4-1 summarizes the changes that have been incorporated on the Utility Disk since the release of version 13.0.

Table 4-1. Changed Utility Disk Files

Utility Disk			Program Version	Description of the Changes
Ver.	P/N Rev	File Name		
13.1	A	ADVSERVO	13.1A	Updated for compatibility with V+ 13.1 Added support for Static Friction Feedforward opcodes Added new subroutines (See Chapter 5 for more information about these changes)
		BELT_CAL	11.3B	Added support of the B floppy drive
		CONFIG_C	13.1C	Added support of the B floppy drive Added new configuration statements in the SYSTEM section: PASSIVE_ESTOP, STARTUP_MESSAGE The NETWORK statement in the SYSTEM section is now ignored when the configuration data is for V+ version 13.1 Increased the default setting of the /TOTAL_KB parameter in the VISION statements After a new configuration statement is added, the user is now always given the option of making changes to it Added recognition of the Adept Digital Workcell option bit in the controller NVRAM
		DISKCOPY	11.4D	List the B floppy drive only if it is actually available
		PROFILER	12.5A	Added support of the new vision analysis task

Table 4-1. Changed Utility Disk Files (Continued)

Utility Disk			**Program Version**	**Description of the Changes**
Ver.	**P/N Rev**	**File Name**		
		SECURE	10.5B	Added support of the B floppy drive
13.1	A	SPEC	13.1A	Added support to frequency response tests for: • Testing multiple frequency bands • Specifying start and stop amplitude • Linear spacing of input amplitudes • Display of amplitudes used • ASCII file export of test data (with option for including a document description) Added the following: • STOP-test option to the Robot/Motor Servo Tuning menu • Support for Static Friction Feedforward Compensation • Allow a document description in SPEC-generated files • Support for new menu “hotkey” feature • Enhanced plotting: overlays in different colors and support for color preferences • Wider search for SPECSUB.LIB • Search for SPEC preferences file during initialization • Enhanced Auto Backup features—now prompts before overwriting an existing backup file • Semi-automated calibration setup of the Gen-II Z-theta unit • Support for three programmable ESTOP delay timeout parameters added to the Robot Initialization Specifications menu

Table 4-1. Changed Utility Disk Files (Continued)

Utility Disk			Program Version	Description of the Changes
Ver.	**P/N Rev**	**File Name**		
13.1	A	SPEC	13.1A	Other changes: • Removed SPEC warning messages that appear when preferences file cannot be found or is incompatible with current version • Miscellaneous bug fixes
		TOUCHCAL	10.4B	Added support of the B floppy drive

For the version 13.1 Utility Disk, each of the following utility programs has been changed in some way that affects its description in the ***Instructions for Adept Utility Programs***.

- **CONFIG_C**
- **SPEC**

Updated descriptions of those utility programs are included on the following pages. The utilities are presented in alphabetical order, with the description for each starting on a new page. The sections described in the table below are included for each utility, as appropriate.

Item	Description
Description	A short description of the utility
Disk File Name	The name of the disk file that contains the program
Program Name	Name of the program that is EXECUTEd to begin the utility
Functions	Major functions of the utility
Abstract	Overview of the utility
Requirements	The equipment and conditions required to use the utility
Procedure	How to load and execute the utility

Item	**Description**
Program Prompts	Descriptions of the utility menu items and the prompts displayed during use of the program
Programming Considerations	How to include the utility in a V+ program
Special Considerations	Additional information

CONFIG_C

Description

Controller configuration utility

Disk File Name

CONFIG_C.V2

Program Name

a.config_c

Functions

- Install a new V+ system
- Display and edit system configuration
- Display and modify device modules for robots
- Display and initialize the NVRAM contents
- Display information about the devices connected to the DeviceNet

Abstract

The Installation option installs a new V+ system, configuring it to match the system being replaced.

The V+ system includes configuration data stored in a special file on the system disk. The CONFIG_C utility lets you edit this file. Each time the controller is restarted, the system reads the statements in the configuration file and sets the appropriate defaults.

> **NOTE:** Changes to the system configuration do not take effect until the V+ system is loaded from the modified system file.

The Robots and Device Modules options allow you to change the kinematic device-control modules included in the V+ system and the robots that are controlled by the system.

The NVRAM section displays and initializes the NVRAM, which is read each time the controller is powered up.

Requirements

The following items are needed to run CONFIG_C. All of these (except as indicated) were supplied with your Adept system:

- Adept controller, using V+ version 13.0
- Adept Utility Disk containing the files CONFIG_C.V2, DISKCOPY.V2, and SPECSUB.LIB (or copies of the files on the Compact Flash).

Before you can run CONFIG_C:

- The V+ system must be loaded.
- If you are using a robot system, the robot must be calibrated and COMP mode must be selected on the manual control pendant.
  or
  The DRY.RUN system switch must be enabled.
  or
  The program can be executed in a program task other than #0.

Procedure

1. Save to disk any program changes you have made.
2. Clear system memory by typing:

   ```
   zero
   ```

3. Load the utility program:[1]

   ```
   load \util\config_c
   ```

4. Execute the program:

   ```
   execute a.config_c
   ```

   (Or use `execute 1 a.config_c` to execute the program as task #1.)

5. Select the option you want from the menu.
6. Follow the prompts provided by the program.

   NOTE: If a floppy diskette is modified, the small write-protect slider on the back of the diskette must be in the unprotected position. That is, it must be in the position that closes the hole through the diskette jacket. Make sure you move the slider back to the protected position after the operation is completed.

[1] Use the correct drive and path specification if the file is stored on another drive or subdirectory.

Do not alter the original V+ system diskette supplied by Adept. Instead, make a copy of the distribution diskette (with this program during the process of installation, or with the DISKCOPY utility), and then modify the configuration information on the copy.

7. Delete the configuration program from the system memory by typing (use `kill 1` in place of `kill` if the program was executed as task #1):

```
kill
delete a.config_c
```

Special Considerations

Changes to the system configuration do not take effect until the V+ system is loaded from the modified system file. Thus, after making all the desired changes to the configuration, turn off power to the controller and reboot the system from the modified system file.

Adept recommends making a backup copy of the modified V+ system. The DISKCOPY utility can be used for that purpose.

The V+ configuration file may contain statements for hardware or options that are not installed in the system. For example, there may be statements for four analog input boards, but only two boards are installed. The extra statements are ignored and can be left in the configuration file.

Not all items in the configuration file can be changed. Items that cannot be changed by the configuration utility are noted as such when they are displayed.

The configuration program defines many real-valued and string variables with cf., dc., it., iu., ld., and sp. as prefixes. These variables must not be modified by the user or by other programs. (All the variables are deleted from memory when the program exits normally. All the variables are deleted when a DELETE command is used to delete the program from memory, as described earlier.)

If DISKCOPY utility is not already in memory when the `V+ installation` menu item is first selected, the CONFIG_C program automatically loads DISKCOPY into memory. If you later attempt to load DISKCOPY while CONFIG_C (and DISKCOPY) is still in memory, you will see many `*Program already exists*` errors, which can be ignored.

The CONFIG_C.V2 file is a protected binary file. Thus, the programs in the file cannot be displayed, edited, or stored from memory to disk. The file can be copied with the FCOPY monitor command or the disk-copy utility program (DISKCOPY).

Menu Structure

```
EXIT to system monitor

V+ Installation

   Return to MAIN MENU
   SAVE current ROBOT DATA
   INSTALL a NEW V+ SYSTEM
   APPLY saved ROBOT DATA

V+ System Configuration Data

   Return to MAIN MENU
   DISPLAY system CONFIGURATION
   EDIT system CONFIGURATION
     Done editing
     Change HEADER configuration
     Change VPLUS configuration
     Change ANALOG_INPUT configuration
     Change ANALOG_OUTPUT configuration
     Change DIGITAL_INPUT configuration
     Change DEVICENET configuration
     Change NETWORK configuration
     Change ROBOT configuration
     Change SERIAL configuration
     Change SERVO_BOARDS configuration
     Change SYSTEM configuration
     Change TASKS configuration
     Change VISION configuration
   EXPORT configuration DATA

Robots and Device Modules

   Return to MAIN MENU
   LIST current CONFIGURATION of system file
   LIST device modules in a device-module FILE
   REPLACE device modules in system file
   APPEND device modules to system file
   SELECT device modules for robots

Controller NVRAM
```

```
   Return to MAIN MENU
   DISPLAY NVRAM contents
   INITIALIZE the NVRAM
   SET IP ADDRESS in the NVRAM
   Set “SWITCHES” in the NVRAM

Scan the DeviceNet

   Return to main menu
   SCAN the DeviceNet for nodes
```

Program Prompts

The configuration utility program displays the following main menu:

```
0 => EXIT to system monitor
1 => V+ Installation
2 => V+ System Configuration Data
3 => Robots and Device Modules
4 => Controller NVRAM
5 => Scan the DeviceNet
```

> **NOTE:** You can exit the program at any prompt by entering CTRL+Z.

These menu items lead to the following submenus:

```
0 => EXIT to system monitor
```

Exit the configuration program and return to the V+ system monitor.

```
1 => V+ Installation
```

When this menu item is selected, the CONFIG_C program checks for the DISKCOPY utility and SPECSUB library to be in memory. If they are not in memory, the program attempts to load the respective files DISKCOPY.V2 and SPECSUB.LIB from the same path from which CONFIG_C was originally loaded. If the files cannot be found, the user is asked to enter the appropriate device/unit/path specification to access the files. This menu item cannot be used until both files have been loaded.

The following menu is displayed at this level:

```
0 => Return to MAIN MENU
1 => SAVE current ROBOT DATA
2 => INSTALL a NEW V+ SYSTEM
3 => APPLY saved ROBOT DATA
```

Items #1 and #3 are flagged with an asterisk (*) if the program cannot be used to save/apply data files from/to the current V+ system. In this case a different version of the SPECSUB library (or the SPEC program) must be used to perform these operations (however, item #2 can still be used).

These menu items perform the following operations, respectively:

1. Save the robot and encoder configuration data for the current V+ system to files, for use by menu item #3 after installing a new V+ system.
2. Copy a new V+ system to the system disk, install the appropriate device modules, and copy the previous system configuration data to the new system.
3. Apply the saved robot and encoder configuration data to the current V+ system disk.

Refer to **page 141** for a detailed description of these processes.

```
2 => V+ System Configuration Data
```

This menu item displays the following menu:

```
0 => Return to MAIN MENU
1 => DISPLAY system CONFIGURATION
2 => EDIT system CONFIGURATION
3 => EXPORT configuration DATA
```

The description of these menu items follows:

```
1 => DISPLAY system CONFIGURATION
```

This menu item displays all the statements in the configuration file that is on the specified system disk.

```
2 => EDIT system CONFIGURATION
```

This menu item allows you to edit the individual statements in the configuration file. See **page 146** for a detailed description of this process.

```
3 => EXPORT configuration DATA
```

This menu item allows you to copy the system configuration data to a regular disk file or network file.

```
3 => Robots and Device Modules
```

The following submenu is displayed:

```
0 => Return to MAIN MENU
1 => LIST current CONFIGURATION of system file
2 => LIST device modules in a device-module FILE
3 => REPLACE device modules in system file
4 => APPEND device modules to system file
5 => SELECT device modules for robots
```

For details on the features of this submenu, refer to the ***AdeptMotion VME Developer's Guide.***

```
4 => Controller NVRAM
```

This menu item displays the following submenu:

```
0 => Return to MAIN MENU
1 => DISPLAY NVRAM contents
2 => INITIALIZE the NVRAM
3 => SET IP ADDRESS in the NVRAM
4 => Set "SWITCHES" in the NVRAM

The description of these menu items follows:

1 => DISPLAY NVRAM contents
```

The controller model, serial number, network IP address, and list of installed options are stored in a special section of nonvolatile RAM (NVRAM). This option displays the contents of the NVRAM.

```
2 => INITIALIZE the NVRAM
```

This option initializes the NVRAM on the AWC processor. This needs to be done only when a replacement NVRAM has been installed on the processor. (The program will not permit an initialized NVRAM to be modified.)



CAUTION: When initializing the NVRAM, you are asked to enter the serial number of the controller. The serial number cannot be changed after it is written to the NVRAM. If the wrong serial number is stored, none of the license passwords for the controller will be accepted.

```
3 => Set IP address in the NVRAM
```

This option adds or changes the network IP address and IP subnet mask for the controller. See the ***AdeptNet User's Guide*** for information about IP addresses and subnet masks.

At each prompt, you can either press ENTER to accept the current setting or enter the desired new address or subnet mask with periods between the components. Changes to the settings will have no effect if they are not written to the NVRAM.

The first value of an IP address cannot be zero. The second through fourth elements of an IP address can be set to zero. Also, all four elements of the subnet mask can be set to zero.

```
4 => Set “SWITCHES” in the NVRAM
```

Refer to **page 156** for a detailed description of this menu item.

```
5=> Scan the DeviceNet
```

This menu item is used to gather information about the nodes that are present on the DeviceNet that is connected to the Adept controller. The information is used when editing the statements in the DeviceNet section of the V+ system configuration data.

> **NOTE:** No other V+ execution tasks should be running while the DeviceNet is being scanned.

Detailed Descriptions of Program Operations

```
1 => V+ Installation
```

This description of the installation process is divided into two parts: an overview section followed by a detailed description.

Overview

The options under this menu item are used when upgrading a V+ system. The purpose of these operations is to carry forward all the custom configuration information from the old system to the new system.

There are three main steps for upgrading to a new V+ system:

1. Save the current robot and encoder configuration data in disk files.

   This step is not needed if the old V+ system is not configured to control any robot and does not access external (belt) encoders.

   If the old V+ system does control one or more robots, or if it accesses external (belt) encoders, perform steps a and b below.

   a. Boot the *old* V+ system.

b. Create a binary configuration data file for each robot and for the external encoders. (These files can be stored on any disk, including a V+ system disk.)

   The CONFIG_C and SPECSUB version used for this step must be compatible with the old V+ system. You will need the CONFIG_C version for the old V+ system as well as for the new V+ system.

2. Copy the new V+ system to the desired output disk drive, install the required device modules, and copy the system configuration data from the old V+ system to the new V+ system.

   Perform this step with the Adept MV controller booted from the *new* V+ system disk and use the edition of CONFIG_C received with the new system. This step is described in detail below.

3. Apply the saved robot and encoder configuration data (see Step #1 above) to the new V+ system disk.

   This step is not needed if the old V+ system is not configured to control a robot and it does not access external (belt) encoders.

   If the old V+ system does control one or more robots, or if it accesses external encoders, perform steps a and b below.

   a. Boot the *new* V+ system (*after* Step #2 is done).

   b. Load the contents of the robot and encoder configuration data files into memory, and write the data to the new V+ system disk using the new edition of the CONFIG_C utility.

The following paragraphs describe these steps in greater detail.

```
*** INSTALLATION OF V+ SYSTEM ***

1 => SAVE current ROBOT DATA
```

> **NOTE:** This menu item followed by an asterisk(*) indicates that the operation cannot be performed because of an incompatibility between the edition of CONFIG_C being used and the current V+ system. Use the edition of CONFIG_C that was supplied with your old V+ system.

This procedure creates configuration data files for the robots and external encoders that the current V+ system is configured to control. These data files can be stored on any disk, and can be used later to restore robot and encoder configuration data to the V+ system.

After the purpose of the menu item is displayed, you can cancel the menu selection.

You are asked which disk drive should receive the data files. If an optional floppy drive is indicated, you are asked to insert a diskette. All the data files are created automatically. Each file name is displayed for reference. For example, if an error occurs, the file name should help you to understand the problem. If the files were written to a diskette, you are advised to write-protect and label the diskette.



CAUTION: Any existing files with the same names are overwritten without warning!

```
2 => INSTALL a NEW V+ SYSTEM
```

This option copies the new V+ system to the desired output disk drive and then duplicates the system configuration from the old V+ system on the new V+ system disk.

CONFIG_C automates most of the installation process. As noted above, however, the robot and encoder configuration data must have been saved in disk files before the installation process is initiated. After completing this installation step, you must complete the installation process by selecting step #3 (`APPLY saved ROBOT DATA`).

You need the following items to install a new V+ system:

1. An old V+ system (on a diskette, hard drive, or Compact Flash).

   This is the V+ system that you have been using. The configuration of this system is duplicated in the new V+ system.

2. Configuration data files for all the devices controlled by the old V+ system.

   If the old V+ system includes customized data for the robots or external encoders, you must have an up-to-date (binary) configuration data file for each robot and the encoders.

   NOTE: In general, those data files can be created only with the edition of CONFIG_C or SPEC that you received with the old V+ system.

3. A new V+ system (on a diskette, hard drive, or Compact Flash).

   This is the V+ system that you want to install. This installation process copies this system to a different disk (if necessary) and configures the system to match the existing (old) V+ system.

> **NOTE:** If your V+ system includes support for a non-Adept robot, the installation process will need to access the appropriate device-module file for the new V+ system. That file is distributed by Adept in the \SYSTEM\ subdirectory on the Utility Disk.

4. Blank high-density diskettes (formatted or unformatted).

   You need four (4) blank high-density diskettes for:

   a. A copy of the old V+ system (if you do not already have one)

   b. A copy of the new V+ system before it is configured

   c. A working copy of the new V+ system for use during the configuration process if you choose to build the new system on a diskette or hard drive instead of on the Compact Flash

   d. A backup copy of the new V+ system after it is configured

The entire installation process could take up to 45 minutes. Most of that time will be spent making backup copies of your old and new V+ systems; you can stop the process at any time while making these copies.

However, after the process of *configuring* your new V+ system has begun, do not interrupt the installation process. The system informs you when the uninterruptible portion of the process has been reached, and you can stop at that point.

> **NOTE:** Use the edition of CONFIG_C that you received with the new V+ system.
>
> Make sure you clearly identify the diskettes used during the installation process. There is no way for the program to tell if you have inserted the wrong diskette in response to a request. Carefully read the program messages before you respond to each prompt.

During installation, answer the program questions about the system disks.

In various situations, the program recommends that you make a backup copy of the old and/or new V+ systems. If you agree, DISKCOPY is invoked to make the copy. You should be familiar with the use of the DISKCOPY program to copy a V+ system and to copy a diskette image.

> **NOTE:** In most situations Adept strongly recommends that you make a backup, and you are asked a second time if you want to make the backup.

After the appropriate backup disks are made, the program asks you to put one or the other of the system disks in the floppy drive, so configuration information can be read and confirmation can be made that needed files are present on the new system disk.

Next, the actual installation process begins, and it should not be interrupted. The following steps are performed:

1. The new system is copied to the specified output drive.
2. The appropriate device modules are installed in the V+ system file on the output disk.

   NOTE: If the installer cannot find the necessary device-module file(s), you are asked for the disk to be accessed.

3. The configuration information from the old system is written to the output disk.

   NOTE: No check is made to ensure the robot selections defined in the configuration data match the device modules that are present on the disk. That is, if the old system is incorrectly configured, the new system is also.

Finally, you are told what steps must be performed to load the robot data from the disk files created in step #1 of the three-step installation process.

```
3 => APPLY saved ROBOT DATA
```

This step applies the saved robot and encoder data from the old V+ system to the new V+ system disk.

The system must have been booted from the disk created with step #2 above.

After the purpose of the menu item is displayed, you are given an opportunity to cancel the menu selection.

Next, the robot and external encoder configuration of the current system is displayed, and you are given another chance to cancel the process.

You are asked which disk drive contains the configuration data files. If an optional floppy drive is indicated, you are asked to insert the diskette.The program looks for the configuration data files on the specified disk in the subdirectory \CONFIG_C\ with the names that were displayed during step #1.

After confirming that the required files are present, you are given one more chance to cancel the process. (If any required file is not found, its file specification is displayed, and the process cannot be continued.)

If you agree to continue, all the data files are read automatically. Each file name is displayed for reference. For example, if an error occurs, the file name should help you to understand the problem.

> **NOTE:** As the files are read, several lines of output may be displayed. You are advised by the program to use CRTL+S/CRTL+Q to control the output. Some of those lines might appear to be error messages. (Refer to the documentation for the SPEC utility in the ***AdeptMotion VME Developer's Guide*** for an explanation of the messages displayed while a configuration data file is being read.) In general, such messages result from changes that have been made to the V+ system or the CONFIG_C program and do *not* indicate a real problem.

After all the data files are read, you are asked if that data should be written to a V+ system disk. If so, you are told that the system disk must be the one from which the current V+ system was booted. If you indicate that the data should not be written to a V+ system disk, the ramifications are explained and you are asked to confirm your choice.

```
2 => V+ System Configuration Data
```

This selection accesses the configuration data on a V+ system disk, not the configuration data currently in system memory.

> **NOTE:** The *target* V+ system should be compatible with the edition of CONFIG_C being used.

```
*** CONTROLLER CONFIGURATION EDITOR MENU ***

1 => DISPLAY system CONFIGURATION
```

This option displays the configuration data on a specified V+ system disk.

```
2 => EDIT system CONFIGURATION
```

> **NOTE:** The configuration editor does not check to see if your changes are consistent with system requirements. If the V+ system fails to boot after the configuration has been changed, you need to boot from a different V+ system disk and then use CONFIG_C to correct the configuration data problem(s) on the inoperable system disk.

After you select this option, the system asks for the drive that holds the operating system you want to change. After a drive is specified and the configuration data is read from the disk, the following submenu is displayed:

```
 0 => DONE editing

 1 => Change HEADER configuration
 2 => Change VPLUS configuration
 3 => Change ANALOG_INPUT configuration
 4 => Change ANALOG_OUTPUT configuration
 5 => Change DIGITAL_INPUT configuration
 6 => Change DEVICENET configuration
 7 => Change NETWORK configuration
 8 => Change ROBOT configuration
 9 => Change SERIAL configuration
10 => Change SERVO_BOARDS configuration
11 => Change SYSTEM configuration
12 => Change TASKS configuration
13 => Change VISION configuration
```

Choosing a specific section allows you to edit the statements for that part of the system. The statements are displayed as equations. The left-hand side of the equation specifies the item that you are configuring. The right-hand side specifies the attributes that the item can have and the value each attribute is given at system startup. The general form of each statement is:

```
ITEM # = "/ATTRIBUTE value /ATTRIBUTE value.../ATTRIBUTE
value"
```

However, not all items have a number (#). Also, some items have only a value associated with them and not a list of attributes.

When you select a section, you are shown all the current statements in the section. You are then presented with each statement and given the opportunity to change that statement. You can delete statements or add new statements to the configuration file. In general, the sequence of steps for editing configuration statements is:

1. Select a section from the edit system configuration submenu.
2. The program displays all the statements in the selected section.
3. The program displays statements one at a time, and asks if you want to edit each one. You have three options:
    a. If you enter **n**, the next statement is displayed. If this is the last statement in a section, the program asks if you want to add a new statement.
    b. If you enter **y**, the program asks if you want to delete the statement. If you respond **y** to the delete prompt, the statement is removed from the configuration data.

If you respond **n** to the delete prompt, the first attribute in the statement is displayed along with its current value and its possible values (or range of values). To leave the value unchanged, press ENTER. To change a value, enter the desired value and press ENTER.

After you press ENTER, the next attribute and its values are displayed. After all attributes have been presented for editing, the revised statement is displayed and you can reedit if necessary.

c. If you select **q** (or complete editing the last statement in a section), the program asks if you want to add a new statement to the configuration data. If you respond **y**, a new statement will be started and you can specify the appropriate attribute values. If you respond **n**, the editing submenu is redisplayed.



CAUTION: You must save the modified configuration data on the system disk if you want the changes to be retained.

The options for the edit system configuration submenu are described below.

NOTE: There are often configuration statements for options that are not present in the controller. Such statements are ignored by V+ and do not need to be deleted.

```
1 => Change HEADER configuration
```

The statement in this section cannot be changed.

```
2 => Change VPLUS configuration
```

This section specifies which processors run a copy of the V+ interpreter. If an auxiliary processor is running only a servo or vision task, removing the V+ interpreter from that processor saves RAM.

NOTE: The V+ Extensions license is required to run V+ on more than one CPU.

The left-hand side of each statement specifies PROCESSOR and a number. Processor numbers are determined by the processor board address switches, not by their physical order in the backplane. See the ***Adept MV Controller User's Guide*** for details.

The right-hand side of the statement specifies the V+ system number to assign to the processor. The V+ system number and the processor number must be the same. To remove the V+ interpreter from a processor, delete the statement for that processor.

```
3 => Change ANALOG_INPUT configuration
```

The left-hand side of each statement specifies BOARD and a number. Analog I/O board numbers are determined by the analog board address switches, not by physical order in the backplane. See the ***Adept MV Controller User's Guide*** for details.

The analog input channels have two attributes, /TYPE and /MODE. The possible values for /TYPE are bipolar and unipolar. If any of the ± voltage ranges (±5V, ±10V, etc.) are selected in the hardware setup, the software setup must specify bipolar I/O. If a positive-only voltage range is hardware selected, the software setup must specify unipolar I/O. For input channels, all the channels on a board must have the same type selection.

The possible values for /MODE are single-ended and differential. Single-ended input uses the returned value of each input channel. Differential input uses the difference between returned values on pairs of input channels. The software setup must match the hardware setup.

```
4 => Change ANALOG_OUTPUT configuration
```

The left-hand side of each statement specifies BOARD and a number, and PORT and a number. Analog I/O board numbers are determined by the analog board address switches, not by physical order in the backplane. Contact Adept Applications for details. On each board, there are four analog channels, numbered 1 to 4. See the description of AIO.OUT in the ***V+ Language Reference Guide*** for details on addressing the individual channels.

The analog output channels have one attribute, /TYPE. The possible values for /TYPE are bipolar and unipolar. If any of the ± voltage ranges (±5V, ±10V, etc.) are selected in the hardware setup, the software setup must specify bipolar I/O. If a positive-only voltage range is hardware selected, the software setup must specify unipolar I/O. For output channels, bipolar or unipolar can be selected for each channel independent of the selections for the other channels.

```
5 => Change DIGITAL_INPUT configuration
```

The left-hand side of each statement specifies either POS_LATCH or VIS_TRIGGER and a number. These items can have one attribute, /SIGNAL. The signal number should be set to match the input channel (1001, 1002, 1003, or none) that will be used.

For position latches, the position latch number specified must match the latch number specified with the SERVO_BOARDS menu option (see **page 151**). See the ***Adept MV Controller User's Guide*** for an overview of the Position Latch and Digital Input features. See the LATCH and LATCHED keywords in the ***V+ Language Reference Guide*** for details on reading latched encoder positions.

For the vision trigger, the hardware setup of the EVI module must match the vision trigger number specified (see the ***Adept MV Controller User's Guide***). See the V.IO.WAIT keyword in the ***AdeptVision Reference Guide*** for details on using the vision trigger.

```
6 => Change DeviceNet configuration
```

The statements in this section provide configuration information for the Adept DeviceNet option. Refer to **"DeviceNet Configuration Setup" on page 159** and the ***Adept MV Controller User's Guide*** for information about these statements.

```
7=> Change NETWORK configuration
```

The statements in this section provide configuration information for the AdeptNet option. Refer to the ***AdeptNet User's Guide*** for information about these statements.

```
8=> Change ROBOT configuration
```

> **NOTE:** The main-menu item "Robots and Device Modules" provides a more convenient method for configuring the robots.

This section specifies which kinematic module is associated with each robot.

The left-hand side of each statement specifies ROBOT and a number. The ROBOT statement has one attribute, /MODULE. The value of /MODULE depends on the kinematic modules installed in your system and robot types connected to the controller. See the ***AdeptMotion VME Developer's Guide*** for details on selecting and specifying motion devices.

```
9=> Change SERIAL configuration
```

This section specifies the communications protocols and parameters for various serial data communications hardware and software.

The left-hand side of each statement specifies either GLOBAL_PORT (for the global serial lines available on the SIO board) or LOCAL_PORT (for the local serial lines available on the processor). The global serial lines can be accessed by all processors. The local serial lines can be accessed only by the processor they are attached to. GLOBAL_PORTs have a number that refers to the serial port number (port #4 is the FP/MCP connection). LOCAL_PORTs have a number that corresponds to the processor number, and a number that refers to the serial port number on the processor. The processor number is determined by an address switch on the processor board (see the ***Adept MV Controller User's Guide***).

GLOBAL_PORTs and LOCAL_PORTs have the following attributes: /DRIVER, /SPEED, /BYTE_LENGTH, /PARITY, /STOP_BITS, /FLOW, /FLUSH, /DTR, /MULTIDROP, and /BUFFER_SIZE. See the ***V+ Language User's Guide*** for general details on serial I/O. See the documentation on any serial devices you may be using for details on the correct settings. Most serial I/O settings can also be changed at runtime with the FSET monitor command or program instruction.



CAUTION: For each of the following pairs of serial ports on the SIO module, the baud rate 19200 cannot be used for one of the ports if baud rate 7200 or 38400 is used for the other port in the pair:

GLOBAL_PORT 1 and GLOBAL_PORT 4
GLOBAL_PORT 2 and GLOBAL_PORT 3

```
10 => Change SERVO_BOARDS configuration
```

This section associates servo interface boards (EJI, MI3, MI6, or VFI) with processor boards. In multiprocessor systems, you may elect to run the servo task for a given servo board on any of the processor boards in the system.

The left-hand side specifies BOARD and a number. The servo interface board numbers are determined by the board address switches, not by physical order in the backplane. See the ***Adept MV Controller User's Guide*** for more details.

The BOARD statement can have two attributes: /PROCESSOR and /POS_LATCH. The /PROCESSOR attribute specifies the processor that runs the servo task for this servo board. The value of /PROCESSOR is a list of processor numbers. The system looks through the list and associates the servo board with the first installed processor that is in the list. For example, if the attribute "/PROCESSOR 3|2|1" is specified, the servo board is associated with processor 3 if it is installed, processor 2 if processor 3 is not installed, or processor 1 if processors 3 and 2 are not installed.

The /POS_LATCH attribute specifies which (if any) high-speed backplane signals each servo board will use for latching robot and encoder positions. The backplane Position Latch signals can be triggered from either the CIP (configured with the DIGITAL_INPUT menu option; see **page 149**) or from a vision (EVI) board (configured using the V.STROBE system switch or the ADV_CAL.V2 utility program).

```
11 => Change SYSTEM configuration
```

This section specifies various characteristics of the V+ system.

DEFAULT_DISK can be given any valid device, unit, and directory path specification. The default disk can be changed at runtime with the DEFAULT monitor command.

DEFAULT_SPEED can be given an integer value from 1 to 100. The monitor speed for motion can be changed with the SPEED monitor command or program instruction.



WARNING: Before setting the default, be sure that your mechanism can safely and reliably move at the specified speed.

The GRAPHICS statement can have the attributes /MAX_WINDOWS, /MAX_ICONS, and /DOUBLE_CLICK. /MAX_WINDOWS specifies the maximum number of windows that can be open at one time. /MAX_ICONS specifies the maximum number of icons with different names that can be loaded. /DOUBLE_CLICK specifies the time interval (in milliseconds) for detecting double clicks (as opposed to two single clicks). Two clicks separated by less than this interval are processed as a double click.

NOTE: The GRAPHICS attributes /MAX_ICONS and /DOUBLE_CLICK apply only to graphics displayed by a VGB. Those attributes do not apply when the AdeptWindows option is being used to display graphics.

The PASSIVE_ESTOP statement controls the operation of the CIP Passive ESTOP Output signal. The Passive ESTOP Output contacts in the CIP normally are closed (i.e., the output signal is on) when motor High Power is on. When the PASSIVE_ESTOP statement specifies IGNORE_ENABLE (which is the default), the Passive ESTOP Output signal does not follow the state of the MCP enable switch during manual mode. That is, although motor High Power will be off when the MCP enable switch is released, the Passive ESTOP Output signal will stay on as if High Power were on. When TRACK_ENABLE is specified, the CIP Passive ESTOP Output signal is turned off whenever the MCP enable switch is released, even during manual mode.

The POWER_TIMEOUT statement determines how long V+ will wait, when enabling robot power, for the user to press the Robot Power button on the CIP after the Robot Power light starts blinking. The default setting is 10 seconds.

SERVO_RATE can have the values 1KHz or 500Hz. This value must match the servo software, kinematic module, and hardware configuration. Do not change this value unless directed to do so by Adept or the system integrator.

STARTUP_MESSAGE allows you to display a custom message at system startup and whenever the ID command is issued. It is displayed as the first line of the ID information. The default is no message and the maximum length is 80 characters.

TRAJ_RATE can have the values 62.5 Hz, 125 Hz, 250 Hz, or 500 Hz. Do not change this value unless directed to do so by Adept or the system integrator.

```
12 => Change TASKS configuration
```

V+ is a multitasking system in which user tasks compete with system tasks for use of the processor. User and system tasks run on a prioritized basis in one or more of 16 one-millisecond time slices within each 16-millisecond major cycle. Refer to the ***V+ Language User's Guide*** for details on task scheduling.



CAUTION: Operation of the V+ system can be adversely affected by incorrect settings of task priorities. Change the default configuration only if you have a good understanding of V+ task scheduling.

The left-hand side of each statement specifies USER and a user task number[1] and PROCESSOR and a processor number. Processor numbers are determined by the processor board address switches, not by physical order in the backplane. See the ***Adept MV Controller User's Guide*** for more information.

The attribute for USER x PROCESSOR y is a sequence of 16 integers that specifies the user task priority in each of the 1-ms time slices. The possible values are:

–1	Do not run in this slice even if no other task is ready to run.
0	Do not run in this slice unless no other task from this slice is ready to run.
1 - 64	Run in this slice according to the specified priority. Higher priority tasks may lock out lower ones. Priorities are broken into the following ranges:
1 - 31	Normal user task priorities.
32 - 62	Used by V+ device drivers and system tasks.
63 - 64	Used by the trajectory generator. Do not use 63 or 64 unless you have very short task execution times. Use of these priorities may cause jerks in robot trajectories.

[1] Standard V+ allows 7 user tasks. Systems with the V+ Extensions license are allowed 28 tasks.

The program asks you for a priority for each time slice.

The priorities of the system tasks cannot be changed.

After the task priorities have been modified (and the V+ system has been rebooted), the PROFILER utility can be used to examine the effects of the changes.

```
13 => Change VISION configuration
```

This section associates a vision system with a processor and sets the vision memory allocation and the virtual frame buffer size.

The left-hand side specifies PROCESSOR and a list of one or more processor numbers. The system looks through the list and associates the vision system with the first installed processor that is in the list. For example, if "PROCESSOR 3|2|1" is specified, the vision system executes on processor 3 if it is installed, processor 2 if processor 3 is not installed, or processor 1 if processors 3 and 2 are not installed.

The attributes for the vision PROCESSOR statement are:

/ID specifies which vision system should be associated with the indicated processor (can have the value 1 or 2).

> **NOTE:** If there is a statement with /ID 2, it *must* appear *before* the statement with /ID 1.

/FRAME_SIZE specifies the size of the virtual frame buffers that the physical frame store is divided into (see the ***AdeptVision User's Guide***).

/TOTAL_KB specifies the total kilobytes of system memory that should be reserved for vision processes. It needs to include all the allocations listed below, 512KB for the vision system, and memory for fonts, prototypes, and miscellaneous requirements (start with at least 200KB for these items).

/AOI_KB specifies the memory allocation for area-of-interest definitions (see the description of VDEF.AOI in the ***AdeptVision Reference Guide***).

/BLOB_KB specifies the memory allocation for blobs (bounded regions).

/BOUND_KB specifies the memory allocation for performing boundary analysis (see the section on boundary analysis in the ***AdeptVision User's Guide***).

/LUT_KB specifies the memory allocation for vision look-up tables (see the description of VDEF.LUT in the ***AdeptVision Reference Guide***).

/OBJECT_KB specifies the memory allocation for object data structures.

/RUN_LENGTH_KB specifies the memory allocation for run-length encoding (see the appendix on vision algorithms in the ***AdeptVision Reference Guide***).

/UNMATCHED_KB specifies the memory allocation for unmatched objects.

/VTRANS_KB specifies the memory allocation for vision transformations (see the description of VDEF.TRANS in the ***AdeptVision Reference Guide***).

See the appendix titled "Using DEVICE With Vision" in the ***AdeptVision User's Guide*** for additional details on setting vision memory allocation.

Setting the IP Address of the Adept Controller

The IP address of the Adept controller (the controller's address on the local area network) is stored in the NVRAM on the AWC board. The CONFIG_C utility is used to change the IP address.

If you need to change the IP address, follow these steps:

1. Load and execute the CONFIG_C program as described on **page 135**.
2. Select "`Controller NVRAM`" from the main menu.
3. Select "Set IP ADDRESS in the NVRAM" from the submenu.

   The current IP address is displayed. If there is no address stored in the NVRAM, the default address for this AWC module is shown.

   The default IP address is based on the serial number of the AWC board. Thus it is possible to use the default address for a network that consists of multiple Adept controllers and a PC (or other server). See the ***AdeptNet User's Guide*** for more information on the default IP address.

4. Enter the desired IP address (in the format displayed), or just press the ENTER key to retain the setting displayed.
5. After you have set the IP address, you need to set the IP subnet mask. If you use the AWC controller on a LAN and you do not know the proper IP subnet mask, ask your network system administrator. For further information see the ***AdeptWindows User's Guide*** and the ***AdeptNET User's Guide***.
6. After you have entered new values, the program asks if you want them stored in NVRAM. Type **y** to have the new values stored in the NVRAM. You must reboot the controller to have the changes take effect.

Software Configurable Switch Settings

In contrast to other Adept VME processor boards and the SIO board, the AWC allows you to define V+ configuration settings in software rather than using hardware DIP switches. Since the hardware DIP switches on the AWC are on the board and not the card edge/front panel, this facility allows you to change these settings without removing the AWC board. See the ***Adept MV Controller User's Guide*** for information about the hardware DIP switches.

> **NOTE:** The Software switch settings are active only when all the hardware DIP switches on the AWC board are in the ON position, or all are in the OFF position. If this is not the case, V+ uses the user interface and autoboot settings selected by the hardware DIP switch settings on the AWC module.

To make changes to the Software switches, start the CONFIG_C utility as described on **page 135** and select the menu option "`Controller NVRAM`".

Then select the following item from the submenu:

```
Set "SWITCHES" in the NVRAM
```

The following material describes the various switch settings.

User Interface

> **NOTE:** The hardware DIP switches on the AWC board can be used to override these settings of the Software switches.

The AWC board offers the following user interfaces:

- ASCII-based user interface via serial connection to an ASCII terminal
- Graphical user interface via VGB module, monitor, and keyboard
- Graphical user interface via serial connection to AdeptWindows PC software on PC
- Graphical user interface via Ethernet to AdeptWindows PC software on PC

ASCII-Based User Interface

To use an ASCII-based user interface (like a WYSE Terminal or terminal emulation software on a PC) you have two choices:

```
1 =>User interface via VGB or serial port
2 =>User interface via serial port
```

If you have a VGB board in the system, you must select option 2 to direct the V+ monitor to an ASCII terminal.

If you do not have a VGB board in the system, you can select option 1 or option 2 to direct the V+ monitor to an ASCII terminal.

VGB-Based User Interface

If you have a VGB module in your controller, select:

```
1 => User interface via VGB or serial port
```

to use it as your graphical user interface. A monitor and keyboard must be connected to the VGB board. See the VGB board chapter in the ***Adept MV Controller User's Guide*** for further information.

AdeptWindows PC Setup

The AdeptWindows PC graphical user interface can be implemented via a serial or an Ethernet connection. For an Ethernet connection, the IP address of the Adept controller must be correctly set, and on the PC side you must access exactly that address. In this manual we describe only the steps that must be performed on the Adept controller. Please see the ***AdeptWindows User's Guide*** for a full description of the required setup on the PC.

AdeptWindows PC Serial Interface

If you want to use the AdeptWindows PC interface via a serial line, you must select it with either the hardware DIP switch or the Software switches. To select a serial AdeptWindows PC interface with the Software switches, select:

```
3 => User interface via serial AdeptWindows
```

AdeptWindows PC Ethernet Interface

If you want to use the AdeptWindows PC interface with an Ethernet connection, you can use either the Adept default IP address or a user-defined IP address. (See **page 155** for information on the Adept default IP address and how to set the user-defined IP address.)

> **NOTE:** The default IP address is always used if no user-defined address has been set, or if the controller NVRAM is found to have invalid contents.

Depending on your preference, select the appropriate setting from one of the following:

```
4 => User interface via Ethernet ... (use IP address in NVRAM)
```

```
5 => User interface via Ethernet ... (use default IP address)
```



WARNING: Adept strongly recommends that shielded Ethernet cables be used to connect the AWC board to an Ethernet network. Use only hubs that provide shielded RJ45 connectors. Adept cannot ensure safe and reliable operation of the Ethernet connection without proper shielding of all the Ethernet connectors and cables.

NOTE: When connecting to a hub, use a straight-through cable that connects the "Transmit" and the "Receive" pairs to the same pins on both ends of the cable. When connecting to a server/PC directly, use a cross-over cable that swaps transmit and receive pairs on the opposite ends.

V+ Auto-Start

Use this setting to select whether or not the V+ system will automatically load a file named AUTO.V2 and execute a command program named AUTO during the process of loading and initializing V+. Select option 1 to disable this feature, or option 2 to enable the feature. See the ***V+ Operating System User's Guide*** for details on auto-start programs.

Boot Sequence

Use this option to select the sequence of drive letters that will be accessed when locating an operating system to load into memory. If option 1 is selected, the boot procedure will first look to drive A for an operating system, then drive B, then drive C, and, finally, drive D. If option 2 is selected, the search sequence is changed to drive B, then A, then D, then C.

NOTE: The two search sequences are effectively the same if there is no SIO board installed in the Adept controller.

Auto-Boot

Normally, the controller is configured to automatically boot V+ when power to the controller is turned ON (that is, auto-boot is normally enabled). Occasionally, you may want to manually specify the disk device from which to load V+. You can disable auto-boot for such purposes.

NOTE: Auto-boot is automatically disabled if the hardware DIP switches on the (main) AWC board are set to a combination other than all OFF. Thus, for example, setting the hardware switches to all ON will cause auto-boot to be disabled, while all the other Software switches will still be used.

DDT Serial I/O on Auxiliary CPU

This option is used only for special diagnostic purposes and is normally disabled. When DDT serial I/O is enabled, DDT start-up messages are output to the TERM serial ports of any auxiliary processors in the controller, and DDT commands can be input via those ports.



WARNING: The controller could appear to function normally if this setting is enabled. However, the start-up messages that are output to the TERM ports could have an unexpected effect on any equipment that is connected to those ports.

For further information about the functionality of the various switch settings see the *Adept MV Controller User's Guide*.

DeviceNet Configuration Setup

The AWC board incorporates DeviceNet scanner functionality. The Adept Utility program CONFIG_C supports configuring the DeviceNet and mapping the DeviceNet hardware to the appropriate V+ signal numbers.

Please follow the instructions below to set up the DeviceNet functionality.

Scanning the DeviceNet for Setup Information

1. Load and execute the CONFIG_C program as described on page 134:
2. Select `Scan the DeviceNet` from the main menu, and then select `Scan the DeviceNet for nodes` from the submenu.



WARNING: No other execution task should be started while this program is running.

3. The program asks you for the Local MAC ID (Media Access Control Identifier) to use for the Adept controller. It can be a value from 0 to 63. The default value is 0. You can enter –1 to cancel the scanning process.

   To avoid a conflict on the network, make sure you select a MAC ID that is different from those for all the connected modes. If you have more than one Adept AWC controller connected to the DeviceNet, you must specify a unique MAC ID in the system configuration data for each Adept controller.

4. Next you are asked to select the baud rate to be used for DeviceNet communications. The baud rate to use depends on the capabilities and configuration of the hardware installed on the DeviceNet and on the length of

the trunk lines. All the devices on the network must be configured for the same baud rate. (See the DeviceNet standard and the documentation for your DeviceNet nodes for further information.)

After you enter a selection for the baud rate, the program stops the V+ DeviceNet scanner, resets and restarts it, and begins scanning the DeviceNet for nodes. You will see information similar to that shown in **Figure 4-1**, although the details will depend on the hardware connected to your DeviceNet.

> **NOTE:** You may need to use CTRL+S to stop the output temporarily, so the information will not scroll off the screen before you can read it. Use CTRL+Q to resume the output.

```
Local MAC ID to use (0 to 63, default is 0; -1 to cancel):  0

Baud rate (1 for 125K [default], 2 for 250K, or 3 for 500K):  1

Stopping the scanner...     Success
Resetting the scanner...    "DeviceNet: Device not found"
Restarting the scanner...   Success
Gathering MAC IDs .......

MAC ID 1:  Input size   = 2
           Output size  = 1
           Vendor ID    = 40
           Device type  = 12
           Product code = 306
           Status       = 0

MAC ID 2:  Input size   = 1
           Output size  = 2
           Vendor ID    = 40
           Device type  = 12
           Product code = 306
           Status       = 0


Resetting the system (please wait)...

Press ENTER to resume.
```

Figure 4-1. Example DeviceNet Scanning Results

In the example shown in **Figure 4-1**, the information acquired by the scanner from two WAGO I/O SYSTEM 750-306 blocks, connected as slaves to the DeviceNet, is displayed. These I/O blocks can be populated with a variety of digital I/O bus modules.

Table 4-2 shows the information you need to record for each DeviceNet component that you want to access with the Adept Controller.

Table 4-3 describes the items listed in **Table 4-2**, as well as the other information that is reported for each component.

Table 4-2. DeviceNet Scanner Results

MAC ID	Input Size	Output Size	Vendor ID	Device Type	Product Code	Status
1	2	1	40	12	306	0
2	1	2	40	12	306	0

Table 4-3. Interpretation of DeviceNet Scanning Results

Keyword	Result	Interpretation
Mac ID	1	User has configured the first node with MAC ID 1 (for example with DIP switch settings).
Input Size	2	During operation of the DeviceNet, the WAGO I/O block will supply this many data bytes to the V+ system.
Output Size	1	During operation of the DeviceNet, the WAGO I/O block will expect this many data bytes from the V+ system.
Vendor ID	40	WAGO DeviceNet vendor identification number.
Device Type	12	Device type number of the WAGO I/O System 750-306 block.
Product Code	306	Identification that WAGO chose for the I/O System 750-306 block.
Product Name		The product name supplied by the WAGO I/O System. (This information is not used by the V+ system.)
Major Revision		A product version ID supplied by the WAGO I/O System. (This information is not used by the V+ system.)
Minor Revision		A product version ID supplied by the WAGO I/O System. (This information is not used by the V+ system.)
Serial Numbers		Product ID information supplied by the WAGO I/O System. (This information is not used by the V+ system.)
Status	0	Status information of the coupler.

Please make up a worksheet and fill in your scanning results. You will need them later to configure your DeviceNet.

DeviceNet Statements in V+ Configuration Data

After you have scanned the nodes on the DeviceNet bus, you need to use the information from the scanning process to configure the components in V+. If you are still in the CONFIG_C program, you can simply go back to the main menu. If not, load and execute the CONFIG_C program.

> **NOTE:** All the examples in this manual are based on the WAGO I/O System 750-306 as a slave on the DeviceNet bus. If you do not use this DeviceNet equipment, you need to ask your DeviceNet vendor for the correct process image of your components.

Figure 4-2 shows a typical configuration for the mapping of three digital inputs and one output on the DeviceNet bus to V+ input signals 1033, 1034, and 1035 and V+ output signal 33. The configuration is specific to the WAGO I/O System 750-306 block. The configuration statements are described in detail below.

```
.DEVICENET
        LOCAL = "/MACID 0 /BAUD 125K"
        MACID 1 = "/INPUT 6 /OUTPUT 5 /VENDOR ID 40 /DEVICE TYPE 12
                /PRODUCT CODE 306 /DISABLE POWER NO"
        MACID 2 = "/INPUT 6 /OUTPUT 5 /VENDOR ID 40 /DEVICE TYPE 12
                /PRODUCT CODE 306 /DISABLE POWER YES"
        MAPPING 1 = "/MACID 1 /BYTE 3 /BIT 2 /BIT LENGTH 1 /SIGNAL 33"
        MAPPING 2 = "/MACID 1 /BYTE 2 /BIT 2 /BIT LENGTH 3 /SIGNAL 1033"
```

Figure 4-2. Typical DeviceNet Configuration for Digital Input and Output

To modify the DeviceNet configuration data, go to the main menu in the CONFIG_C program:

1. Select:

   ```
   2 => V+ System Configuration Data
   ```

2. From the displayed menu, select:

   ```
   2 => EDIT system CONFIGURATION
   ```

3. Enter the letter for the drive that contains your system data. For example, enter D for the Compact Flash in the AWC module.

4. Select:

   ```
   6 => Change DEVICENET configuration
   ```

The program displays the current configuration statements if there are any. (The V+ system is supplied by Adept with no DeviceNet configuration statements.) If there are no statements, you need to add the appropriate statements. Otherwise, you can modify the existing statements as required for changes to your DeviceNet setup.

LOCAL Statement

The `LOCAL` statement in the DeviceNet configuration specifies the MAC ID of the Adept controller on the DeviceNet bus. The default setting is 0. Set the MAC ID so that all the nodes on the bus have different MAC IDs.

This statement also defines the baud rate of the DeviceNet scanner. The baud rate depends on multiple factors, such as the length of the DeviceNet cable, the DeviceNet components on the bus, etc.

Syntax of the `LOCAL` Statement:

```
LOCAL =  /MACID awc_id
         /BAUD baud_rate
```

Parameters:

- awc_id — MAC ID for the Adept controller on the bus
- baud_rate — The baud rate to be used on the DeviceNet

MACID Statement

The `MACID` statement defines the parameters for a node on the bus. The DeviceNet scanner uses this information to communicate with each node. There must be a `MACID` statement for every node (for example, each I/O coupler) that the V+ system is to access.

Syntax of the `MACID` statement:

```
MACID node_macid = /INPUT in_bytes
                   /OUTPUT out_bytes
                   /VENDOR_ID vend_num
                   /DEVICE_TYPE dev_num
                   /PRODUCT_CODE prod_num
                   /DISABLE_POWER pow_switch
```

Parameters:

- node_macid — MAC ID number of the node on the bus [0 to 63]

- in_bytes — Number of data bytes of input from the node [0 to 32767]
- out_bytes — Number of data bytes of output to the node [0 to 32767]
- vend_num — Vendor Identification number, assigned by ODVA. Use the number that the scanning process found.
- dev_num — Device Number of the node. Specifies the type of the node, for instance an I/O block.
- prod_num — Specified by the DeviceNet vendor.
- pow_switch — Has two values, YES and NO. The default value is NO. If set to YES, robot power cannot be enabled if the specified node is off-line; power is turned off if the node ever goes off-line.

Except for the `/DISABLE_POWER` parameter (which you can set as you wish), the values for all the parameters should be taken directly from the results of scanning the DeviceNet (see **page 159**).

> **NOTE:** The values of the `/VENDOR_ID`, `/DEVICE_TYPE`, and `/PRODUCT_CODE` parameters are currently ignored by the V+ system.

MAPPING Statement

The `MAPPING` statement is used to map the process image of a node to the appropriate V+ signals. This part of the DeviceNet setup requires that you define the way the connected nodes organize their I/O data in messages to and from V+. Please see the documentation of your DeviceNet vendor for this information.

In this manual we describe the mapping of three digital inputs and one output of a WAGO I/O System 750-306 block to the corresponding V+ signals.

First we describe the syntax of the `MAPPING` statement, and then we give an example.

Syntax of the `MAPPING` statement:

```
MAPPING map_index = /MACID node_macid
                    /BYTE byte_num
                    /BIT bit_num
                    /BIT_LENGTH bit_len
                    /SIGNAL sig_num
```

Parameters:

- `map_index` — This is an integer number that is used as an index for mapping DeviceNet information to V+. [1 to 100]
- `node_macid` — MAC ID number of a selected node on the bus [0 to 63].
- `byte_num` — Start reading/writing data at the specified byte number. [1 to 32767]
- `bit_num` — Start reading/writing information at the specified bit number in the specified byte. [1 to 32767]
- `bit_len` — Number of bits to read/write. [1 to 32767]
- `sig_num` — Starting number of a range of V+ signals that receive/transmit status information from/to the selected node. The range is specified by the number in `bit_len`.

  This number establishes whether the bits being mapped are for input or output. If the number is for a V+ input signal (1001 through 1512) , the bits will be used for inputs. If the number is for a V+ output signal (1 through 512), the bits will be used for outputs.

Figure 4-3 and **Figure 4-4** show examples of DeviceNet bit mapping to V+ signals. The example in **Figure 4-3** shows the mapping of three digital inputs and one output of a WAGO I/O block to the specified V+ signals.

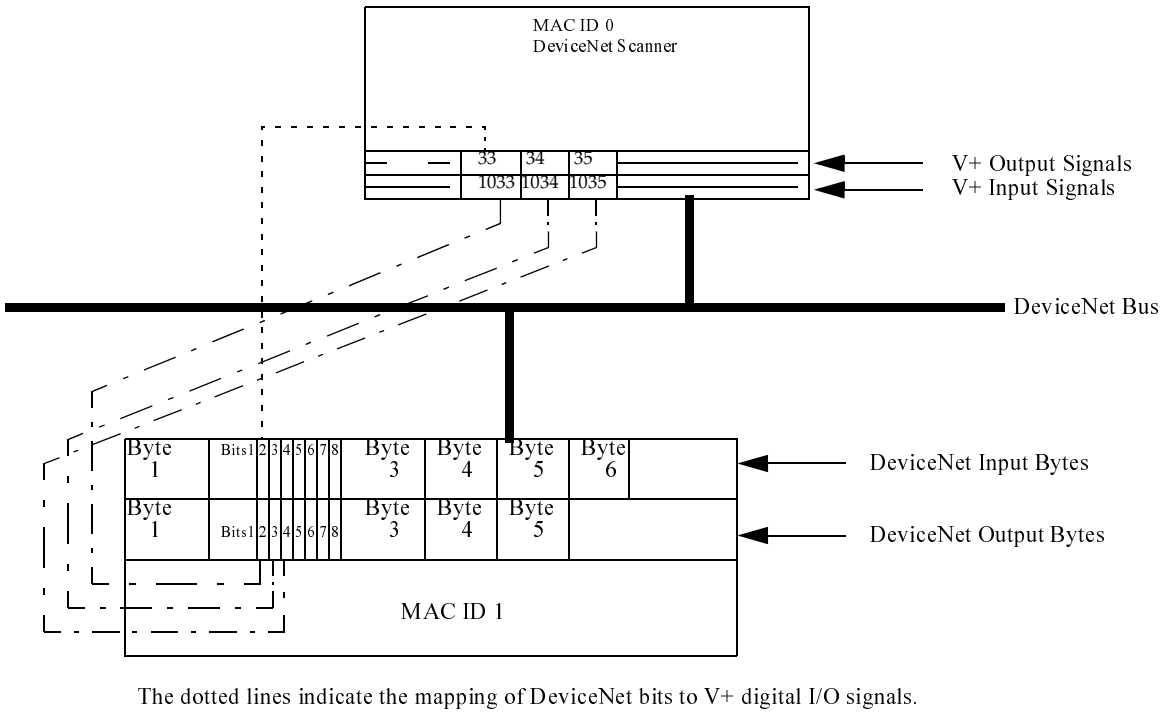


Figure 4-3. DeviceNet Bit Mapping to V+ Signals

```
Current statements in the DEVICENET section:

        MAPPING 1 = "/MACID 1 /BYTE 2 /BIT 2 /BIT_LENGTH 1 /SIGNAL 65"
        MAPPING 2 = "/MACID 1 /BYTE 2 /BIT 2 /BIT_LENGTH 1 /SIGNAL 1065"
```

Figure 4-4. Example of DeviceNet Bit Mapping in V+

Figure 4-4 shows the mapping of V+ signals 33, 1033, 1034, and 1035 to digital I/Os on the DeviceNet. As you can see in **Figure 4-3**, the WAGO I/O block reads 6 bytes as an input stream from the DeviceNet bus and writes 5 bytes as an output stream over the DeviceNet bus to V+. The information about how many bytes V+ reads from the DeviceNet bus and how many it writes is specified in the `MACID` statement. That is why the `MACID 1` statement in **Figure 4-4** has the value 6 assigned to the `/INPUT` and the value 5 to the `/OUTPUT` parameter. These values were found while scanning the DeviceNet bus with the CONFIG_C program.

The `MAPPING` statements in **Figure 4-4** show map information for the digital inputs and outputs of a WAGO I/O block to digital I/O signals in V+.

The `MAPPING 1` statement in **Figure 4-4** shows how to map one digital output on the WAGO I/O block to one digital I/O signal in V+.

Figure 4-3 shows the bit mapping for V+ output signal 33. First we select the DeviceNet component to which we want to write information with the parameter `/MACID` in the `MAPPING` statement.

Figure 4-4 shows that V+ will write information to `/MACID 1`, which is the WAGO I/O block that we have previously configured with the `MACID 1` statement. The `/BYTE` parameter defines the start byte of the DeviceNet Output stream to the I/O block.

After defining the start byte of the data stream, the `/BIT` parameter specifies the first bit that is mapped to a V+ signal.

The `/BIT_LENGTH` parameter defines how many bits we want to map to V+ signal numbers. In the `MAPPING 1` statement the parameter `/BIT_LENGTH` has the value 1. This means we map exactly one bit to one signal, starting with the signal number given in the `/SIGNAL` parameter. In the `MAPPING 2` statement the parameter `/BIT_LENGTH` has the value 3, which means we map three signals, starting with signal 1033.

The `/SIGNAL` parameter specifies the starting number of a sequence of V+ signals. The number of signals is specified with the value of the `/BIT_LENGTH` parameter in the `MAPPING` statement.

In summary, V+ signal 33 is mapped to bit number 2 in the second DeviceNet input byte to the I/O block.

The mapping of digital inputs of the WAGO I/O block on the DeviceNet bus to V+ signals requires reading the DeviceNet output information that the block writes to the bus. The first byte of its DeviceNet output string does not contain data. Starting with the second bit in the second byte we receive data for the selected digital inputs from the block. The state of all the digital inputs on the block follow as sequential bit information. In **Figure 4-4** the statement `MAPPING 2` maps, starting with the second bit, the three V+ input signals 1033, 1034, and 1035 to the I/O block. **Figure 4-3** shows the mapping of the V+ signals to the process image of the WAGO 750-306 I/O block on the DeviceNet bus.

SPEC

Description

Robot specification utility

Disk File Name

SPEC.V2

Program Name

a.spec

Functions

The SPEC program allows customizing of robot parameters for a particular application. This description documents the restricted access mode of SPEC seen with Adept robots. The unrestricted access mode of SPEC is required only for developers of custom robot systems and is documented in the ***AdeptMotion VME Developer's Guide***.

For systems with AdeptModules devices, the SPEC program must be used to specify the configuration of the devices when the system is initially set up. See the ***AdeptModules Instruction Handbook, Vol. 1, Adept MV Controller Interface,*** for detailed information about this procedure.

The SPEC utility contains new/enhanced options that might not be documented in your edition of the ***AdeptMotion VME Developer's Guide*** or your *Robot Instruction Handbook.* For details on these options, please refer to the sections listed below:

The following options are available only when using the unrestricted mode of the SPEC utility or when tuning Joint 4 of a SCARA robot in restricted mode.

- Changes to the frequency response tests. See **"Options for the Frequency Response Tests" on page 187**.
- Ability to add a description to SPEC-generated data files. See **"Adding a Description to a SPEC-Generated Data File" on page 189**.
- Support for static friction feedforward compensation. See **"Static Friction Feedforward Compensation" on page 190**.
- The Expert User feature. This allows an experienced user to enter select parameters by entering the menu option number followed by a comma and the new parameter value.
- The ability to set various SPEC Program Preferences that allow the user to configure global SPEC settings.

- The ability to create user-defined obstacles in the workcell.
- The ability to use a dedicated home sensor when calibrating an AdeptModules device.
- The ability to specify a manual mode envelope error limit that allows a smaller envelope error (that is, how far the motor is allowed to lag behind the commanded position) for motions made with the MCP.
- The ability to create a custom point-to-point motion routine that will execute in place of the standard routine in SPEC. See **"Custom Point-to-Point Motion Routine" on page 196**.
- The ability to specify a unique acceleration *and* deceleration percentage for the point-to-point move test during motor tuning. See **"Motion Acceleration and Deceleration Percentages" on page 187**.

The following robot parameters can be changed with the restricted access mode of SPEC:

- The digital signals that control the robot hand or end-effector
- The positional tolerances used at the end of motions
- The robot software joint limits
- The program-controlled motion speed above which accelerations are not to be scaled
- Joint 4 tuning parameters on Adept SCARA robots

Abstract

The V+ system contains internal data structures that describe robot parameters such as those mentioned above. This internal data can be modified whenever the configuration of any of the robot parameters needs to be changed to suit a particular application. This program makes the desired changes to the data.

Requirements

You need the following items to run SPEC. All of these, except as indicated, were supplied with your robot system.

- Adept robot system (or a non-Adept robot controlled with the AdeptMotion VME option)
- Diskette for storing configuration data file (optional)
- SPECSUB.LIB file (included on the V+ Utility Disk)

Before you run SPEC the system must have been booted from the *actual* system disk that will be modified during the configuration procedure.

Procedure

Load the program:[1]

```
load \util\spec
```

Execute the program:

```
execute 1 a.spec
```

Choose an option from the menu displayed (see below).

Follow the prompts provided by the program (see below).

You can delete the SPEC program from the system memory with the following commands:

```
kill 1
delete a.spec
```

When the SPEC utility is executed, it checks for the SPECSUB programs in system memory. If the programs are not found, SPEC attempts to load the file SPECSUB.LIB from the following paths:

DISK>B:\	DISK>C:\UTIL\	NFS>XA:\
DISK>A:\	DISK>D:\	NFS>XC:\
DISK>C:\	DISK>D:\UTIL\	NFS>XC:\UTIL\

SPEC searches for the preferences file (SPEC.PRF) in the following paths:

$DEFAULT () directory specification

D:\SPEC\

B:\ (if available)

A:\ (if available)

If the preferences file cannot be found or fails to load, the SPEC program reverts to the system defaults (i.e., no error is generated).

Special Considerations

The configuration program defines many real-valued and string variables with the prefix “sp.” that must not be modified by the user or by other programs.

[1] Use the correct drive and path specification if the file is stored on another drive or subdirectory.

> **NOTE:** Some variables are intentionally left in RAM when the program exits normally. For example, the name of the custom point-to-point motion routine is left in memory (see **"Custom Point-to-Point Motion Routine" on page 196**). However, all variables are deleted when a DELETE command is used to delete the program from memory (as described earlier).

The disk file SPEC.V2 is a protected binary file. Thus, the programs in the file cannot be displayed, edited, or stored from memory to a disk. The file can be copied from one disk (or subdirectory) to another with the FCOPY monitor command or the DISKCOPY utility program. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

> **NOTE:** TIFF file exports are not supported when using the AdeptWindows PC interface.

Menu Structure

> **NOTE:** All menu items (both restricted and unrestricted access) are listed. Footnotes are included that describe any special access or usage requirements for corresponding menu items. Menu structures may be different depending on the robot and external encoder configuration of your V+ system.

```
Exit to system monitor

Enter password to increase access[1]

AdeptModules quick setup[2]

Change robot options and motor configuration[3]

Perform hardware diagnostics

Edit robot specifications

   Exit to main menu
   Edit robot initialization specs
```

[1] Password-protected robots only.
[2] Only for systems with AdeptModules devices.
[3] Display of this menu item and options depends on the type of device connected to your system and/or user access level.

```
  Exit to robot menu
  Hand OPEN signal
  Hand CLOSE signal

Edit motor amp/encoder specs[1]

  Exit
  Change motor number
  Encoder scale factor (ct/mm or ct/deg)
  Encoder counts per zero index (ct)
  Encoder configuration
  Encoder sign
  Motor sign
  Maximum DAC output
  Maximum DAC output in Manual mode
  *Duty-cycle exceeded* DAC limit
  *Duty-cycle exceeded* filter parameter
  *Motor stalled* timeout (sec)
  *Manual mode envelope error* limit (cts)
  *Soft envelope error* limit (cts)
  *Hard envelope error* limit (cts)
  Velocity envelope error limit (ct/ms)
  Machine input polarity
  Perform hardware diagnostics

Edit motor tuning parameters[1]

  Exit
  Change motor num
  Proportional gain...
  Proportional zero...
  Proportional pole...
  Integral gain.......
  Max integrator value
  Max integrator step.
  Velocity feedforward
  Accel feedforward...
  Static friction ffwd...
  DAC output filter...
  S-A: Skew error gain
  S-A: Max velocity FF
  Select test.........
```

[1] Display of this menu item and options depends on the type of device connected to your system and/or user access level.

```
  Test and plot
  Test and display
  Export results
  STOP Test[1]

Edit motor calibration parameters[2]

  Exit
  Change motor number
  Change homing configuration
  Speed and direction (ct/msec)
  'Motor stalled' timeout during cal (msec)
  Maximum search distance (ct)
  Maximum home switch width (ct)
  Distance from home switch to zero index (ct)
  Motor position at zero index (ct)
  Calibration groups
  Park position after calibration (ct)
  Teach calibration specs
  Calibrate motor
  Calibrate robot

Edit joint motion specs

  Exit
  Change joint number
  FINE nulling tolerance (cts)
  COARSE nulling tolerance (cts)
  Lower joint limit (mm or deg)
  Upper joint limit (mm or deg)

Edit link dimensions[1]

Edit Cartesian motion specs[1]

  Exit to robot menu
  Cartesian translation speed at SPEED 100 (mm/sec)
  1st Cartesian rotation speed at SPEED 100 (deg/sec)
  Translation acceleration at ACCEL 100 (mm/sec^2)
  Accel for 1st rotation at ACCEL 100 (deg/sec^2)
  Max WORLD/TOOL pendant translation speed (mm/sec)
```

[1] Display of this menu item depends on the test selected using the "select test" menu item.
[2] Display of this menu item and options depends on the type of device connected to your system and/or user access level.

.

```
      Max WORLD/TOOL pendant rotation speed (deg/sec)
      Time to reach max WORLD/TOOL pendant speed (sec)
      Min WORLD/TOOL pendant translation increment (mm)
      Min WORLD/TOOL pendant rotation increment (deg)

   Edit general motion specs

      Exit to robot menu
      Upper speed limit for SCALE.ACCEL (%)

   Edit S-Curve trajectory profiles

      Exit to robot menu
      Enable and edit a profile
      Disable a profile

   Edit motor stop-on-force specs[1]

      Exit
      Change motor number
      Stop-on-force integrator gain

   Edit collision detection specs

      Exit to robot menu
      User specified Cartesian obstacle #1
      User specified Cartesian obstacle #2
      User specified Cartesian obstacle #3
      User specified Cartesian obstacle #4

Load robot specifications from a disk file

Save robot specifications to a disk file

Save ALL specifications to system disk

Edit SPEC program preferences
```

`UNDO SPEC changes`[1]

`Change robot number`[2]

`Switch to external encoder specifications`[3]

Program Prompts

In the 13.0 version of SPEC, each time SPEC was executed and the Auto Backup feature was enabled it would overwrite the stored backup files. Now SPEC will prompt you to overwrite the stored backup file.

The SPEC robot configuration utility program then displays the initial screen. You may notice some differences due to your system configuration. In particular, the menu may have more or fewer items, which might be numbered differently.

```
        ***  ADEPT ROBOT SPECIFICATION PROGRAM  ***
                    (Version 13.1A1)

     Copyright (c) 1988-1998 by Adept Technology, Inc.

  Servo code version: 13.1A.2        Servo rate: 1000 Hz

        ROBOT 1: Adept 550/600/800 TT Robot Module.
                 Unrestricted access mode

                       MENU HOTKEYS

    F1-EDIT Specs     F4-LOAD file      F7-Options & Cfg
    F2-Diagnostics    F5-SAVE to file   F8-
    F3-               F6-SAVE ALL

     0 => Exit to system monitor
     1 => Change robot options and motor configuration
     2 => Perform hardware diagnostics
     3 => Edit robot specifications
     4 => Load robot specifications from a disk file
     5 => Save robot specifications to a disk file
     6 => Save ALL specifications to system disk
     7 => Switch to external encoder specifications
     8 => Edit SPEC program preferences
     9 => Edit hidden specifications

          Enter selection:
```

Figure 4-5. Initial SPEC Screen

[1] Only displayed if “Auto Backup” is enabled in the preferences menu.
[2] Multiple-robot systems only.
[3] Systems with external encoders only.

NOTE: AdeptModules Quick Setup has been partially automated for the Generation-II Z-Theta unit. The user is walked through the setup and configuration by answering YES or NO questions and pressing the ENTER key. Data input has been eliminated, with the exception of identifying the module model number.

`0 => Exit to System Monitor`

Exit the configuration program and return to the V+ system monitor.

NOTE: You can exit the program at any prompt by pressing CTRL+Z. However, this leaves all the program's global variables defined in memory. You can delete these variables by restarting the program and then exiting with the main-menu selection.

`1 => Change robot options and motor configuration`

This menu item displays the motor/joint configuration and allows changing of the configuration by the motor or joint.

```
             ***  ROBOT OPTIONS, CONFIGURATION, AND MAPPING  ***
               ROBOT 1: Enhanced 5/6-Axis PUMA Robot Module.

Robot Option Bits.  Check your device module description for bit meanings.

  Bit:   16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
  Value:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

Joint:    Joint number (1 to 6).
Axis:     Axis number as defined in device module documentation.
Motor:    Motor number (1 to 6).
Board:    Servo board number associated with a motor.  '0' means disabled.
Channel:  Channel number on servo board.

        Joint   Axis   Type       Motor   Board  Channel
          1       1   Primary       1       1       1
          2       2   Primary       2       1       2
          3       3   Primary       3       1       3
          4       4   Primary       4       1       4
          5       5   Primary       5       2       1
          6       6   Primary       6       2       2

Do you want to change this configuration (Y/N)?
```

Figure 4-6. Robot Options and Motor Configuration Screen

`2 => Perform hardware diagnostics`

This menu item displays the following read-only information for troubleshooting of a robot system. Type **Q** or **0** (zero) to exit the display. For more information refer to the ***AdeptMotion VME Developer's Guide***.

```
                    ***  Diagnostics: Robot 1  ***

                Robot power is OFF and can NOT be enabled.

                        Press Q or 0 to quit.









Motor/Board/Channel   1/B1/C1   2/B1/C2   3/B1/C3   4/B1/C4   5/B2/C1   6/B2/C2
Encoder position             0         0         0         0  Disabled  Disabled
Last zero-index              0         0         0         0
Zero-index delta             0         0         0         0
Home                       OFF       OFF       OFF       OFF
Overtravel                 OFF       OFF       OFF       OFF
Amp fault                  OFF       OFF       OFF       OFF

Amp enable                 OFF       OFF       OFF       OFF
Brake release              OFF       OFF       OFF       OFF
DAC voltage            Max DAC   Max DAC   Max DAC   Max DAC
DAC count            val is 0  val is 0  val is 0  val is 0
```

Figure 4-7. Hardware Diagnostics Screen

`3 => Edit robot specifications`

This menu item allows editing of selected robot specifications.

The following menu items are displayed at this level:

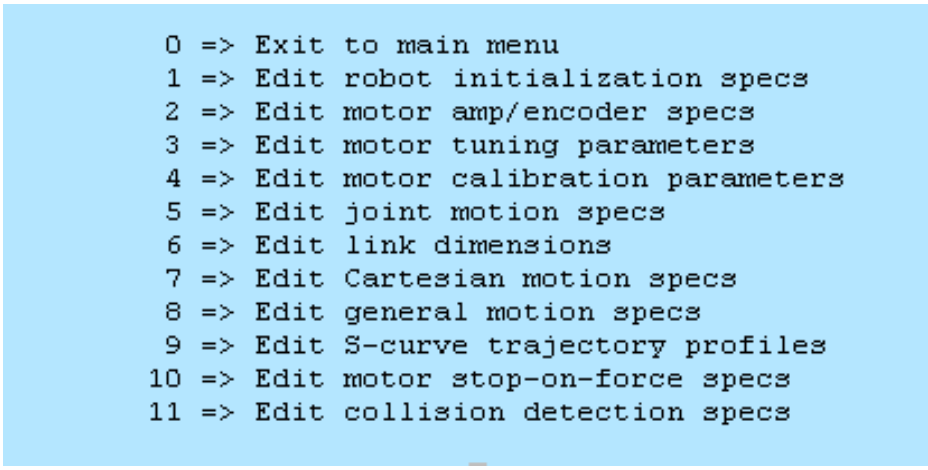


Figure 4-8. Robot Specifications Screen

These menu items allow:

- Changing the default gripper OPEN and CLOSE signals
- Changing tuning parameters on motor 4 of Adept SCARA robots. See the ***AdeptMotion VME Developer's Guide*** for details on motor tuning.
- Changing the joint limits and COARSE and FINE nulling tolerance for each joint
- Changing the percentage speed above which the SCALE.ACCEL switch has no effect
- Changing the servo mechanism ID (available only when the selected robot is an AdeptModules device)
- Editing the special arm-power-off deceleration parameter (this parameter is displayed only if robot option word bit 12 is set to enable this feature)
- Editing collision detection, including Cartesian obstacles and clearance distances. See **"Edit collision detection specs" on page 182** for details.
- Editing Programmable ESTOP delay timeout parameters

```
4 => Load robot specifications from a disk file
```

Use this menu item to load robot configuration data from a disk file created with the next menu item.

```
5 => Save robot specifications to a disk file
```

After you have changed the robot configuration data, you can store the new data in a disk data file. The data in the file can be used later to apply your specific robot configuration data to another V+ system disk.

You can save specifications in a readable ASCII text file or in an encrypted binary file. Only the editable data are included in the ASCII file; binary files contain both editable and hidden data that completely specify the device. ASCII files are adequate in many cases, since you ordinarily want to save only those specifications that you have edited. ASCII files are also adequate for any non-Adept device that is not password-protected, since there are no hidden specifications in such devices. Binary file format is typically used on password-protected robots to save the complete description of the mechanism.

If your Adept system controls more than one robot, these menu choices access the data for the specific robot you chose in the main menu.

```
6 => Save ALL specifications to system disk
```

After you have changed the configuration data for any of the items, you should store the new data on the V+ system disk. This menu item causes all the current configuration data to be written to the V+ system disk. If the changed data is not stored on the system disk, the previous configuration data will be used the next time the robot system is booted.



CAUTION: Before you modify a V+ system disk, you should make a copy of the system (with the DISKCOPY utility program).

If you normally boot your system from a floppy disk, make a copy of your V+ system disk (with the DISKCOPY utility) before you use this procedure. Save the original diskette and store the new configuration information on the copy.

The V+ system must have been booted from the actual system disk (floppy disk or hard disk) on which the new configuration information is saved.

If your Adept system controls more than one robot, this menu selection saves the data for ALL the robots (and any external encoders).

Ensure that all SPEC changes are correct before saving information to the V+ system disk.

```
7 => Edit SPEC program preference
```

This selection allows the user to configure defaults for the SPEC program.

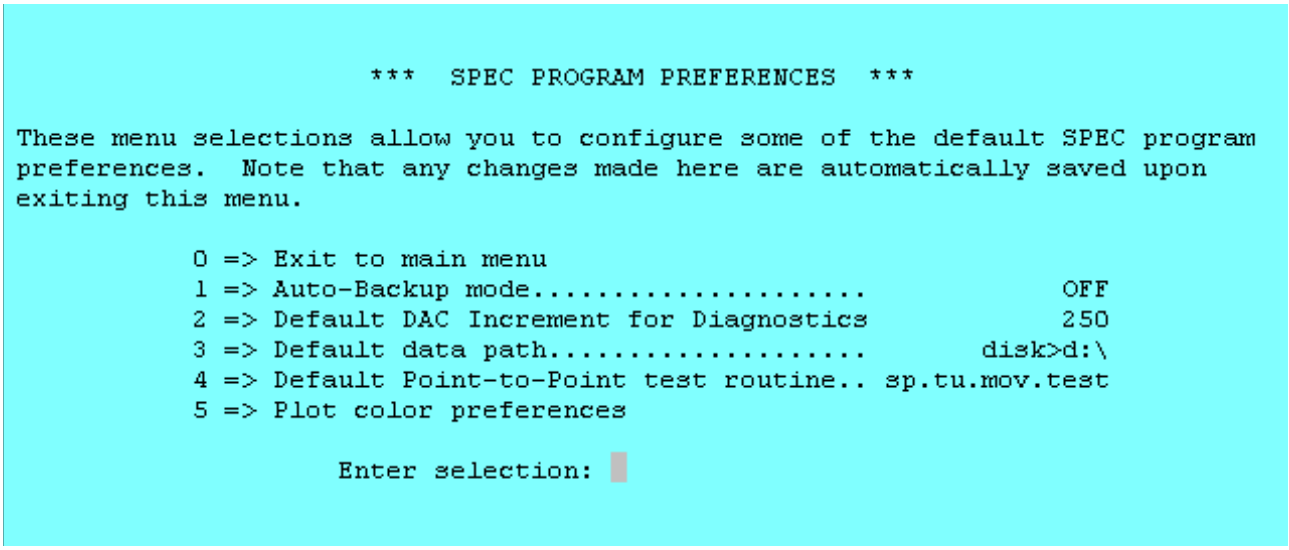


Figure 4-9. SPEC Program Preference Screen

`8=>UNDO SPEC changes`

This selection allows restoration of any SPEC changes in RAM memory to the original settings.

> **NOTE:** If any change to the SPEC utility is saved to the system disk before selecting this menu item, you must save the SPEC utility to the system disk again after selecting the menu item.

Depending on the configuration of the V+ system, the following two menu items might be available.

`Change robot number`

If your system has multiple robots, selecting this item allows you to access data for another robot.

`Switch to external encoder specifications`

If your system supports external encoders for conveyor tracking, select this item to edit the encoder parameters. See the ***AdeptMotion VME Developer's Guide*** for more information.

Descriptions of Selected Program Operations

The parameters described below can be edited after selecting "`Edit robot specifications`" in the main program menu. After changing any of these items, the data should be written to the V+ system disk so it will be used the next time the controller is turned on.

`Edit robot initialization specs`

`Hand OPEN signal`

`Hand CLOSE signal`

The numbers of the digital signals accessed by V+ for controlling the end-effector are stored on the V+ system disk. These menu items permit you to change these specifications.

The default settings for the gripper signals are listed in the user's guide for the robot. These can be changed to any of the robot I/O signals (3001 to 3008), digital output signals (numbered 1 to 8 and 33 to 512), and internal software signals (2001 to 2512). Setting a negative signal number indicates that the gripper will be activated when the signal is deasserted.

`Edit joint motion specs`

The following parameters can be adjusted for each joint of the robot:

`FINE nulling tolerance (cts)`

`COARSE nulling tolerance (cts)`

Choose these items to change the positioning tolerances used at the end of any motion that requires nulling.

NOTE: Do not modify the positioning tolerances unless you have a good understanding of how they are used by the V+ system. Refer to the descriptions of the COARSE and FINE program instructions in the ***V+ Language Reference Guide***.

`Lower joint limit (mm or deg)`

`Upper joint limit (mm or deg)`

The motion limits for the individual robot joints can be changed from their standard settings to limit robot joint travel.

`Edit general motion specs`

```
Upper speed limit for SCALE.ACCEL (%)
```

This defines the maximum program speed setting for which the SCALE.ACCEL switch has an effect. The SCALE.ACCEL or speed-scaling feature provides automatic reduction of accelerations at speed settings below this value, thus eliminating the problem of short, rapid accelerations at low program speeds. For more information on speed scaling and the SCALE.ACCEL switch, see the ***V+ Operating System User's Guide***.

```
Edit collision detection specs
```

The menu shown in **Figure 4-10** is displayed.

This menu allows the user to define parameters used to avoid collisions with static obstacles in the workspace. During operation, the path of the robot tool tip is automatically tested to ensure it will not collide with these objects under the following circumstances: when the robot is being moved in WORLD or TOOL manual control mode; when the destination of each motion is being planned; and while straight-line motions are being performed.

Each obstacle is defined by its shape, location, and size. The shape of an obstacle can be a box, a cylinder, or a sphere. The location of each obstacle is defined with respect to the base reference frame of the robot when its BASE transformation is null.



CAUTION: Because the collision detection references the *tool point* of the end-effector, which is typically at the center of the tooling, you generally should add 1/2 of the diameter of the robot's end-of-arm tooling to each surface of the actual obstacle being modeled.

```
                ***  COLLISION DETECTION SPECIFICATIONS  ***

                  0 => Exit to robot menu
                  1 => User defined Cartesian obstacle #1  0
                  2 => User defined Cartesian obstacle #2  0
                  3 => User defined Cartesian obstacle #3  0
                  4 => User defined Cartesian obstacle #4  0

                   Enter selection: 1

The following values define the shape, location, and size of a
USER DEFINED static Cartesian obstacle.  The robot's tool tip and tool
mounting flange are automatically tested to ensure they will not
collide with these objects under the follow circumstances: when the
robot is being moved in JOINT, WORLD, or TOOL manual control mode;
when the destination of the next motion is being planned; and during a
straight line motion.

The location of the obstacle is defined with respect to the base
reference frame of the robot when its BASE transformation is null.

    PARAMETER (Range -9.22E+18 to 9.22E+18)     CURRENT VALUE    NEW VALUE

Shape (0-none, 1-box, 2-cylinder, 3-sphere)                0
```

Figure 4-10. User-Defined Obstacle Definition Screen

Choose one of the user-defined options.

Enter the desired value for the object shape, position, and size. The list of shape parameters may vary depending on the type of object selected (box, cylinder, or sphere).

Figure 4-11 shows an example of a completed screen.You can repeat the previous steps for up to four objects in the robot workcell.

```
                ***  COLLISION DETECTION SPECIFICATIONS  ***

                  0 => Exit to robot menu
                  1 => User defined Cartesian obstacle #1  0
                  2 => User defined Cartesian obstacle #2  0
                  3 => User defined Cartesian obstacle #3  0
                  4 => User defined Cartesian obstacle #4  0

                   Enter selection: 1

The following values define the shape, location, and size of a
USER DEFINED static Cartesian obstacle.  The robot's tool tip and tool
mounting flange are automatically tested to ensure they will not
collide with these objects under the follow circumstances: when the
robot is being moved in JOINT, WORLD, or TOOL manual control mode;
when the destination of the next motion is being planned; and during a
straight line motion.

The location of the obstacle is defined with respect to the base
reference frame of the robot when its BASE transformation is null.

    PARAMETER (Range -9.22E+18 to 9.22E+18)     CURRENT VALUE    NEW VALUE

Shape (0-none, 1-box, 2-cylinder, 3-sphere)                0    1

    PARAMETER (Range -9.22E+18 to 9.22E+18)     CURRENT VALUE    NEW VALUE

Centroid:  X position (mm)                                 0    300
Centroid:  Y position (mm)                                 0    250
Centroid:  Z position (mm)                                 0    50
Rotation:  Yaw (deg)                                       0    0
Rotation:  Pitch (deg)                                     0    0
Rotation:  Roll (deg)                                      0    0
Size: X length (mm)                                        0    25
Size: Y length (mm)                                        0    25
Size: Z length (mm)                                        0    100
```

Figure 4-11. Sample Collision Detection Options for Box Shape

Edit Motor Calibration Parameters

The SPEC program contains an option that allows users with AdeptModules controlled by an AdeptWindows Controller to change the homing configuration so that the robot calibration procedure searches for a dedicated home sensor. (Previously, the AdeptModules devices calibrated only to an overtravel sensor.)

```
              ***  ROBOT 1/MOTOR 1 CALIBRATION  ***

                   CURRENT HOMING CONFIGURATION
          Initial search for:          current position
          Calibrate to zero index?:    yes

                           MENU HOTKEYS

                          ESC-Main Menu
F1-Init           F4-Calibration   F7-Cartesian     F10-Force
F2-Amp/Encoder    F5-Joint Specs   F8-General       F11-Collision
F3-Tuning         F6-Link Dims     F9-SCurve

     0 => Exit
     1 => Change motor number
     2 => Change homing configuration.....................    2
     3 => Speed and direction (ct/msec)...................   20
     4 => 'Motor stalled' timeout during cal (msec).......  250
     5 => Distance from current position to zero index (ct)    0
     6 => Motor position at zero index (ct)..............    0
     7 => Calibration groups.............................    1
     8 => Park position after calibration (ct)...........    0
     9 => Calibrate motor
    10 => Calibrate robot

              Enter selection:
```

Figure 4-12. The Robot/Motor Calibration Menu

Select **`Change homing configuration`** from the menu. A submenu is displayed followed by a series of questions.

```
The homing configuration is a bit field specifying various calibration options.
The following prompts set its value.  Pressing ENTER to any 'Y/N' prompt will
assume the answer 'N', leaving the current value unchanged.

Following selections specify how the motor will start calibration:

   0: Search for home sensor
   1: Search for hard stop
   2: Stay at current position
   3: Absolute calibration (requires special hardware)
   4: Calibrate via dual encoder (for Adept robots only)
   5: Search for overtravel sensor

Enter a value (ENTER to leave unchanged) :  0
```

Figure 4-13. Change Homing Configuration Submenu

To use a home sensor mounted on your AdeptModules axis for calibration, select **`Search for home sensor`** from the submenu and press ENTER. To cancel the option, simply press the ENTER key.

After selecting the option, you are asked to disable (or enable) the "calibrate to zero index" feature. To toggle the setting press `y` and then press ENTER. To use the current setting, enter `n` and then press ENTER, or simply press ENTER to accept the default value (`n`).

After answering this question, you are asked to enable (or disable) the "fine search toward home" switch. To toggle the setting for fine search toward home, enter `y`. To use the current setting, press `n`, or simply press ENTER to accept the default value (`n`). After you answer this question, the screen is cleared and the Robot/Motor Calibration menu is displayed.

`Edit motor amp/encoder specs`

The SPEC program allows you to specify a manual mode envelope error limit. This option allows you to configure a smaller envelope error (that is, how far the motor is allowed to lag behind the commanded position) for motions made with the manual control pendant. A small value for this parameter results in faster shutdown of a jammed robot. This parameter is related to the Soft Envelope Error and Hard Envelope Error as follows:

Man Env Err ≤ Soft Env Err ≤ Hard Env Err

To change the manual mode envelope error limit, select `*Manual mode envelope error* limit` from the menu and press ENTER. You are prompted for the allowable manual mode position error count. Type the new value and press ENTER to accept, or simply press ENTER to cancel the option.

`Edit motor tuning parameters`

The SPEC program contains options that allow you to configure both the acceleration and deceleration percentages for motor tuning tests, and to specify a custom point-to-point motion routine. To access these options select the following item from the Robot Servo Tuning Submenu:

```
Select test..Taught Pt
```

The following material describes use of these options.

Motion Acceleration and Deceleration Percentages

This feature allows the user to specify unique acceleration and deceleration percentages for the point-to-point motion test during motor tuning. This can be useful for high-inertia robots that may require different acceleration and deceleration values.

From the Motor Tuning Test Configuration menu, select either **Move acceleration** (for setting the acceleration percentage) or **Move deceleration** (for setting the deceleration percentage). Type the desired value and press ENTER to accept it, or simply press ENTER to cancel the option and return to the Motor Tuning Test Configuration menu.

Options for the Frequency Response Tests

The SPEC program contains options for the frequency response tests that allow you to specify the number of frequency bands to test, the start amplitude, and the stop amplitude. To access these options:

1. Load and execute the SPEC utility. Refer to **page 170** for details.
2. From the main menu, choose the following menu options to access the Motor Tuning Test Configuration menu:

   Edit robot specifications → Edit motor tuning parameters → Select test...

   The Motor Tuning Test Configuration menu is displayed.

   NOTE: The options shown on this menu change depending on what test type is selected.

```
            ***  MOTOR 1 TUNING TEST CONFIGURATION  ***

 0 => Exit
 1 => Square wave
 2 => Move between taught points
 3 => Plant+controller frequency response..................... SELECTED
 4 =>   Number of frequency bands to test.....................        1
 5 =>   Number of frequencies to test.........................        5
 6 =>   Low frequency (Hz)....................................        1
 7 =>   High frequency (Hz)...................................       20
 8 =>   Start amplitude (encoder counts)......................        0
 9 =>   Stop amplitude (encoder counts).......................      100
10 => Plant-only frequency response (Closed-loop excitation)
11 => Auto-tune to refine existing feedback gains
12 => Auto-tune to find feedforward gains

               Enter selection:
```

Figure 4-14. Motor Tuning Test Configuration Menu
(Frequency Response Test Selected)

3. To access the frequency response tests, select one of the following menu options:
   - Plant+controller frequency response...
   - Plant-only frequency response...

   You can now make adjustments to the settings for the frequency response items shown in **Table 4-4**

Table 4-4. Frequency Response Test Menu Items

Menu Item	**Input Range**	**Description**
Number of frequency bands to test	1 to 10	Specifies the number of frequency bands to test.
Number of frequencies to test	2 to 1000	Specifies the number of test iterations for each frequency band.
Low frequency (Hz)	0.1 to 20	Specifies the lowest input frequency (in Hz) used for the test.

Table 4-4. Frequency Response Test Menu Items (Continued)

Menu Item	Input Range	Description
High frequency (Hz)	1 to 500	Specifies the highest input frequency (in Hz) used for the test.
Start amplitude (encoder counts)	0 to 100000	Specifies the start and stop wave amplitude in encoder counts. Amplitude is linearly spaced between the Start and Stop values. To hold the amplitude constant, set the Start amplitude value to 0.
Stop amplitude (encoder counts)	0.1 to 100000	

Adding a Description to a SPEC-Generated Data File

Whenever a SPEC-generated data file is saved, you have the option of entering a description of the file contents. When saving a data file, you will receive a prompt similar to the following:

```
Do you want to include a descriptive message regarding this Freq Resp data
file (Y/N)?  Y


Enter a descriptive message regarding the contents of this Freq Resp data
file.  You can add as many lines as you like.  To terminate text
input, type a single "." on the last line.

> This plot represents a test of the features on the frequency response
> test menu.
> .
```

Figure 4-15. Adding a Description to a SPEC Data File

To add a description to a file, type y, then ENTER, and then type the message at the “>” prompt. You can add additional lines (including blank lines) by pressing ENTER at the end of each line. To end the message, type a period character (.) by itself on the last line and then press the ENTER key. The message is automatically included in the header portion of the data file.

Static Friction Feedforward Compensation

SPEC includes parameters for setting up a motion-control feature called Static Friction Feedforward Compensation. This feature is designed to minimize tracking errors by providing a corrective torque proportional to the amount of friction in the system. This allows a motion to start more quickly, particularly in the case of a sticky drive train.

> **NOTE:** Modifying these parameters when moving in square-wave drive mode has no effect since the commanded velocity is 0.

To access this option:

1. Load and execute the SPEC utility. See the ***Instructions for Adept Utility Programs*** for details.
2. From the main menu, choose the following menu options to access the Motor Tuning Test Configuration menu:

   Edit robot specifications → Edit motor tuning parameters → Static friction ffwd...

The Static Friction Feedforward menu is displayed.

```
                    Static Friction FFWD Parameters

The static friction feedforward parameters can minimize tracking errors by
providing a corrective torque proportional to the amount of friction in the
system.  This allows a motion to start quicker, particularly in the case of
a sticky drive train.  Modifying these parameters when moving in square-wave
drive mode will have no effect since the commanded velocity is 0.

                          0 => Exit
                          1 => Max trigger.......  0
                          2 => Min trigger.......  0
                          3 => Max positive value  0
                          4 => Max negative value  0
                          5 => Time constant.....  0

                 Enter selection:
```

Figure 4-16. Static Friction Feedforward Menu

You can make adjustments to the settings for the static friction feedforward items shown in **Table 4-5**.

Table 4-5. Static Friction Feedforward Menu Items

Menu Item	Input Range	Description
Max trigger	0 to 9.22E+18	Specifies the maximum commanded velocity value (cnts/msec) for ramping the static friction compensation torque up to the maximum positive or maximum negative value, depending on the direction of motion of the joint. Once the velocity reaches the max trigger value, the static friction compensation output is clipped at the max positive or max negative value. To disable static friction feedforward compensation, set this value to 0.
Min trigger	0 to 9.22E+18	Specifies the minimum commanded velocity value (cnts/msec) that initiates static friction feedforward compensation.
Max positive value	0 to *	Specifies the maximum positive or maximum negative amount of torque that can be applied to the commanded velocity when the joint is moving in the corresponding direction. In most cases the maximum positive and maximum negative values should be identical.
Max negative value	0 to *	
Time constant	–9.22E+18 to 9.22E+18	This integer value allows the servo software to anticipate or delay the effect of the static friction compensation torque on the commanded velocity. If the value is positive, compensation will lead the servo setpoint. If the value is negative, compensation is delayed.
* The displayed maximum value equals the maximum DAC output multiplied by 256 for the selected motor.		

Navigation Hotkeys

Many of the SPEC menus now include hotkeys that provide quick navigation to specific functions. SPEC program menus are dynamic and are different depending on system configurations. Hotkeys are static, providing quick navigation in SPEC for all systems regardless of configuration. An example of a menu that contains hotkeys is shown in **Figure 4-17**.

```
                  ***  ROBOT SPECIFICATIONS  ***

                ROBOT 1: X/Y/Z/Theta Robot Module.

                           MENU HOTKEYS

                          ESC-Main Menu
F1-Init           F4-Calibration   F7-Cartesian     F10-Force
F2-Amp/Encoder    F5-Joint Specs   F8-General       F11-Collision
F3-Tuning         F6-Link Dims     F9-SCurve

              0 => Exit to main menu
              1 => Edit robot initialization specs
              2 => Edit motor amp/encoder specs
              3 => Edit motor tuning parameters
              4 => Edit motor calibration parameters
              5 => Edit joint motion specs
              6 => Edit link dimensions
              7 => Edit Cartesian motion specs
              8 => Edit general motion specs
              9 => Edit S-curve trajectory profiles
             10 => Edit motor stop-on-force specs
             11 => Edit collision detection specs

              Enter selection:
```

Figure 4-17. Sample Menu with Hotkeys

Pressing a function key advances the SPEC program directly to the menu associated with that function key.

Plot Color Preferences

This feature allows you to configure the Plot screen, by changing the colors of all the graphics within the plot window (see **Figure 4-18**).

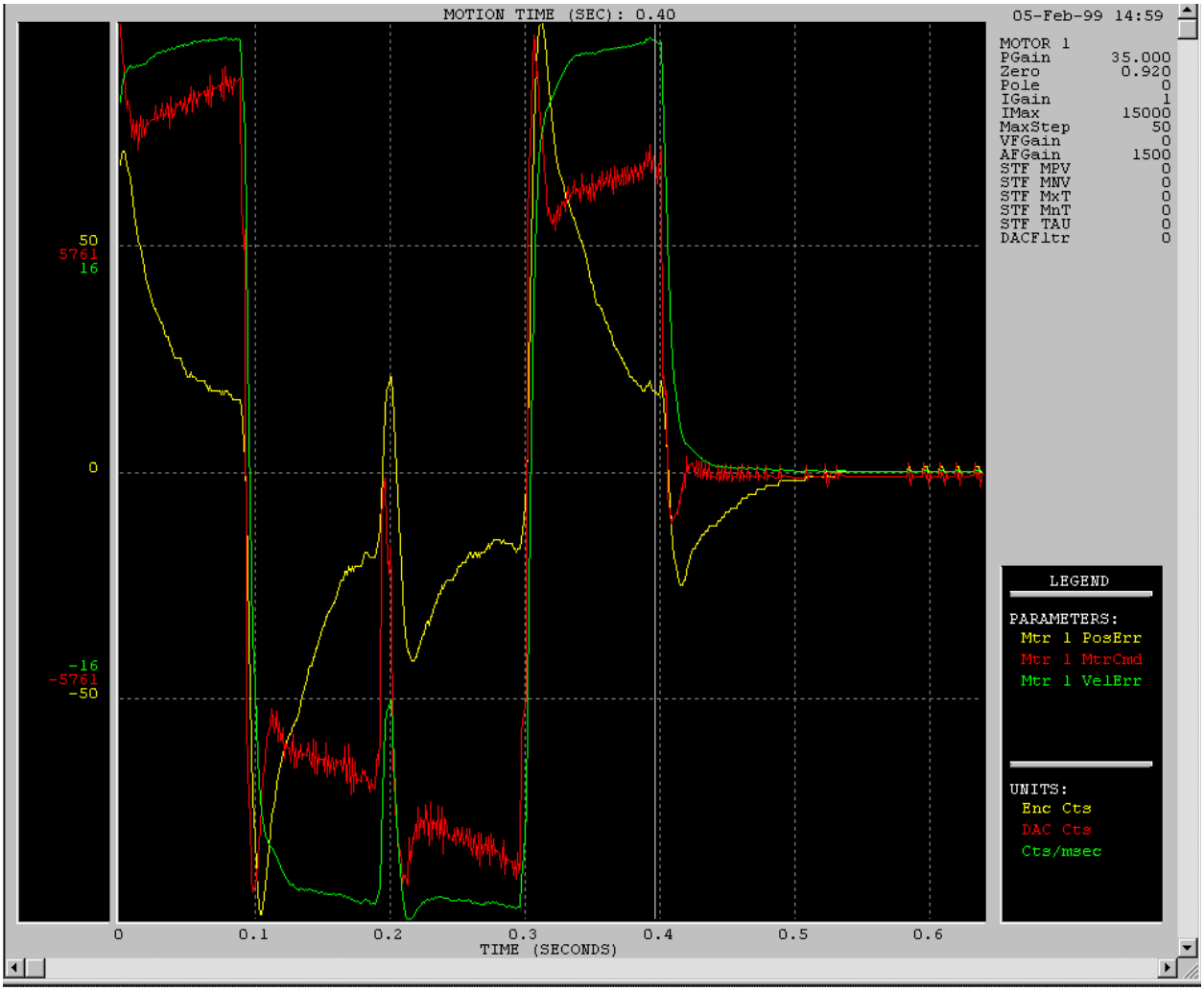


Figure 4-18. Sample Plot Color Preferences

To access this option:

1. Load and execute the SPEC utility. Refer to **page 170** for details.
2. From the main menu, choose the following menu options to access the Plot Color Preferences Configuration menu:

   SPEC program preferences ➔ Plot Color Preferences...

   The Plot Color Preferences menu is displayed (**Figure 4-19**).

```
                    ***  PLOT COLOR PREFERENCES  ***

The colors used to display various performance plots can be changed to
suit your needs.

                 0 => Exit to Preferences menu
                 1 => Plot background color.....    Black
                 2 => Plot trace #1 color.......   Yellow
                 3 => Plot trace #2 color.......      Red
                 4 => Plot trace #3 color.......    Green
                 5 => Plot trace #4 color.......   Orange
                 6 => Plot trace #5 color..... Lt. Blue
                 7 => Plot end-of-motion marker color     Pink
                 8 => Plot grid color.............     Gray
                 9 => Plot axis color.............    White
                10 => Plot text color.............    Black

                    Enter selection:
```

Figure 4-19. Sample Plot Color Preferences Menu

3. Select a preference to change. A list of available colors for that preference is displayed (see **Figure 4-20**).
4. Enter the number of the desired color and press the ENTER key.

```
                0 => Exit to Preferences menu
                1 => Plot background color.....    Black
                2 => Plot trace #1 color.......   Yellow
                3 => Plot trace #2 color.......      Red
                4 => Plot trace #3 color.......    Green
                5 => Plot trace #4 color.......   Orange
                6 => Plot trace #5 color..... Lt. Blue
                7 => Plot end-of-motion marker color     Pink
                8 => Plot grid color...........     Gray
                9 => Plot axis color...........    White
               10 => Plot text color...........    Black

                  Enter selection: 1
The color choices are:
                             1 => Black
                             2 => Dk. Gray
                             3 => Blue
                             4 => Blue Gray
                             5 => Lt. Blue
                             6 => Green
                             7 => Dk. Green
                             8 => Yellow
                             9 => Orange
                            10 => Red
                            11 => Maroon
                            12 => Pink
                            13 => Gray
                            14 => Lt. Gray
                            15 => White

The current background color is: Black

Enter new background color or press ENTER for no change:
```

Figure 4-20. Sample Plot Color Preferences With List of Colors

Programmable ESTOP Delay-time Parameters

Three programmable ESTOP delay-time parameters have been added to the Robot Initialization Specification menu. This allows you to select a time delay after an ESTOP is detected before power is removed from the motors of the robot. This time delay gives V+ time to decelerate and stop the robot using its motor power, reducing the mechanical jerk on the mechanism.

> **NOTE:** The Programmable ESTOP feature is available on systems that have an AWC Gen-II processor, except for those systems that use the Adept Manual Mode Safety Package (CE Category 3).

Approximately 1000msec after an ESTOP signal is detected, power is removed from the motors of the robot. Setting the Programmable ESTOP delay-time to stop the motors within that 1000msec gives the robot time to stop, settle, and apply the brakes before power is removed.

If the Programmable ESTOP delay-time parameters are set to 0 (zero), V+ removes power instantly when the ESTOP is activated. The robot will stop based on mechanical resistance and any available brakes.

To access this option:

1. Load and execute the SPEC utility. Refer to **page 170** for details.
2. From the main menu, choose the following menu option:
   Edit Robot Specifications → Robot initialization specifications → ...

   The Robot initialization specification menu is displayed.
3. From the Robot initialization specification menu select one of the three Programmable ESTOP delay-time parameters.

```
15 => Manual mode E-STOP shutdown timeout (sec)

16 => Auto mode E-STOP shutdown timeout (sec)

17 => MCP Enable E-STOP shutdown timeout (sec)
```

Custom Point-to-Point Motion Routine

This feature allows a custom point-to-point motion routine that will execute in place of the standard routine in SPEC. (The standard routine is protected and cannot be edited.) If you want to do something special with the robot and analyze data collected during the motion, write a program, and then specify the name in the SPEC program. SPEC will verify that the custom routine exists (in RAM) and is executable. At execution time, SPEC will call the custom program instead of the standard routine.

> **NOTE:** You can also specify a custom point-to-point motion routine as one of the SPEC program preferences.
>
> **NOTE:** The custom routine must be in RAM when the name is specified to the SPEC utility. SPEC will verify that the custom routine exists and is executable (at least as far as V+ is concerned).

Select the **`Pt-to-Pt motion routine`** from the Motor Tuning Test Configuration menu. You are prompted for the name of the new routine. Type the name of the custom routine and press ENTER. (Enter a space and press ENTER to revert to the standard routine, or press ENTER to cancel the option and return to the Motor Tuning Test Configuration menu.)

When using a custom motion routine, please note:

- The routine must have this calling sequence (see below for details):

```
routine(plot, ctl[ ], $err)
```

- The routine must be loaded into RAM.

Program Listing for sp.tu.mov.test( )

The program listing for the standard routine `sp.tu.mov.test( )` is provided below to show how the routine works. This will allow creation of custom motion routines that use SPEC to collect motion data.

> **NOTE:** The standard routine is protected. This listing is supplied so that developers can reference the standard program as an example for creating custom motion routines.

```
.PROGRAM sp.tu.mov.test(plot, ctl[], $err)

; ABSTRACT:  Performs programmed motion for taught-point tuning test.

;       Assumes precision points #sp.tu.pt[0] and #sp.tu.pt[1] have
;       been defined. These variables contain the two precision points
;       taught for the point-to-point test.  Custom routines may also
;       use two transforms that are taught at the same time:
;       sp.tu.loc[0] and sp.tu.loc[1].
;
;       This is the standard point-to-point motion program.  Custom
;       motion programs can be used to perform special moves for
;       servo tuning analysis.  Any custom program MUST have the
;       same calling sequence as this program.
;
;       In order to take advantage of the data collection and plotting
;       features of the SPEC program, note the CALLs to the following
;       programs in the code below:
;
;               CALL sp.tu.config(time)         ;Specfies the amount of data
;                                               ;collection time in seconds.
;              CALL bf.enable(mode, sync)      ;Enables data collection buffer.
;                  The values for "mode" and "sync" are detailed below:
;                      mode 0 (default): Stop data collection
;                           1: Trigger re-enabled after collection
;                           2: Single sweep
;                      sync    Trigger synchronization
;                           0 (default): Trigger now
;                           1: Trigger to internal trajectory generator
;
;               CALL sp.tu.plot(time, $err)     ;Plots the results.  Note the
;                                               ;"time" argument is internally
;                                               ;calculated.
;
;               CALL sp.tu.dump(lun)            ;Dumps data to the specified
;                                               ;Logical Unit Number (lun).
;                                               ;LUN = 4 is the terminal or
;                                               ;MONITOR window.
;
; The data collected will be that configured in the data-collection
; menu.
;
```

```
; INPUT PARM:   plot    Flag indicating whether to plot results
;               ctl[]   Array of test control values.  Relevant values
;                       for point-to-point motion test are:
;                       ctl[sp.tu.move.spd] = Move speed (% motor max)
;                       ctl[sp.tu.move.acc] = Acceleration (1-100%)
;                       ctl[sp.tu.move.dec] = Deceleration (1-100%)
;                       ctl[sp.tu.move.prf] = Accel profile
;
; OUTPUT PARM:  $err    Error string.  If non-empty, then an error
;                       occurred that will be reported to the user.
;
; SIDE EFFECTS: None
;
; Copyright (c) 1994-1998 by Adept Technology, Inc.

        AUTO DOUBLE time

        CALL sp.tu.config(10)           ;Configure 10 sec of data collection

; Set up speed and acceleration

        SPEED ctl[sp.tu.move.spd] ALWAYS
        ACCEL (ctl[sp.tu.move.prf]) ctl[sp.tu.move.acc], ctl[sp.tu.move.dec]

; Move to 1st point a little slower than the test motions if the test
; motion speed is greater than or equal to 50%.

        IF ctl[sp.tu.move.spd] >= 50 THEN
            SPEED ctl[sp.tu.move.spd]/2         ;Move at 1/2 speed
        END

        COARSE                          ;Don't bother to settle accurately
        MOVE #sp.tu.pt[0]               ;Move to Start point
        DELAY 0.2                       ;Allow some time for settling
        BREAK                           ;Complete settling time

        FINE                            ;Perform motion accurately
        MOVE #sp.tu.pt[1]               ;Initiate motion
        WAIT                            ;Allow motion to start

        time = TIMER(-3)                ;Reset cycle timer
        CALL bf.enable(1)               ;Enable buffer: wrap-around, start now

        BREAK                           ;Complete motion
        MOVE #sp.tu.pt[0]               ;Return to start position
        BREAK                           ;Complete motion
        time = TIMER(-3)-time           ;Record cycle time

        WAIT.EVENT , 0.25               ;Collect for a while longer
        CALL bf.enable()                ;Turn off buffer

        IF plot THEN
            CALL sp.tu.plot(time, $err) ;Plot results
        ELSE
            CALL sp.tu.dump(4)          ;Dump data buffer to monitor
        END

        RETURN
```

5 Advanced Servo Library

Introduction 200
Changes to the Advanced Servo Library 200
New Opcodes 201
Corrected Documentation Errors 202
New ASL Routines 204

Introduction

This chapter describes the changes to the Advanced Servo Library (ASL) for V+ version 13.1. For more information about the Advanced Servo Library refer to the ***Advanced Servo Library Reference Guide***.

Changes to the Advanced Servo Library

The following changes were made to the Advanced Servo Library:

- Updated the ASL code for compatibility with V+ version 13.1
- Added support for the Static Friction Feedforward opcodes; see **page 201**
- Corrected documentation errors; see **page 202**
- Added new ASL routines; see **page 204**

New Opcodes

The opcodes listed in **Table 5-1** were added to support the V+ version 13.1 Static Friction Feedforward opcodes.

> **NOTE:** See the description of **sl.read( )** in the ***Advanced Servo Library Reference Guide*** for descriptions of existing opcodes.

Table 5-1. New SPEC Servo Configuration Parameter Names

Parameter Name (Opcode)	**Data Returned**	**Units**
sl.stftau	Static friction FFWD time constant	None
sl.stfumi	Static friction FFWD minimum trigger	cnts/msec
sl.stfuma	Static friction FFWD maximum trigger	cnts/msec
sl.stfmpv	Static friction FFWD maximum positive value	DAC counts
sl.stfmnv	Static friction FFWD maximum negative value	DAC counts
sl.satmax	Maximum time in torque saturation (at maximum DAC)	seconds
sl.pktrq	Peak torque for specified motor	DAC counts

Corrected Documentation Errors

Calling Sequence

sl.free `(mtr.mask, enable, error)`

Function

Places the desired motor(s) into FREE mode if FREE mode is allowed for the specified motor(s).

Usage Considerations

This routine automatically places the required channel into and out of ASYNC mode.

The "FREE mode digital signal" defined in the SPEC data is accessed by this routine.

If disabling FREE mode, the motor(s) is(are) automatically placed back into SYNC mode (position servo mode) and the robot is DETACHed and reATTACHed to the calling task.

Input Parameters

`mtr.mask` Bit mask indicating which joint(s) to free up (one bit per motor; motor #1 is the LSB):

Motor	Bit	Binary Bit Mask	Hex Bit Mask
1	1	^B1	^H1
2	2	^B10	^H2
3	3	^B100	^H3
4	4	^B1000	^H4
5	5	^B10000	^H5
6	6	^B100000	^H20
etc.			

`enable` If TRUE, enable FREE mode. If FALSE, disable FREE mode and reattach the robot.

Output Parameters

`error` Standard success/error code. If FREE mode is not allowed (i.e., it is disabled in the SPEC data) for a requested motor, error code –609 (`*Illegal joint number*`) is returned. An informational message is displayed on the monitor if the global flag **sl.adv.diag** is nonzero.

Details



WARNING: Placing a gravity-loaded joint into FREE mode could cause the joint to drop, causing damage to the payload or the workspace.

FREE mode releases servo control of the motor by either disabling the Drive Enable signal for the specified channel(s), or by commanding zero torque. The specific method depends on the drive configuration. This mode is intended to allow the mechanism to be easily moved by hand during setup or to create a compliant robot joint or joints to ease part insertion.

See the example routine "ex.free( ) - FREE Mode Example" in the ***Advanced Servo Library Reference Guide*** for more details.

Related Routine

sl.c.free

New ASL Routines

This section describes the functions and calling sequences of new routines in the Advanced Servo Library. These routines may be called by application software written by a system customizer. See the ***Advanced Servo Library Reference Guide*** for descriptions of existing routines.

The descriptions of the routines are presented in alphabetical order, with each routine starting on a separate page. The description for each routine contains the following sections, as applicable.

Calling Sequence

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

> **NOTE:** The variable names used for the routine parameters are for explanation purposes only. Your application program can use any variable names you want when calling the routine. Although some CALL instructions are shown continued on a second line, instructions in an actual V+ program must always be completed on one line.

Function

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

Usage Considerations

This section is used to point out any special considerations associated with use of the routine.

Input Parameters

Each of the input parameters in the calling sequence is described in detail. For any parameter that has a restriction on its acceptable values, or is required to be defined as a double-precision variable, the restriction/requirement is specified.

Output Parameters

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

Details

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

Related Routines

Other library routines with related functions are listed.

Calling Sequence

```
CALL sl.rw.ready (mot.num, write, ready, stt)
```

Function

Reads/writes READY position for a single joint.

Input Parameters

`mot.num`	Read/write data for this joint number.
`write`	If TRUE (nonzero), write value. If FALSE (zero), read value.
`ready`	If `write` is TRUE, new real value for joint READY position.

Output Parameters

`ready`	If `write` is FALSE, current value of joint READY position.
`stt`	Success/failure status: 0 = Success <0 = Failure (V+ error code)

Details

This routine reads or writes the READY position for the specified joint of the current robot. This is the position the joint moves to when the V+ READY instruction is executed.

Calling Sequence

```
CALL sl.rw.chgmonspd (write, change.rate, stt)
```

Function

Reads or writes the "maximum rate of change of Monitor speed" parameter for the selected robot.

Input Parameters

write	If TRUE (nonzero), write values. If FALSE (zero), read values.
`change.rate`	If `write` is TRUE, the value represents the new rate of change of monitor speed.

Output Parameters

`change rate`	If `write` is FALSE, the value represent the current rate of change of monitor speed.
`stt`	Success/failure status: 0 = Success <0 = Failure (V^+ error code)

Details

This routine reads or writes the maximum rate of change of monitor speed of the current selected robot.

Calling Sequence

```
CALL sl.rw.estop.dly (write, auto.tmo, man.tmo,
                                          mcp.tmo, stt)
```

Function

Reads/writes the programmable ESTOP delay time parameters.

> **NOTE:** These values are *system* specific. That is, they are used for all the robots in the system.

Input Parameters

`write` If TRUE (nonzero), write values.
If FALSE (zero), read values.

`auto.tmo` (Optional) If `write` is TRUE, this is the new value of the ESTOP delay time when in AUTO mode.

`man.tmo` (Optional) If `write` is TRUE, this is the new value of the ESTOP delay time when in MANUAL mode.

`mcp.tmo` (Optional) If `write` is TRUE, this is the new value of the ESTOP delay time when the MCP Enable switch is released.

Output Parameters

`auto.tmo` (Optional) If `write` is FALSE, this is the current value of the ESTOP delay time when in AUTO mode.

`man.tmo` (Optional) If `write` is FALSE, this is the current value of the ESTOP delay time when in MANUAL mode.

`mcp.tmo` (Optional) If `write` is FALSE, this is the current value of the ESTOP delay time when the MCP Enable switch is released.

Details

This routine reads or writes the programmable ESTOP delay-time parameters for the system. All robots within the system will use the new ESTOP delay-time parameters.

Index

A

a.config_c (program) 134
a.spec (program) 168
Abbreviations 18
ABS_POSITION (FSET argument) 47
AdeptModules, dedicated home sensor for calibration 185
AdeptMotion VME, configuration 168
AdeptNet 59
ADW system switch 34
AMOVE program instruction 36
Analog input, configuring 149
Analog output, configuring 149
Assignment
  location variables 85
Automatic Learning details 108
Axis control 36

B

BORDER (FSET argument) 47
Branching 73
BYTE_LENGTH (FSET argument) 57

C

Calibration
  camera 122
    training sensitivity to 122
  position for training 122
CE 18
Change Homing Configuration 186
Changed Utility Disk files 130
Changes
  to utility programs 130
  to V+ commands and keywords 20
  to V+ keywords 31
  to Vision keywords 116
Collision detection, defining obstacles 182
Colors, graphics 50
Commands and keywords 19
Communications testing 82
CONFIG_C.V2 (disk file) 134
Configuration
  controller 134
  motion device 168
Conventions 15
Coordinates
  world 85
Correlation template 117, 118, 119
Creating
  location variables 61, 85
Ctrl+X, use with XSTEP 100
CURSOR (FSET argument) 47
Custom point-to-point motion routine 196
Customer service assistance
  phone numbers 14

D

DDCMP status 82
Debugging, XSTEP 100
Dedicated home sensor for calibrating AdeptModules 185
Default
  disk drive selection 152
  motion speed 152
  speed 152
Defining
  location variables 61, 85
Defining obstacles 182
Digital I/O
  input configuration 149
  OPEN, CLOSE signals 181
Directory
  memory 38, 80
  modules 80
  programs 38, 80

DIRECTORY monitor command 38
Disk drive, default 134
Disk driver task 27
DISPLAY (FSET argument) 48
DTR (FSET argument) 57

E

Enabling software options 75
EVENT (FSET argument) 48
EXECUTE monitor command 40
Executing programs 40, 87, 100
Expression, logical 73

F

Finder, see ObjectFinder
FLOW (FSET argument) 57
FLUSH (FSET argument) 57
FONT_HDR (FSET argument) 49
Frame size, vision system 154
FSET
  attributes for AdeptNet 58
  graphics window attributes 47
  program instruction 45
  serial-line attributes 57

G

Graphics 59
  configuration 134
  instructions 45
  system configuration 152
  window
    attributes 45
    modification 45

H

H_ARROWINC (FSET argument) 49
H_HANDLE (FSET argument) 49
H_RANGE (FSET argument) 49
H_SCROLL (FSET argument) 50
HERE monitor command 61
High-speed signals, configuration 134
High-speed trigger 149

I

ID monitor command 63
ID real-valued function 67
IF program instruction 73
IGNORE (FSET argument) 50
Input parameters using virtual cameras 125
INSTALL monitor command 75
Installation of new V+ system 141
Installing software options 75
Installing V+ 19, 141
Introduction 11

K

Kermit task 27
Keyboard conventions 17
Keyword changes 106

L

Label, program step 73
LOAD monitor command 77
Loading
  locations 77
  programs 77
  real variables 77
  string variables 77
Location
  definition 61, 85
  modification 85
  variable 61, 85
Location loading 77
Location variable
  creating from keyboard 85
Logical expression 73
LUT (FSET argument) 50

M

MARGINS (FSET argument) 50
MDIRECTORY monitor command 80
Memory
  allocating for vision 154
  directory 38, 80
MENU (FSET argument) 51
Messages, system 23
Modification
  location 85
Modification of graphics window 45
Module
  creation 77
  directory 80
  loading 77
Modules

displaying modules in memory 80
Monitor commands
DIRECTORY 38
EXECUTE 40
HERE 61
ID 63
INSTALL 75
LOAD 77
MDIRECTORY 80
NET 82
PING 82
POINT 85
PRIME 87
TIME 90
XSTEP 100
Monitor speed, setting default 152
Monitor task 27
Motion acceleration and deceleration percentages 187
Motion device configuration 168
MULTIDROP (FSET argument) 57

N

NET monitor command 82
Network configuration 134
Network/DDCMP task 27
New functionality, V+ 22
NODISPLAY (FSET argument) 48
Notes, Cautions, and Warnings, meaning of 15
NOUPDATE (FSET argument) 55
NVRAM 140

O

ObjectFinder changes 106
Obstacles, defining 182
Optical Character Recognition (OCR) 117
Optional V+ Developer's Manuals 14
Other Adept Product Manuals 13
Overview, ObjectFinder changes 107

P

PARITY (FSET argument) 57
Pendant task 27
PING monitor command 82
Pitch (P) orientation angle 62, 86
POINT monitor command 85
POINTER (FSET argument) 51
POSITION (FSET argument) 52
Position latch 149
Precision point 85
PRIME monitor command 87
Priority, task 153
Program
execution 40, 87, 100
label 73
listing for sp.cu.move.tst( ) 197
loading 77
module 77, 80
placing on stack 87
protected 38
read-only 38
task configuration 134
Program execution
initiating 40, 87, 100
Program instructions
AMOVE 36
FSET 45
IF...GOTO 73
TIME 92
VSHOW.MODEL 117
VTRAIN.FINDER 122
Protected program 38
Protocols, setting serial line 56
PULLDOWN (FSET argument) 52

R

Read-only program 38
Real variable
loading 77
Recording current robot location 61
Related Manuals 12
Relational test 73
Robot
configuration 168
number 70, 71
Roll (R) orientation angle 62, 86

S

Sample collision detection options for box shape 184
SELECT (FSET argument) 53
Selecting, Choosing, Pressing, and Performing 17
Serial I/O

configuration 134, 150
DDCMP 82
task 27
Serial line
configuring with FSET 56
Servo board configuration 151
Servo communication task 27
SHOW (FSET argument) 53
Signals, external trigger 134
Single-step execution 100
SIZE (FSET argument) 53
Software options installing 75
SPEC.V2 (disk file) 168
SPECIAL (FSET argument) 53
SPEED (FSET argument) 58
STACK (FSET argument) 54
Standard manuals 12
Status
DDCMP communications 82
Step label 73
STOP_BITS (FSET argument) 57
String
function 96, 98
variable loading 77
Summary
of AdeptVision changes 106
of changes 19
of utility changes 130
of V+ changes 22
Support
phone numbers 14
System
configuration 139
date 90
identification 63
options, displaying 140
priorities 28
tasks 27, 153
System memory
displaying modules in 80
listing contents 38
loading file into 77

T

Task
configuration 134, 153
system 27, 28
Teaching locations 61
TERMINAL (FSET argument) 54
Terminal/graphics tasks 27
Test, relational 73
Testing communications 82
$TIME string function 96, 98
TIME
monitor command 90
program instruction 92
real-valued function 94
TITLE (FSET argument) 55
Trajectory generator task 27
Transformation 85
Typographic Conventions 16

U

UPDATE (FSET argument) 55
User Defined Cartesian obstacles 182
User Defined Obstacle Instruction
Screen 183
User tasks, configuring 153
Using 56
Using FSET
with NFS and TCP 58
with serial lines 56
with windows 45

V

V+ interpreter, enabling/disabling 148
V+ system tasks 27
V_ARROWINC (FSET argument) 55
V_HANDLE (FSET argument) 56
V_RANGE (FSET argument) 56
V_SCROLL (FSET argument) 56
Variables
location 85
defining 61, 85
Variables, loading 77
Vision
analysis task 27
communication task 27
memory allocation 154
trigger 149
VSHOW.MODEL program
instruction 117
VTRAIN.FINDER program
instructions 122

W

Window, graphics

attributes **45**
modification **45**
World coordinates **85**

X

XSTEP monitor command **100**

Y

Yaw (Y) orientation angle **62**, **86**

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.

If you prefer to submit your comments by e-mail, you can send them to:

tpubs@aol.com

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

00963-11120, Rev. A