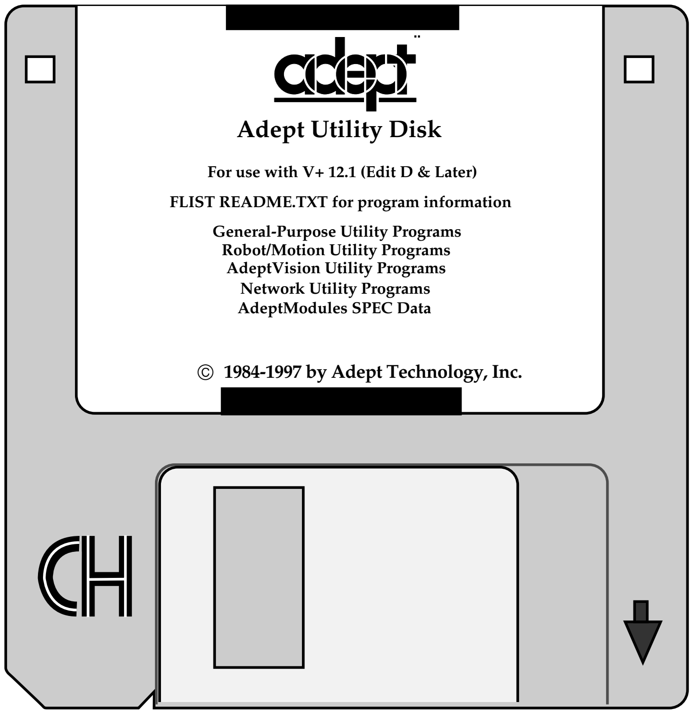
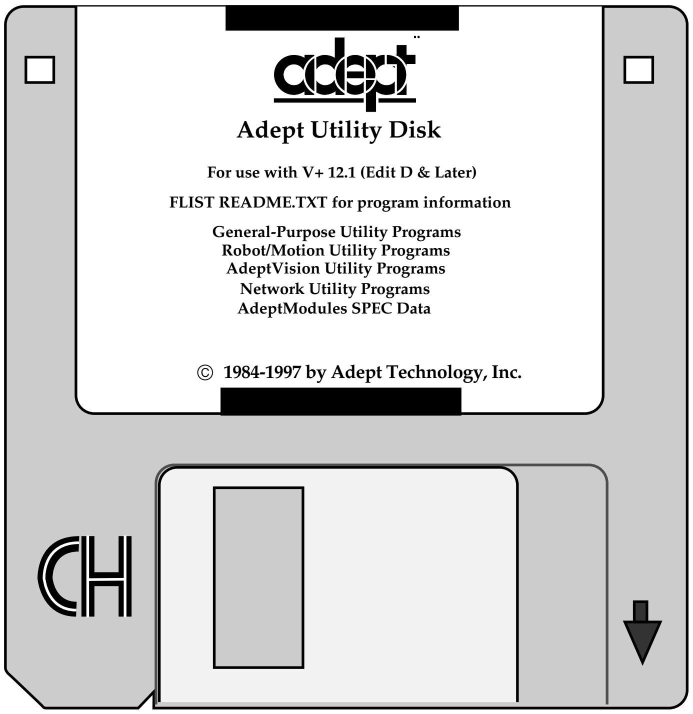
Instructions for Adept Utility Programs

Version 12.1


adept
Adept Utility Disk
For use with V+ 12.1 (Edit D & Later)
FLIST README.TXT for program information
General-Purpose Utility Programs
Robot/Motion Utility Programs
AdeptVision Utility Programs
Network Utility Programs
AdeptModules SPEC Data
© 1984-1997 by Adept Technology, Inc.
CH

Instructions for Adept Utility Programs

Version 12.1


adept
Adept Utility Disk
For use with V+ 12.1 (Edit D & Later)
FLIST README.TXT for program information
General-Purpose Utility Programs
Robot/Motion Utility Programs
AdeptVision Utility Programs
Network Utility Programs
AdeptModules SPEC Data
© 1984-1997 by Adept Technology, Inc.

Part # 00962-01000, Rev. A
September, 1997

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

Table of Contents

Introduction 1

This manual describes the utility programs on the Adept Utility Disk for V$^+$ version 12.1. For each utility, the following information is provided:

- A list of the equipment and conditions required to use the program
- A detailed description of the procedure for using the program
- Explanations of program requests for input from the user
- Any special considerations that apply to the program

An index is provided at the end of this manual to assist with locating topics of interest.

NOTE: The programs described in this manual have been tested by Adept Technology to verify that they perform their intended functions. However, Adept Technology assumes no responsibility for any errors or omissions in the programs or in this support documentation, nor for the results of any modification of the programs by the user.

1.1 Notes, Cautions, and Warnings

There are four levels of special notation used in this instruction handbook. 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 statement will typically describe the potential hazard, its possible effect, and the measures that must be taken to reduce the hazard.

WARNING: If the WARNING is indicated with a lightning bolt instead of an exclamation mark, an electrical danger or shock is possible for personnel working with the system.

CAUTION: If the action specified in the "CAUTION" is not complied with, damage to your equipment could result.

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

1.2 Conventions

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.
`regular monospace`	Monitor displays and code examples
`oblique monospace`	Placeholders for information that you provide in formal syntax definitions. You must replace such a placeholder written in ***`bold`*** weight but need not replace an optional one, which is written in *`regular`* weight.
bold	In a typing or entering instruction, anything that you type exactly as it appears. For example, if you are asked to type **execute 1 a.diskcopy**, you would 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 subroutine names, variable names, and program names, such as **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.
italics	Placeholders that you must provide in typed input. This font also indicates new terms and other emphasized words.
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.
SMALL CAPITALS	A *physical* key or button that you must press, such as the ENTER key.

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, and Pressing Items

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 picking 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* buttons or keys. For example, you press the SAVE key or press the ENTER key. By contrast, you select or choose a window button.

1.3 Functional Grouping of Utility Programs

The Utility Disk contains four groups of utility programs:

- General-purpose utility programs
- Robot/Motion utility programs
- Vision utility programs
- Network utility programs

General-Purpose Utilities

This group of utilities can be used on all Adept MV controllers.

In the File Type column, B = Binary, A = ASCII, and R = Read-Only. See the next section for a description of file types.

General Utilities

File Name	Description of Contents	Option Required	File Type
CONFIG_C.V2	Utility for configuring controller-based system features		B
DISKCOPY.V2	Utility for copying diskettes and manipulating disk files		B
EDITICON.V2	Icon editing utility	A-Series controller	B
FORMAT.V2	Utility for formatting the hard disk		B
LOADICON.V2	Callable routine for loading graphic icons into memory	A-Series controller	A
LOADTCH.V2	Callable routine for loading stored touchscreen calibration	System controller with touchscreen	A
PROFILER.V2	Utility for monitoring CPU usage	A-Series controller	B
SECURE.V2	Utility for restricting access to diskettes and disk files		B
SQUEEZE.V2	Utility for removing comments from program files		B
TOUCHCAL.V2	Utility for calibrating a touchscreen	System controller with touchscreen	B
XMODEM.V2	Utility for transferring files using the XMODEM protocol		R

Robot/Motion Utilities

This group of utilities can be used only with AdeptMotion VME or a robot controlled by an Adept MV controller.

In the File Type column, B = Binary, A = ASCII, and R = Read-Only. See the next section for a description of file types.

Robot/Motion Utilities

File Name	Description of Contents	File Type
BELT_CAL.V2	Utility for calibrating moving belts	R
DUTY_CYC.V2	Monitors the robot duty cycle values	B
LOADBELT.V2	Callable routine for loading stored belt calibration data	A
SAFE_UTL.V2	Utility for test and commission of CAT3 hardware	B
SFUTIL.V2	Utility for working with data for MOVEF/MOVESF	B
SPEC.V2	Utility for configuring and tuning AdeptMotion VME systems	B

Vision Utilities

This group of utilities can be used only with the AdeptVision option.

In the File Type column, B = Binary, A = ASCII, and R = Read-Only. See the next section for a description of file types.

Vision Utilities

File Name	Description of Contents	File Type
ADV_CAL.V2	Utility for calibrating stand-alone and robot-mounted cameras	B
ADV_USER.V2	Callable routines for use with ADV_CAL and for using calibration data	A
LOADAREA.V2	Callable routine for loading stored area vision calibration	A

Network Utilities

This group of utilities can be used with the indicated options:

In the File Type column, B = Binary, A = ASCII, and R = Read-Only. See the next section for a description of file types.

Network Utilities

File Name	Description of Contents	Option Required	File Type
FTP_CLNT.V2	Utility for AdeptFTP client functions	AdeptFTP Client/ Server	B
FTP_CLNT.V2	Utility for AdeptFTP client functions	AdeptFTP Client/ Server	B

1.4 Types of Files on the Utility Disk

All the utilities on the utility disk may be loaded with the V$^+$ LOAD command, and executed with the EXECUTE command (except ADV_USER, LOADAREA, LOADBELT, LOADICON, and TOUCHBELT). This section describes some limitations on copying and editing the utilities.

ASCII Files

You may FCOPY, LOAD, and STORE these unprotected ASCII files. They typically contain V$^+$ programs that are intended to be called from your own application program.

When you STORE your program, a copy of any of these routines present in memory and CALLed by your program will automatically be stored with your program. Do not edit these programs, as you may inadvertently change their functionality. If you use the STOREM command to store your program, the Adept programs must first be added to your program module (by using the MODULE command).

Read-Only Files

You can FCOPY, FLIST, and LOAD, these read-only text files, and you can EXECUTE the programs they contain. You can also view these programs with the LISTP command and the SEE editor. You cannot, however, edit or store these programs. When the FDIRECTORY command is used to list the file directory, these files are marked with an R in the on-screen file list.

Binary Files

Binary program files can be loaded and executed by V$^+$ just like ASCII program files.

You can freely FCOPY and LOAD these secured files, and you can EXECUTE the programs they contain. You cannot, however, edit the programs they contain, list them with the FLIST or LISTP commands or store them to a new file.

1.5 Installation

NOTE: The Adept utility programs are installed in each system shipped by Adept. The procedure for copying new utility programs on older systems is shown below.

If you have plenty of space on your hard disk, Adept recommends that you copy all the utility programs onto your hard disk. If you are short of disk space, you need only copy the utilities that you expect to use. Here are some recommendations:

General-purpose utilities:

- All users: DISKCOPY and CONFIG_C
- All robot users: SPEC

Specialized utilities:

- AdeptVision option: ADV_CAL, LOADAREA, and ADV_USER
- Belt tracking option: BELT_CAL and LOADBELT
- Touchscreen users: TOUCHCAL and LOADTCH
- AdeptMotion installers: SPEC and CONFIG_C
- Program developers: EDITICON, LOADICON, PROFILER, SQUEEZE, and SECURE
- AdeptFTP option: FTP_CLNT and FTP_SRVR
- Miscellaneous: XMODEM and FORMAT

You may load these utility programs directly from the Utility Disk. However, if you have the optional hard disk, Adept recommends that you use the following procedure to copy some or all of the programs to a subdirectory on the hard disk.

1. Create a subdirectory (refer to the *V$^+$ Operating System User's Guide* if you need more information about the FDIRECTORY command). Type:

   fdirectory/c c:\util\

2. Type **load a:diskcopy** to load the DISKCOPY utility.
3. Type **ex 1 a.diskcopy** to execute the DISKCOPY utility.
4. Use the DISKCOPY utility to copy all the files from drive A to drive C:

   a. Insert the Adept Utility Disk into the floppy drive (drive A).

   b. Choose

      4, Copy multiple files.

   c. When prompted, type **a** to specify drive A as input, and type **c** to specify drive C as output.

   d. Type **y** to verify the copy operations.

   e. Type ***.*** as the specification of the files to copy.

   f. Type **\util** as the output subdirectory.

g. Type **y** to replace any existing files of the same name.

h. When the name of the first file is displayed, type **g** (Go) to copy all files. (If you do not want to copy all of the files, type **y** (Yes) for each file you want copied.)

See the description of the DISKCOPY utility program later in this manual if you need more information about that program.

Description of Utility Programs 2

This chapter provides detailed descriptions of the utility programs provided on the Adept Utility Disk.

The utilities are presented in alphabetical order, with the description for each utility starting on a new page.

Each utility contains the following sections, as applicable:

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.[1]

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.

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.

[1] Or CALLed if the program is incorporated into your application code.

ADV_CAL

Description

Camera calibration utility

Disk File Name

ADV_CAL.V2

Program Name

a.adv_cal

Functions

Perform camera calibration.

Abstract

In most vision applications, the cameras must first be calibrated. The principle results of camera calibrations are:

- A millimeter-to-pixel ratio that allows the vision system to use distances in real-world millimeters
- Camera-to-robot calibrations that allow robots to acquire parts
- A transformation that accounts for perspective distortion caused by a camera imaging element that is not parallel to the image surface

This utility is described in the *AdeptVision User's Guide*.

ADV_USER

Description

Routines for use with ADV_CAL and for using calibration data

Disk File Name

ADV_USER.V2

Abstract

This file contains utility programs that can used by application programs that access the calibration data for the AdeptVision product.

See the *AdeptVision User's Guide* for details on the programs in this disk file.

BELT_CAL

Description

Conveyor tracking calibration utility

Disk File Name

BELT_CAL.V2

Program Name

a.belt_cal

Functions

Determine the calibration data required to have the robot track a moving conveyor belt.

Test the calibration data by having the robot track a point on the conveyor.

Abstract

There are various data required to have the robot track a moving conveyor belt. These include the following:

- The relationship between belt position and encoder output
- A description of the where the belt is located in the robot reference frame, and the direction in which the belt moves
- A definition of the limits of the robot working range along the belt

The belt calibration program can be used to define all these data, and to store the data in a disk file for subsequent use by application programs.

Once the data are defined, the performance of the robot/conveyor/encoder system can be tested.

Requirements

The following items are needed for use of the program:

- Adept robot system with the conveyor tracking feature
- V^+ system disk with the conveyor tracking feature
- Adept Utility Disk containing the file BELT_CAL.V2
- Manual control pendant[1]
- Conveyor-tracking encoder
- Cable from encoder to the robot controller
- Calibration disk[1]
- Calibration pointer[1]

[1] The indicated items are available from Adept as system options.

- Conveyor belt that can be stopped and started (provided by the customer). (The portion of the belt to be used by the robot must be straight and flat, but not necessarily level or parallel to any robot axis.)

The following conditions must exist before the program can be used:

- The conveyor belt must be securely installed so it cannot move relative to the robot.
- The encoder must be mounted on the conveyor so that its drive disk rolls on the belt, or so the shaft of the encoder is connected to the belt drive train without backlash.
- The calibration pointer must be mounted on the robot end-effector flange.
- The V^+ system must be loaded from the system disk.
- The robot must be calibrated and COMP mode must be selected on the manual control pendant.

Procedure

1. If you want to save all of your programs and data on a disk, type:

   store save_all

2. Clear the system memory by typing:

   zero

3. Load the calibration program into the system memory by typing:[1]

   load \util\belt_cal

4. Start execution of the program by typing:

   execute a.belt_cal

5. If your system is configured for more than one belt encoder, enter the number of the encoder channel you want to work with. The channels are numbered 1 through 6.
6. If your system controls more than one robot, enter the number of the robot you want to work with.
7. Select the options you want from the menu displayed (see below).
8. Follow the prompts provided by the program during the various procedures (see below). At the end of the calibration procedure you will be able to store the calibration information in a disk file for later use.
9. Delete the calibration program by typing the commands:

   kill
   delete a.belt_cal

   NOTE: The above command deletes all the calibration data from memory, except for any variables that are referenced by some other program in memory.

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

10. Later, to restore the calibration data from the disk, your application program should CALL the subroutine **load.belt** contained in the utility file LOADBELT.V2.

Program Prompts

The conveyor tracking calibration program displays a menu of selections to the user. The following paragraphs describe the menu items and provide additional information to help you decide how to respond to prompts from the program. After you respond to a prompt, the program displays instructions on the monitor or terminal.

NOTE: If you want to exit the program at any prompt, press CTRL+Z.

`0 => Exit to system monitor`

Choose this item to exit the program and return to the V^+ monitor.

`1 => Perform complete calibration sequence`

This menu selection initiates processing of all the steps for defining the conveyor calibration data.

If you want to define only a portion of the calibration data (for example, because something in the setup has changed), do not make this selection. Select only the appropriate menu choice(s).

During the calibration procedure the program will ask you to move the robot pointer to several locations on the conveyor. To see a description of the locations and a diagram indicating proper positions, respond **y** to the prompt

```
Do you want to see an explanatory diagram (Y/N)?
```

See the descriptions that follow for more information on individual steps of the calibration procedure.

`2 => Load calibration data from the disk`

This selection loads belt calibration data from a previously created. DAT file.

You are asked for the disk unit on which the calibration data file is stored. Then you are asked to specify the subdirectory in which the file is located. If the correct subdirectory path is shown as the default, press the ENTER (RETURN) key.

The program displays the names of all the files currently in the selected subdirectory that appear to be belt calibration data files. (The data is normally stored in a disk file with its name in the format BELT*nnn*.DAT, where *nnn* is a user-specified data set number.)

After you enter the desired data set number, the data is read from the disk, and is used to re-initialize the conveyor tracking system.

NOTE: See "Programming Considerations" on page 23 for information on the variables defined by the program for use during conveyor tracking.

`3 => Store calibration data onto the disk`

This selection allows you to store the current belt calibration data in a disk file for later use. After the file is created, the data can be read from disk by this program or by an application program that contains the subroutine **load.belt** (provided in the file LOADBELT.V2 on the Adept Utility Disk).[1]

You are asked which disk device you want the data stored on. Then you are asked to specify the subdirectory in which the file is to be located. If the correct subdirectory is shown as the default, you can simply press the ENTER (RETURN) key.

The program displays the names of any files currently in the selected subdirectory that appear to be belt calibration data files. (The data is normally stored in a disk file with its name in the format BELT*nnn.*DAT, where *nnn* is a user-specified data set number.)

After you enter the data set number you want to use, the data file is created. (The name of the new data file is BELT*nnn.*DAT, where *nnn* is the data set number you entered.)

`4 => Display belt encoder output`

Choose this selection to have the program display the position and speed of the conveyor as indicated by the output from the belt encoder. This is useful to verify that the encoder is functioning correctly.

`5 => Set encoder scale factor`

The encoder scale factor defines how far the belt moves between each count output by the belt encoder. This menu selection initiates the steps required to determine that scale factor.

If you know the value of the scale factor, you can enter it directly to the program. Otherwise, you can use the robot to measure the belt motion between two positions.

> **NOTE:** When positioning the calibration pointer over the calibration disk, keep the robot end-effector flange from rotating relative to the workspace. Otherwise, any eccentricity of the calibration pointer results in errors in the measurements between recorded positions.

The procedure for using the robot to determine the encoder scale factor involves recording a robot location on the stationary belt and its corresponding encoder reading. Then the belt is moved so that point on the belt moves to the opposite extreme of the robot's reach. A new robot location and encoder reading are recorded. The difference between the two robot locations determines how far the belt moved, and the difference between the two encoder readings is the number of encoder counts output during that belt motion.

[1] See the LOADBELT description in this manual for information on how your application programs can read calibration data from disk files.

The Adept calibration pointer and disk are designed to simplify this procedure. The calibration program asks you to perform the following steps:

1. Place the disk on the belt as far upstream as the robot can reach, on the side of the belt nearest the robot. (It's a good idea to tape the disk to the belt so it can't slip when the belt is started and stopped.)
2. Center the robot pointer on the disk.
3. Press the ENTER key to indicate the pointer is in place.
4. Back the pointer away and run the belt until the disk moves near the other extreme of the working range of the robot. (Be careful that the disk does not slip on the belt when the belt is started and stopped.)
5. Without moving the disk, center the pointer on the disk a second time.
6. Press the ENTER key again to indicate the pointer is in place.

`6 => Define belt reference frame`

The belt reference frame describes the direction in which the belt moves and the orientation of the surface of the belt.

Four robot locations on the conveyor must be recorded to define the belt reference frame. The two locations used to determine the encoder scale factor (see above) are used when the complete calibration procedure is performed. Otherwise, all four locations need to be recorded.

For each location requested by the program, position the calibration disk on the belt and center the robot pointer on the disk. Note these considerations when following the instructions from the calibration program:

1. The calibration disk must not move across the belt while the upstream and downstream locations are being recorded.
2. The upstream, downstream and across-the-belt locations must all be recorded at the same height above the surface of the belt.
3. The center-of-the-belt location can be anywhere in the vicinity of the center of the robot working range on the belt.
4. Care should be taken to keep the robot end-effector flange from rotating relative to the belt. Otherwise, any eccentricity of the calibration pointer results in errors in the measurements between recorded positions.

`7 => Define belt working window`

The system should be told how far upstream and downstream the robot can reach. That is done by defining a robot location at each end of the desired working *window*. These locations are used to define imaginary planes perpendicular to the belt, and thus only their positions *along* the belt are used by the program.

> **NOTE:** The upstream and downstream locations should be defined such that the robot is able to reach all belt locations within the working window.

`8 => Initialize the belt variable(s) with current data`

This menu item causes the system to incorporate the calibration data determined by the steps above. As a result, the belt variable **%b[i]** is defined, where **i** is the number of the belt encoder being used.

> **NOTE:** See "Programing Considerations" below for information on the variables defined by the program for use during conveyor tracking.

`9 => Test belt calibration`

To test the belt calibration you can have the robot track a point on the belt. Place a calibration disk on the belt and position the calibration pointer slightly above the disk.

As the robot is tracking the belt, you can stop and start the belt, change its speed, and reverse its direction of motion. You can stop the belt tracking by pressing the ENTER key on the keyboard, or the DONE button on the manual control pendant. The robot automatically stops tracking the belt if it moves close to an end of the defined working window along the belt.

You should verify that the robot accurately tracks the disk. That is, it should remain centered over the disk as the disk moves down the belt. Inaccurate tracking indicates the belt calibration procedure should be repeated.

If the robot correctly follows the path of the disk, but gains on it or lags behind, only the encoder scale factor needs to be recalibrated. If the robot moves across the belt as it tracks the disk, you should repeat the procedure that defines the belt reference frame.

```
Do you want to save the calibration data on the disk (Y/N)?
```

After completing the calibration procedure the calibration data can be stored in a disk file for use later by entering **y** to the prompt shown above. The steps that follow are identical to those for the main-menu selection "Store calibration data onto the disk", which is described above.

`10 => Select belt to work with`

Use this menu item to change the number of the belt encoder channel you want to work with. The number of the channel currently selected is displayed above the main menu.

`11 => Select robot to work with`

If your V^+ system controls more than one robot, use this menu item to change the number of the robot you want to work with. The number of the robot currently selected is displayed above the main menu.

Programming Considerations

For use when tracking the conveyor belt(s), the program defines the global variables listed below. Your application program can access these variables, or it can define other, equivalent variables by calling the routine contained in the file LOADBELT.V2 on the Adept Utility Disk.

Variable Name	Variable Type	Description
%b[i]	Belt	Belt variables, which define the motion of the conveyor belts (i = 1 to 6)
belt.cal[i]	Real-value	TRUE/FALSE indications of whether or not belts have been calibrated (i = 1 to 6)
belt.nom[i]	Transformation	Nominal belt transformations, which define the position, orientation, and direction of conveyor belts (i = 1 to 6)
belt.sf[i]	Real-value	Scale factors for belt travel per belt encoder count (i = 1 to 6)
belt.win1[i]	Transformation	Locations defining one end of windows on the respective conveyor belts (i = 1 to 6)
belt.win2[i]	Transformation	Locations defining other end of belt windows on the conveyor belts (i = 1 to 6)

Special Considerations

The calibration program defines several real-valued and location variables that use **bc.** and **belt.** as prefixes. These variables must not be modified by the user, or by other programs. (All these variables are deleted from memory when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file BELT_CAL.V2 is a read-only file on the distribution diskette. Thus, the programs in the file cannot be edited or stored from memory to a disk. The FCOPY command can be used to copy the disk file to another disk.

The disk-copy utility program (in DISKCOPY.V2) can be used to make a backup copy of the entire distribution diskette. That program can also be used to copy the file BELT_CAL.V2 from the distribution diskette to the optional hard disk.

See the V$^+$ *Language User's Guide* for additional details on programming conveyor belt applications.

If you are using *AIM MotionWare* for belt tracking operations, the belt calibration utility built into MotionWare can be used to create, store and load calibration data.

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

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 disk.

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 in the SIO module, 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 12.1
- V$^+$ system disk
- Adept Utility Disk containing the file CONFIG_C.V2 (or a copy of the file on the hard drive)

Before you can run CONFIG_C:

- The V$^+$ system must be loaded from the system disk.

- 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 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 (see below).

   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.

   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 (type **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 disk. Thus, after making all the desired changes to the configuration, turn off power to the controller and reboot the system from the modified system disk.

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.

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

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.

You should clearly mark the system disk to indicate that it contains your modified configuration. It is advisable to make a backup copy of the modified system disk. The DISKCOPY utility can be used for this purpose.

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.)

The entire DISKCOPY utility is included in the file CONFIG_C.V2. Thus, after that file is loaded, DISKCOPY is in memory and available for use. If you attempt to load DISKCOPY from disk when CONFIG_C is in memory, you will see many `*Program already exists*` errors, which can be ignored.

The disk file CONFIG_C.VE 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 disk-copy utility program (DISKCOPY). The disk-copy program can be used to make a backup copy of the entire distribution diskette.

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 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
```

Program Prompts

The configuration utility program displays the following menu:

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

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

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 program (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 disk files.
2. Copy a new V$^+$ system to the system disk, install the appropriate device modules, and copy the 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 31 for a detailed description of these processes.

2 => V+ System Configuration Data

This menu item displays the following menu:

```
*** CONTROLLER CONFIGURATION EDITOR ***

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

Enter selection and press ENTER:
```

The description of these menu items follows:

1 => DISPLAY system CONFIGURATION

This menu item displays all the statements in the configuration file.

2 => EDIT system CONFIGURATION

This menu item allows you to edit the individual statements in the configuration file. See page 35 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.

`3 => Robots and Device Modules`

For details on the features of this menu item, refer to the *AdeptMotion VME Developer's Guide*.

```
*** ROBOTS AND DEVICE MODULES ***

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

Enter selection and press ENTER.
```

`4 => SysIO NVRAM`

This menu item displays the following menu:

```
*** CONTROLLER NVRAM ***

0 => Return to MAIN MENU
1 => DISPLAY NVRAM contents
2 => INITIALIZE the NVRAM
3 => SET IP ADDRESS in the NVRAM

Enter selection and press ENTER.
```

The description of these menu items follows:

`1 => DISPLAY NVRAM contents`

The controller model, serial number, network IP address, and list of installed options is 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 in a new SIO module. This needs to be done only when a replacement SIO module has been installed in the controller.

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.

`3 => Set IP address in the NVRAM`

```
This option adds or changes the network IP address and IP subnet
mask for the controller.
```

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 be hav effect if they are not written to the NVRAM.

Detailed Descriptions of Program Operations

`1 => V+ Installation`

This description of the installation process is divided into two parts: an overview section, followed by the installation procedure itself.

Overview

The options under this menu item are used when upgrading a V$^+$system. These operations 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 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 it 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 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.

      If the old V$^+$ system is version 11.1 edit J, or later, the CONFIG_C utility can be used to create all the robot and encoder configuration data files. The files are stored in the appropriate subdirectory, with the appropriate file names, for access later with CONFIG_C (see step #3).

      If the old V$^+$ system is prior to version 11.1 edit J, use the CONFIG_R utility (if the system does not access an external encoder) or the SPEC utility to create these files. A *binary* (as opposed to ASCII) data file must be created for each device. See the *AdeptMotion VME Developer's Guide* for a description of the SPEC utility. To take advantage of the automated features available for step #3 of the installation process (see below), the files should be stored on any disk in the subdirectory \CONFIG_C\. Furthermore, the file names should be CFG*rr_mm*.SPC, where *rr* is the robot number (00 for the external encoders, 01 for robot #1, etc.), and *mm* is the corresponding device-module ID (with a leading zero if necessary to make it two characters). (The device-module ID numbers are displayed by the V$^+$ ID monitor command at the ends of the Robot lines.)

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 CONFIG_C program looks for the configuration data files in the subdirectory \CONFIG_C\ with names in the form described above.

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 this edition of CONFIG_C and the current V$^+$ system. Use the edition of CONFIG_C that was supplied with your old V$^+$ system.

This menu item can be used only if the old V$^+$ system is version 11.1 edit J, or later.

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 the 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 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.

You need the following items to install a new V$^+$ system:

1. An old V$^+$ system (on a diskette or the hard drive).

   This is the V$^+$ system that you have been using. The configuration of this system is duplicated in the new V$^+$ system (see below).

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. (In general, those data files can be created only with the edition of CONFIG_C, CONFIG_R, or SPEC that you received with the old V$^+$ system.)

3. A new V$^+$ system (on a diskette or the hard drive)

   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.

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 instead of on the hard drive)

   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 uninterruptable 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 following questions about the system disks:

```
Are you going to install the new V+ system on the hard drive (Y/N)?
Is your OLD V+ system currently on the hard drive (Y/N)?
Is your NEW V+ system currently on the hard drive (Y/N)?
```

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. (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.) You should be familiar with the use of the DISKCOPY program to copy a V^+ system and to copy a diskette image.

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.
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 the floppy drive is indicated, you are asked to insert the diskette.

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 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 the 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 RAM. Note, however, that 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 selecting 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 sub-menu 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 NETWORK configuration
 7 => Change ROBOT configuration
 8 => Change SERIAL configuration
 9 => Change SERVO_BOARDS configuration
10 => Change SYSTEM configuration
```

```
11 => Change TASKS configuration
12 => Change VISION configuration

Enter Selection:
```

Choosing a specific option allows you to edit the statements for that part of the system. The statements display 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 start-up. The general form of each statement is:

```
ITEM # = "/ATTRIBUTE value /ATTRIBUTE value.../ATTRIBUTE value"
```

Not all items have a number (#). Some items have only a value associated with them and not a list of attributes.

When you select an option, 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 an option from the edit system configuration sub-menu.
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 them. You have three options:
   a. If you select **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 select **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 pressing 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 re-edit 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 sub-menu 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 sub-menu 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 the statement specifies PROCESSOR and a number. Processor numbers are determined by the processor board address switches, not their physical order. 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 the statement specifies BOARD and a number. Analog I/O board numbers are determined by the analog board address switches, not by physical order. 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 the 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. 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$^+$ Operating System 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 the 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_BOARD menu option (see page 39). 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^+ Operating System 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 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.

```
7 => 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 the 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.

```
8 => Change SERIAL configuration
```

This section specifies the communications protocols and parameters for various serial data communications hardware and software.

The left-hand side of the statements specifies either GLOBAL_PORT (for the global serial lines on the SIO module) or LOCAL_PORT (for the local serial lines available on each 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 on the SIO front panel (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*).

> **NOTE:** If your controller has a Manual Control Pendant attached, you should *not* change or add a statement for GLOBAL_PORT 4.

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, the 19200 baud rate cannot be used for one of the ports if the 7200 or 38400 baud rate is used for the other port:

030:	LOCAL_PORT 1 and LOCAL_PORT 2
SIO:	GLOBAL_PORT 1 and GLOBAL_PORT 4
	GLOBAL_PORT 2 and GLOBAL_PORT 3

```
9 => 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. 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 SIO (configured with the DIGITAL_INPUT menu option, see page 38) or from a vision (EVI) board (configured using the V.STROBE system switch or the ADV_CAL.V2 utility program).

```
10 => Change SYSTEM configuration
```

This section specifies the initial default disk and monitor speed, the graphics system defaults (including the pointer device double-click interval), the network buffers, the servo interrupt rate, and the rate at which motion trajectories are planned.

DEFAULT_DISK can be given any valid device and directory path specification. The default disk can be changed 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.

GRAPHICS 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 process as a double click.

The NETWORK statement specifies the number of buffers to be allocated for receiving messages with the DDCMP protocol. If no serial lines are configured to use this protocol, leave the setting at zero. (This statement is not related to the NETWORK section of the configuration data.)

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.

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.

```
11 => 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. Refer 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 the 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. See the *Adept MV Controller User's Guide*.

The attribute for USER x PROCESSOR y is a sequence of 16 integers that specifies the user task priority in each of the 1ms 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] Standard V^+ allows 7 user tasks. Systems with the V^+ Extensions license are allowed 28 tasks.

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.

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.

```
12 => 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 associates 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 attributes to 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.

/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*).

/FRAME_SIZE specifies the size of the virtual frame buffers that the physical frame store is divided into (see the *AdeptVision User's Guide*).

See the Appendix titled "Using DEVICE with Vision" in the *AdeptVision User's Guide* for additional details on setting vision memory allocation.

DISKCOPY

Description

Disk copy utility

Disk File Name

DISKCOPY.V2

Program Name

a.diskcopy

Functions

Copy an entire diskette to another diskette.

Copy a V^+ system from one disk to another.

Copy individual or multiple files (using wildcards) from one disk to another or from one subdirectory to another on the same disk.

Delete individual or multiple files (using wildcards) from a disk.

Abstract

Certain file-copy operations cannot be performed with the V^+ FCOPY monitor command or with the READ and WRITE program instructions. The disk-copy utility program can be used to perform those operations. For example, use this program to copy from one diskette to another using a single diskette drive.

Protected disk files cannot be copied with the FCOPY command. This program will copy protected files to a hard disk.

Operations on multiple files cannot be performed interactively with the FCOPY or FDELETE commands. This utility permits interactive operations to copy and delete files.

Requirements

Before you can run DISKCOPY:

- The V^+ system must be loaded from the system disk.
- 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 as a program task other than #0.

Procedure

1. Load the program:[1]

   load \util\diskcopy

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

2. Execute the program:

   execute a.diskcopy

   (Or type **execute 1 a.diskcopy** to execute the program as task #1.)

3. Select an option from the menu displayed:

```
*** DISKCOPY MENU ***

   0 => EXIT to system monitor
   1 => COPY V+ SYSTEM to another disk
   2 => COPY DISK image to another disk
   3 => (Not used)
   4 => COPY multiple FILES
   5 => DELETE multiple FILES

Enter selection:
```

4. Follow the prompts provided by the program (see “Program Prompts” below).

5. The disk-copy program can be deleted from system memory by typing (type **kill 1** in place of **kill** if the program was executed as task #1):

   kill
   delete a.diskcopy

Program Prompts

The disk-copy utility program displays a menu of selections to the user. The following paragraphs describe the menu items.

NOTE: To exit the program at any prompt, you can press CTRL+Z.

0 => EXIT to system monitor

Choose this item to exit the program and return to the V+ monitor.

1 => COPY V+ SYSTEM to another disk

This menu selection initiates copying a V+ system from one disk to another. This can be used to copy a V+ system from a distribution diskette to the hard disk, or from the hard disk to a diskette. (If you want to copy a V+ system from one diskette to another, you must use menu item #2.)

All of the following items are copied when a V+ system is copied:

a. V+ system file

b. V+ configuration file

c. V+ boot file

d. \SYSTEM\ subdirectory containing device-module files

e. \CALIB\ subdirectory containing robot calibration program files

`2 => COPY DISK image to another disk`

This option copies an entire diskette to another disk.

> **NOTE:** This selection will not copy a disk image from a diskette to the hard disk, since the result will be that the hard disk will have the same storage capacity as a floppy disk. The selection “Copy multiple files” copies files from a diskette to the hard disk (see below). You also cannot copy a disk image from a double-density floppy to a high-density floppy disk.

During the copy operation, the contents of the input diskette are stored in system memory. You will be asked to exchange the input and output diskettes. The buffered disk contents will then written to the output disk. Keep the following in mind when performing a one-drive copy operation:

1. When you are asked to change the diskette in the drive, make sure you insert the correct diskette. *The program does not perform any check to verify that the correct diskette is present.*
2. If there is not enough RAM available to hold the entire contents of the input disk, you will have to exchange the input and output diskettes more than once during the copy operation.
3. If there are other program tasks active, which could be consuming memory, you are asked if it is okay for the copy operation to continue.
4. After the copy is complete, you have the option of creating another copy.

`4 => COPY multiple FILES`

This option copies files from one disk to another. This function allows you to:

- Copy multiple files from one disk to another (or from one subdirectory to another, on the same or different disks).

  NOTE: You could perform the copy operations with individual FCOPY monitor commands. However, with this program function it might be possible to specify all the files to be copied with a *single* file specification containing wildcard characters (see below). (The FCOPY monitor command does not accept wildcard characters in its file specifications.)

- Copy protected files from a diskette to a hard disk. (The FCOPY monitor command cannot be used to copy protected files.)

  This menu selection will copy protected and read-only files onto the hard disk (and from one subdirectory to another on that disk), provided that the hard disk is properly secured. If the hard disk is not secured, the program gives you the option of securing the disk.

  NOTE: If you specify that the hard disk should be secured for exclusive use by the current system, then it is not possible to use that hard disk with any other system until it is reformatted. (Formatting *erases* everything stored on the disk.)

The program first asks for the input and output disk devices to be accessed. In addition to the disk units (for example, “A” or “C” for local disks), the responses can include specifications for the type of devices to be accessed (for example, “NFS>” for an AdeptNFS device).

Next you are asked:

```
Do you want to verify the copy operations (Y/N)?
```

Next, you are asked to enter a file specification string. (To exit this menu selection and return to the main menu, press ENTER without a file specification).

Wildcard characters (asterisks, “*”) can be used in the file specification. A wildcard character *within* a file name or extension indicates that any character should be accepted in that position. A wildcard character at the *end* of a file name or extension indicates that any trailing characters are acceptable. For example, the file specification “*.*” references all the files on the input disk.

An optional subdirectory specification can be included in the (input) file specification. If no subdirectory specification is included, the file(s) are copied from the subdirectory shown before the prompt. If a subdirectory is specified, it supersedes or is combined with the default shown. (For an explanation of how subdirectories are combined, see the chapter “Using Files” in the *V$^+$ Operating System User’s Guide* or the description of the DEFAULT monitor command in the *V$^+$ Operating System Reference Guide*).

The following prompt is displayed next:

```
Enter output subdirectory (default is \xxx\):
```

If a subdirectory specification is entered, it supersedes or is combined with the default shown, and all of the selected files will be copied to the resulting subdirectory on the output disk. If ENTER is pressed without a subdirectory specification, all the files copied will be directed to the default subdirectory on the output disk.

NOTE: If you specified that the copy operation is to use the same diskette drive for both input and output, you are asked at this point to confirm that you want to continue.

When the input file specification omits either the name or the extension, or includes a wildcard character, (and the copy operation is between two disk drives in the Adept controller) the program displays the prompt

```
Do you want existing files automatically superseded (Y/N)?
```

If you enter **y**, the program will delete (*without warning*) any file on the output disk with the same name as a file to be copied. If you enter **n** and the program encounters a file on the input disk that has the same name as an existing file on the output disk, the program will display a warning and ask if you want to supersede the file on the output disk.

The name of each file on the input disk that matches the file specification is displayed, and you are asked whether or not the file should be copied to the output disk. One of the following responses must be entered:

Response	Meaning
Y	Yes, copy the file

Response	Meaning
N (or press ENTER)	No, do not copy the file
G	Go, copy this file and all subsequent files
Q	Quit, without copying this file

If you specify a copy operation between two disk devices, or between two subdirectories on one device, the copy operation is very straightforward. The program simply copies from the input to the output.

If you specify a copy operation between floppy disks in the Adept controller, the procedure for the copy operation is slightly different. In this case, the contents of the input file are stored in system memory, the user is asked to remove the input diskette and insert the output diskette, and then the buffered file contents are written to the output disk. Because of this mode of operation, there are some points to know when performing a one-drive copy operation:

1. When you are asked to change the diskette in the drive, be careful to make sure you insert the correct diskette. *The program does not perform any check to verify that the right diskette is present*.
2. If there is not enough memory available to hold the entire contents of the input file, you will have to exchange the input and output diskettes more than once during the copy operation.
3. If there are other program tasks active that are consuming memory, you are asked if it is okay for the copy operation to continue.
4. After the output disk is written, you have the option of writing another copy of the output.

`5 => DELETE multiple FILES`

You are asked to specify a disk unit to access, and then successive file specifications. (You can press ENTER without a file specification to exit this menu selection and return to the main menu.)

The file specification string can contain the "*" wildcard character as part of the file name or file extension. A wildcard character within a file name or extension indicates that any character should be accepted in that position. A wildcard character at the end of a file name or extension indicates that any trailing characters are acceptable. For example, the file specification "test*" causes all the files on the disk with names beginning with "test" (with any extension, since ".*" is assumed when no extension is specified) to be considered for deletion.

An optional subdirectory specification can be included in the file specification. If no subdirectory specification is included, files contained in the default subdirectory on the specified input disk will be referenced.

The program displays the name of each file that matches the given file specification, and asks whether or not the file should be deleted. One of the following responses must be entered:

Response	Meaning
Y	Yes, delete the file
N (or press ENTER)	No, do not delete the file
G	Go, delete this file and all subsequent files without asking again
Q	Quit, without deleting this file

Special Considerations

The disk-copy program defines several real-valued variables with the prefix **dc.** that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The entire DISKCOPY program is included in the CONFIG_C program. Thus, if either of these utilities are in memory, many `*Program already exists*` error messages are displayed when the other program is loaded from disk. These messages can be ignored.

The disk file DISKCOPY.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

When you try to copy a protected file (i.e., one with the “P” attribute shown in its directory entry) from a diskette to the hard drive, the program may indicate that the hard drive must first be *secured*. If you indicate that it is okay for the program to do that, the hard disk will be secured and the program will be copied. A secured disk is totally compatible with the V^+ system.

DUTY_CYC

Description

Monitors the robot duty cycle values

Disk File Name

DUTY_CYC.V2

Program Name

a.duty_cyc

Functions

Monitors the duty cycle values of an AdeptThree-XL robot.

Reports back a set of percentage data against the working duty-cycle limits derived for the working ambient temperature.

Abstract

The Adept duty cycle program (**a.duty_cyc**) monitors the duty-cycle values and reports back a set of percentage data against the working duty-cycle limits derived for the working ambient temperature. These include:

- How hard the robot is running
- On-line percentage values (percentages values greater than 90 percent for any of the joints indicates a potential danger of overheating the robot)

Requirements

The items listed below are needed to run the **duty_cyc( )** routine. All of these (except as indicated) were supplied with your Adept system.

- Adept system controller, using V^+ version 12.1
- AdeptThree-XL robot

Procedure

1. Load the program:[1]

   load \util\duty_cyc

   NOTE: This program must be run after the robot is calibrated and appropriate application programs have been initiated.

2. Execute the program:

   ex 1 a.duty_cyc

3. The following prompt is displayed if you have previously run **a.duty_cyc**:

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

```
Duty cycle profiler is already initialized. Re-initialize
(Y/N)?
```

4. Type:

   Y

   (Typing Y will re-initialize program; typing N will continue the program)

5. The following message will appear:

```
This utility can be used to monitor the run-time duty cycle
values and report back a set of percentage data against the
working duty-cycle limits derived for the working ambient
temperature. They indicate how hard the robot is running. A
percentage value that is greater than 90% for any of the
joints indicate a potential danger of overheating the robot.

Enter the worst-case ambient temperature (e.g., 30C or 80F,
the default is 50C, “C” is assumed if the scale is omitted):
```

6. Type (for example):

   95F

7. The following message will appear indicating which, if any, joints are inactive (example):

```
Duty cycle checking is inactive for the following joints:

3 4
```

NOTE: The user should allow AT LEAST 4 minutes of robot run time to let the duty cycle filter in servo reach a steady state.

The entry cycle profiler’s internal optimization process reaches steady state when the duty cycle percentage difference of J1 and J2 is within 2%.

```
A worse-case ambient temperature of 35C will be used.

Duty Cycle Percentages:

     J1: 35.46%  J2: 32.00%
```

(Percentages are readouts based upon the temperature selected in step #6 above.)

8. To stop the program type:

   abort 1

9. The duty cycle program can be deleted from the system memory by typing the following commands:

   kill 1
   delete a.duty_cyc

Special Considerations

The disk file DUTY_CYC.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

EDITICON

Description

Icon editing utility

Disk File Name

EDITICON.V2

Program Name

a.editicon

Functions

Create graphic icons and store them in disk files for retrieval later.

Read icon definitions from disk and modify them.

Abstract

The Adept graphics system provides the ability to store predefined graphic symbols in memory and to have them displayed on request. The Icon Editing Utility provides the means for defining the graphics symbols you want to use with your application programs.

After icons have been defined, they can be stored in disk files for retrieval later by applications programs. The subroutine **load.icon** on the Utility Disk in the file LOADICON.V2 can be called to load icons from a disk file into graphics memory. Then the GICON program instruction can be used to display the icons.

Requirements

The items listed below are needed to use the EDITICON program.

- Adept graphics-based system controller
- PC-style keyboard and high-resolution graphics monitor (system option)
  or
  AdeptWindows PC with Ethernet connection

Before you can run EDITICON:

- The V^+ system must be loaded from the system disk.
- 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 as a program task other than #0.

Procedure

1. Load the program:[1]

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

load \util\editicon

When loading is complete, you can remove the Utility Disk from the disk drive.

2. Execute the program:

   execute a.editicon

   (Or type **execute 1 a.editicon** to execute the program as task #1.)

3. After the program identifies itself, it displays the prompt

   ```
   Do you want to use a non-standard editing grid (Y/N)?
   ```

   To use the standard 20-by-20 editing grid (see below for an explanation), press the ENTER key.

   For a different editing grid, enter **y** and press the ENTER key. You are then asked to enter the width and height (in pixels) for the editing grid. (The minimum and maximum acceptable values are shown in the prompts. Press the ENTER key for either value if the default shown is the value you want.)

4. Use the pointing device to select items in the pull-down menus and to perform editing operations (see below).

5. The program can be deleted from the system memory with the following commands (type **kill 1** in place of **kill** if the program was executed as task #1):

   kill
   delete a.editicon

Program Operation

The Icon Editing Utility creates a new window on the monitor screen.

The following paragraphs describe the window and how to interact with the program.

Program Window

At the top of the program window there is a *16-color palette* that displays all the colors available for icon images. While using the icon editor you will select colors from this palette by clicking on a color with one of the mouse buttons. The first color in the palette (color #0) has a special interpretation for icons. Each icon pixel with that color is transparent, that is, the background shows through those pixels. The icon editor uses a checkerboard pattern (of black and background) to represent color #0 in the program's main window.

To the right of the 16-color palette there are three colored rectangles. These are the *drawing colors*, since they are the colors you use to draw the icon image. The letters M (= left) and R (= right) are displayed below the drawing colors to remind you that they correspond to the Left and Right mouse buttons, respectively. Each of the drawing colors can be changed by clicking with the corresponding mouse button on a color in the 16-color palette.

NOTE: The letter L, is used only with a third-party 3-button mouse.

When an icon is selected for editing, you are presented with two views of it. The actual icon is shown full size in a small window beside the program's main window. An enlarged view of the icon is shown in an editing grid in the program's main window. A square in the editing grid corresponds to one pixel in the icon. In general, you make changes to the icon in the editing grid, and your changes are shown in the full-size image.

If the icon is larger than the size of the editing grid, only a portion of the icon can be seen in the editing grid. Select a different portion of the icon by:

1. Moving the mouse pointer to the full-size icon,
2. Pressing and holding the left mouse button, and
3. Dragging the edit frame around the image. When you release the button, the newly selected portion of the icon is displayed in the editing grid.

If an icon array is displayed in the full-size display, there are two ways to select a different element to modify. Use the Select menu in the main window, or select in the full-size display by

1. Moving the mouse pointer to the current icon in the full-size display,
2. Pressing and holding the left mouse button,
3. Dragging to the desired element, and
4. Releasing the mouse button.

The standard size for the editing grid is 20 pixels wide and 20 pixels high (or the size of the icon if it is smaller).

You can specify a different size for the editing grid when you first execute the program (see above). If you want to change the size of the grid when the program is in use, exit the program, start it again, and specify the new grid size. (Any previous icon data will be displayed in the new grid.)

While using the icon editor, you will often see *pop-up windows* displayed in the working window. These pop-up windows contain messages or questions. Pop-up windows with buttons are closed after a button is pressed. Pop-up windows without buttons are closed after the next mouse click.

When a pop-up window requests typed input, keep in mind: the pop-up window must be selected (that is, its top bar must be blue) and the input will be completed when you press the ENTER key. If a default entry is displayed in the pop-up window, you can type BACKSPACE or CRTL+U to modify or erase the default entry.

Exiting the Program

To exit the program, choose **Exit** from the File menu. If you have icons defined, you will be asked if you are sure you want to exit. Choose **Yes** to exit, or **No** to continue with the program.

> **NOTE:** Icons are retained in (program) memory even when you exit the program. Thus, you can restart the program later and continue your editing session.

Editing an Icon

Immediately after the program is executed there is no icon selected for editing. Select the icon by loading an existing icon data file (see Load file in the File menu below), or by creating a new icon (see Create in the Icon menu below).

When the editing grid is displayed, use the mouse buttons to color squares in the grid, and thus pixels in the icon. Click on a square with one of the mouse buttons to change that square to the drawing color assigned to that button. Click on a square and drag the mouse pointer to color all the squares crossed while the button remains pressed. (If you accidentally apply the wrong color to a square, simply select the desired color from the 16-color palette and re-color the square.)

In addition to directly coloring individual squares in the editing grid, several editing operations modify the icon. These operations are accessed from the Edit menu, as described below.

Program Menus

There are four different pull-down menus located at the top of the program's window. Each of these menus is described in detail below.

Menu	**Purpose**
File	Read and write data files; clear all icons; exit
Icon	Create and delete icons; change icon size and name
Edit	Modify the current icon
Select	Select a new icon to edit

At times some menu items cannot be selected. For example, when there is no icon displayed, none of the items in the Edit menu can be selected. Menu items are displayed dimmed when they cannot be selected.

File Menu

The File menu is used to read and write disk files, and for overall administration of the program. The menu items have the functions listed below.

When the program asks for a file specification, the response can have the form:

physical_device>unit:directory_path\name.extension

Where everything except the name element is optional. The defaults for the first three items are taken from the current DEFAULT setting.   The default file extension is .DAT.

NOTE: You can use the V^+ FLIST command to display information in the icon data file. The information includes the creation date of the file and a list of the icons defined in the file.

Load file All the icons defined in a requested disk file are loaded into memory for possible editing. If there are any icons already in memory, the program asks if it is okay to delete them. If you choose **No**, the load operation is cancelled.

Append file	All the icons in a requested disk file are loaded into memory for possible editing. Unlike **Load file**, however, existing icons are not replaced.
Save to file	All the icons in memory are written to the specified disk file.
Clear all	After receiving confirmation, all the icons in memory are deleted.
Exit	Execution of the program is terminated. If there are icons in memory, they are retained until the program is deleted from memory.

NOTE: The icons can occupy a large amount of system program memory. Thus, if you do not need them retained, select **Clear all** before exiting the program.

Icon Menu

The Icon menu is used to create and delete icons, and to change their size or name. Selections in this menu allow recent changes to the icon to be canceled, or to be preserved as a basis for further changes.

Create	A new icon is created. The program asks for the name, size and offset for the new icon. If an icon array is being created, you are also asked for the arrangement of the elements to be used in the full-size display (see **Chg array** below). Icon names follow the rules for V^+ names, and can include an array index. The width and height of an icon can range from 5 to 255 pixels, and do not need to be equal. The positional (xy) offsets of an icon can range from 0 to 255 pixels.
Change size	Changes the size of the current icon to the new size entered by the user. The right and bottom edges are moved in response to the change.
Change name	Changes the name of the icon to the new name entered.
Copy	Copies the image of another icon to the current icon. If the source icon is larger than the current icon, the copy is taken from the upper left corner.
Delete	If confirmed by the user, the current icon is deleted from memory.
Undo edits	Recent editing changes to the icon are discarded and the previous saved image is restored. (The image is saved when the icon is created or selected, the size is changed, and when **Keep edits** is requested.)
Keep edits	Saves a copy of the current icon image for possible later use to undo subsequent changes.
Show data	Displays data for the current icon.
Chg array	If the current icon is an element of an icon array, this item can be used to specify a new arrangement of the icon elements in the full-size display window. The choices are:

-1	Have the program choose the arrangement that results in roughly equal numbers of rows and columns.
0	Display only the current array element.

>0 Display the array with the specified number of columns.

Background Allows selection of a new background color so you can see how the icon looks with the background on which it will be displayed.

To specify the new background color, simply click the desired color in the 16-color palette with any of the mouse buttons. (You can click outside the color palette to cancel the **Background** selection.)

To select a new icon after changing the background, see the Select menu below.

Edit Menu

The Edit menu modifies the icon currently being edited. All the changes performed by this menu can be canceled by choosing **Undo edits** in the Icon menu (see above). Before performing an editing operation, you may want to choose **Keep edits** in the Icon menu (see above) to save the current icon image.

Change color All the pixels in the icon with a specified color are changed to a new color.

Before selecting this item, make sure one of the drawing colors is the old color you want replaced in the icon.

Then, after choosing **Change color**, assign a new drawing color to the mouse button for the old color. Then all the pixels in the icon with the previous drawing color for that button are changed to the new color. (You can cancel the change operation by clicking outside the 16-color palette.)

Flood A region of the icon is filled with a new color. You specify the region and the new color by clicking on a square in the editing grid with one of the mouse buttons – the square selected indicates the region to flood, and the button used indicates which drawing color to us. (You can cancel the flood operation by clicking outside the editing grid.)

The region filled consists of all the connected squares with the same color as the first square. Any square with a different color is considered to be on the boundary of the region. Note that region squares can be connected vertically and horizontally, but not diagonally. Thus, the flooding does not *leak* through a diagonal boundary that is one square thick.

If the icon is larger than the editing grid, flooding does not extend beyond the boundaries of the grid.

Erase The entire icon image is cleared to the drawing color assigned to the mouse button pressed most recently.

Rotate 90° The icon image is rotated 90 degrees clockwise.

Shift right, **Shift left**, **Shift up**, **Shift down**

Each of these selections shifts the icon image one pixel in the indicated direction. (The first row or column shifted is lost; the last one shifted is duplicated.)

Right<>left The icon image is flipped about a vertical line through its center.

Top<>bottom	The icon image is flipped about a horizontal line through its center.

Select Menu

The Select menu is used to select a different icon to edit. The menu displays the names of all the icons currently defined in the program's memory, as well as the predefined items listed below.

If you select an icon by name, it is displayed in the full-size display and in the editing grid. If you select an icon array, however, you are first asked to enter the index for the array element you want to see, and the arrangement of the elements to be used in the full-size display (see the description of **Chg array** above).

Prev Element	If the current icon is an element of an icon array, this items selects the previous element in the array. (The last element is considered previous to element #0).
Next Element	If the current icon is an element of an icon array, this item selects the next element in the array. (element #0 is considered to follow the element item.)

Special Considerations

The program defines several real-valued and string variables with the prefixes **ic.** and **wn.** that must not be modified by the user, or by other programs. (All these variables are deleted from memory when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file EDITICON.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

FORMAT

Description

Format utility for the hard disk

Disk File Name

FORMAT.V2

Program Name

a.format

Functions

Format the hard disk and erase its entire contents.

Abstract

Before a hard disk can be used it must be formatted. Internal hard drives shipped with Adept systems are already formatted and ready to accept data. If you replace the internal hard drive, you will have to format the drive.

Requirements

You need the following items for FORMAT:

- Adept system controller with a hard disk
- V^+ system disk, version 11.0 or later

Before you can run FORMAT:

- The V^+ system must be loaded from the system disk.
- 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 as a program task other than #0.

Procedure

1. Load the program:[1]

   load \util\format

   When loading is complete, you can remove the Utility Disk from the disk drive.

2. Execute the program:

   execute a.format

   (Or type **execute 1 a.format** to execute the program as task #1.)

3. Select an option you want from the menu displayed.

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

4. Follow the prompts provided by the program (see below).
5. The program can be deleted from the system memory by typing the following commands (type **kill 1** in place of **kill** if the program was executed as task #1):

   kill
   delete a.format

Program Prompts

The format utility program displays a menu of selections to the user. The following paragraphs describe the menu items.

NOTE: If you want to exit the program at any prompt, press CTRL+Z.

`0 => Exit to system monitor`

Choose this item to exit the program and return to the V+ monitor.

`1 => Format the hard disk`

This menu selection initiates the procedure for formatting (or reformatting) the hard disk.

CAUTION: Reformatting the hard disk erases all the information stored on the disk. The program displays warning messages if it finds information present on the disk.

Before performing the actual format operation, the program displays the type of format that will be used, and asks if it is correct. Enter **y** if the format shown is correct. Otherwise enter **n**, and specify the capacity of the hard disk you have.

CAUTION: When reformatting the disk, enter **n** only if the disk had previously been formatted incorrectly.

`2 => Display disk format information`

This selection displays information on the hard disk format.

Special Considerations

The FORMAT monitor command must be used to format floppy disks, but it cannot be used to format a hard drive. The FORMAT utility cannot be used to format a floppy disk.

The FORMAT program defines many real-valued variables with the prefix **fw.** that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program completes execution normally, and when a DELETE command is used to delete the program from memory, as previously described.)

The disk file FORMAT.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

FTP_CLNT

Description

Utility for AdeptFTP client functions

Disk File Name

FTP_CLNT.V2

Program Name

a.ftp_clnt

NOTE: For more information on this utility, see the *AdeptNet User's Guide*.

FTP_SRVR

Description

Utility for AdeptFTP server functions

Disk File Name

FTP_SRVR.V2

Program Name

a.ftp_srvr

NOTE: For more information on this utility, see the *AdeptNet User's Guide*.

LOADAREA

Description

Load area vision calibration data for use by an application program

Disk File Name

LOADAREA.V2

Program Name

load.area

Function

Restore area vision calibration by reading data from a disk file created by the Adept area vision calibration program, ADV_CAL, or by the *AIM VisionWare* system.

Abstract

The Adept area vision calibration program **(a.adv_cal**) writes data into a disk file. An application program can read that data from the disk and restore it in the vision system by calling the subroutine **load.area** with the appropriate argument list.

This section describes the procedure for including the subroutine in your application program.

Requirements

The items listed below are needed to utilize the **load.area** subroutine in your application program. All of these (except as indicated) were supplied with your Adept system.

- Adept system controller with the AdeptVision option
- PC-style keyboard and high-resolution graphics monitor (system option)
  or
  AdeptWindows PC with Ethernet connection
- Your application program (not provided by Adept)

Procedure

1. Load the program:[1]

   load \util\loadarea

2. Use the system program editor to enter your application program into the system memory, or load your program from a disk file. Make sure the program calls the subroutine **load.area** (see below).
3. Place a diskette to receive your application program in disk drive A.
4. Store your application program and the **load.area** subroutine in a disk file by typing the command:

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

Programming Considerations

The **load.area** subroutine reads the area vision calibration information that is stored in a disk file. If the data are found to be valid, the following instructions are executed to initialize the vision system, and the data are returned to the calling program.

```
IF NOT SWITCH(VISION) THEN
    ENABLE VISION
END
PARAMETER V.THRESHOLD[camera_number] = stored_threshold_value
SWITCH V.BACKLIGHT[camera_number] = stored_backlight_value
VPUTCAL (camera_number) stored_calib_array[],
        stored_pmm.to.pix[,], stored_pix.to.pmm[,]
```

After the routine has processed the calibration data file, the AdeptVision system is ready to process images. All of the calibration data are returned to the calling program for completeness, but normally only the variable **to.cam** is used by the application program.

Your application program should call the subroutine at some point before the vision system begins acquiring images. The subroutine normally needs to be called only one time (for each camera) per operating session. The CALL instruction must have the following format:

```
CALL load.area($file, cam.virt, threshold, backlight, to.cam,
              cam.cal[], pmm.to.pix[,], pix.to.pmm[,],
              pmm.to.mm[,], mm.to.pmm[,], $error)
```

where the program arguments have the interpretations that are described below.

> **NOTE:** The variable names used here are for explanation purposes only. Your application program can use any variable names you want.

$file Input parameter that passes the specification of the disk file containing the area camera calibration data. This parameter can be a string constant, variable or expression.

The file specification usually has the form

disk_unit:*name.extension*

The file specification can also include a directory path, as follows

disk_unit:*directory_path*\name.extension

However, the file name and extension must have the form AREA*nnn*.DAT (where *nnn* is the data set number) if the data file was created by the Adept area vision calibration program ADV_CAL, or by the VisionWare system.

$error If no error occurs while **load.area** is executing, the parameter **$error** receives an empty string ("") and the output parameters described below receive the indicated values. If an error does occur, **$error** receives a string describing the error and the other output parameters are not defined. Thus, your calling program should check the variable **$error** before proceeding.

cam.virt Input parameter *and* output parameter that identifies the (virtual) camera receiving the calibration data.

As an input parameter, **cam.virt** specifies which *virtual* camera is to receive the calibration data. If this parameter is undefined or has the value zero, the camera number stored in the data file will be used. If **cam.virt** has a nonzero value, it will be compared with the virtual camera number in the data file, and **$error** will be assigned a warning message if the camera numbers differ.

This parameter will be used as an output parameter if it was undefined (or zero) on input. Then, **cam.virt** will return the *virtual* camera number used, as a real value in the range 1 to 32. (The physical camera number is contained in the **cam.cal[]** data array [see below].)

threshold Output parameter that receives the real value used to set the V.THRESHOLD system parameter.

backlight Output parameter that receives the real value used to set the V.BACKLIGHT system switch. The value will be either TRUE (-1) or FALSE (0).

to.cam Output parameter that receives the transformation value representing the position and orientation of the vision camera in the robot frame of reference.

This transformation must be combined with the transformation returned by the vision system whenever the robot is to be moved to an object in the field of view. That is, for example, for a camera fixed over the workspace the robot could be moved to the location defined by

```
to.cam:vis.loc
```

where **vis.loc** is the transformation returned by a VLOCATE instruction.

NOTE: This example applies only if the camera is fixed over the workspace. For any other camera mounting, refer to the *AdeptVision User's Guide* for the details of how to use the **to.cam** transformation.

cam.cal[ ] Output parameter that receives the area camera calibration data that is used in the VPUTCAL instruction (see above). This parameter must be a real-array variable. (For details on the elements of this array, see the description of the VPUTCAL instruction in the *AdeptVision Reference Guide*.)

The following output parameters return transformation matrices for dealing with perspective distortion. Each of these parameters must be a two-dimensional real-valued array variable. (If the calibration data file being read does not contain perspective data, these arrays receive simple scale transformations.)

NOTE: Refer to the *AdeptVision Reference Guide* for more information on perspective distortion in general, and on these arrays in particular.

pmm.to.pix[,] Output parameter that receives the millimeter-to-pixel transformation matrix.

pix.to.pmm[,] Output parameter that receives the pixel-to-millimeter transformation matrix.

pmm.to.mm[,] Output parameter that receives the transformation from real object space (in millimeters) to simple object space (in millimeters).

For efficiency, most vision tools are positioned with coordinates in simple millimeter space, and some tools (such as blob analysis) return coordinates in this space. The **pmm.to.mm[,]** transformation may be used to precisely *position* vision tools (such as VWINDOW and the graphics instructions) that do not automatically correct for perspective distortion.

NOTE: Using this correction is appropriate only when the perspective distortion is significant. Using the array **mm.to.pmm[,]** (see below) to correct the *results* of vision tools, however, may be desirable for increased precision even when the perspective distortion is slight.

mm.to.pmm[,] Output parameter that receives the transformation from simple object space (in millimeters) to real object space (in millimeters).

This array can be used to correct for perspective distortion in the *results* from vision tools that do not automatically convert the coordinates with the transformation **pix.to.pmm[,]** (for example, this is the case with blob or prototype locations).

Special Considerations

The subroutine **load.area** enables the VISION system switch, restores the stored settings of the V.THRESHOLD system parameter and the V.BACKLIGHT system switch, and restores the vision calibration data via the VPUTCAL instruction.

The subroutine **load.area** assumes that the calibration data file has the specific format used by the Adept area vision calibration program, ADV_CAL. The calibration program writes a format version number in the data file—this version number must agree with the one expected by **load.area**, or the loading is not performed.

LOADBELT

Description

Load belt calibration data for use by an application program

Disk File Name

LOADBELT.V2

Program Name

load.belt

Function

Load conveyor tracking calibration from a disk file created by the Adept belt calibration program, BELT_CAL.

Abstract

The Adept belt calibration program BELT_CAL, writes data to a disk file. An application program can read that data from the disk and restore it in the system by calling the subroutine **load.belt** with the appropriate argument list.

This section describes the procedure for including the subroutine in your application program.

Requirements

The items listed below are needed to utilize the **load.belt** subroutine in your application program.

- Robot system with the conveyor tracking feature
- Your application program (not provided by Adept)

Procedure

1. Load the program:[1]

   load \util\loadbelt

2. Use the system program editor to enter your application program into the system memory, or load your program from a disk file. Make sure the program calls the subroutine **load.belt** (see below).
3. Place a diskette to receive your application program in disk drive A.
4. Store your application program and the **load.belt** subroutine in a disk file by typing the command:

   store a:*file_name = name_of_your_main_program*

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

where *file_name* represents the name you want to assign to your disk file. *name_of_your_main_program* represents the name of the top-level program in your application program, which must logically refer to all the other routines for the application.

Alternatively, type the command

store a:*file_name*

to store *all* the programs and data in the system memory into the specified file.

Programming Considerations

The **load.belt** subroutine reads belt calibration information from a disk file, executes the following instructions to initialize the conveyor tracking system, and returns the calibration data to the calling program.

```
ENABLE BELT
FOR i = 1 TO 6
    IF cal[i] THEN
        DEFBELT %b[i] = nom[i], i, 32, sf[i]
        WINDOW %b[i] = win1[i], win2[i]
    END
END
```

After the routine has processed the calibration data file, the V$^+$ system is ready to have the robot track the calibrated belt(s). All of the calibration data is returned to the calling program for completeness. Normally only the belt variables (**%b[]**) and the locations defining the window limits (variables **win1[]** and **win2[]**) will be used by the application program.

The application should call the subroutine before the robot moves to a belt location. The subroutine normally needs to be called only one time per operating session. The CALL instruction must have the following format:

```
CALL load.belt($file, cal[], %b[], sf[], nom[], win1[],
               win2[], $error)
```

where the program arguments have the interpretations described below.

NOTE: The variable names used here are for explanation purposes only. Your application program can use any variable names you want. If you do not want to use a belt variable array, you must explicitly include appropriate DEFBELT and WINDOW instructions in the application program (see above).

$file Input parameter specifying the disk file containing the belt calibration data. This parameter can be a string constant, variable, or expression and can include a directory path in the form:

```
unit:directory_path\name.extension
```

The file name and extension must have the form BELT*nnn*.DAT (where *nnn* is the data set number) if the data file was created by the Adept conveyor tracking calibration program BELT_CAL.

$error If no error occurs while **load.belt** is executing, this parameter receives an empty string ("") and the output parameters described below receive the indicated values. If an error occurs, **$error** receives a string describing the error and the other output parameters are not defined. The calling program should check this variable before proceeding.

Each of the following output parameters is an array containing up to six elements. Element #1 pertains to the conveyor belt connected as ENCODER 1; element #2 pertains to the belt connected as ENCODER 2; etc.

See the "Conveyor Tracking" chapter in the *V$^+$ Language User's Guide* for more complete descriptions of the data items listed below.

cal[ ]	Real array indicating the calibration status of the belts. Each element contains the value TRUE (-1) if the corresponding belt has been calibrated or FALSE (0) if the information contained in the corresponding element of the other data arrays is invalid.
%b[ ]	Belt variable array. A belt variable must be combined with one or more transformations whenever the robot is to be moved to a location on a conveyor belt. For example, the robot could move to the location defined by: `%b[1]:start` where **start** is a transformation variable defined relative to the belt variable **%b[1]**.
sf[ ]	Real array containing belt scale factors. These constants relate motion of the corresponding belt with the output of its belt encoder. The values represent the distance the belt moves (measured in millimeters) for each count from the belt encoder.
nom[ ]	Transformation array containing the nominal locations of the belts.
win1[ ]	Transformation array containing the locations used to define one extreme of the working windows on the belts.
win2[ ]	Transformation array containing the locations used to define the other extreme of the working windows on the belts.

Special Considerations

The subroutine **load.belt** enables the BELT system switch, restores the belt variables used by the robot to track the conveyors and restores the working windows on the belts (see above).

The subroutine **load.belt** assumes that the calibration data file has the specific format used by the Adept calibration program, BELT_CAL. The calibration program writes a version number in the data file—that version number must agree with the one expected by **load.belt**. Otherwise, the loading is not performed.

When the working windows for the belts are defined, no program is primed for execution in the event of a window violation. Thus, your program may include a WINDOW instruction for each belt to specify the name of a program to be executed if a window error ever occurs while the robot is tracking the belt. That is, your WINDOW instructions should have the format:

```
    WINDOW %b[i] = win1[i], win2[i], program_name
```

where **win1[i]** and **win2[i]** are the window defining locations returned by **load.belt**, and *program_name* represents the name of the program you want to have executed if a belt-window violation occurs for belt **i**.

LOADICON

Description

Load graphic icons for use by an application program

Disk File Name

LOADICON.V2

Program Name

load.icon

Function

Load graphic icons into memory from a disk file created by the Adept graphic icon editing utility, EDITICON.

Abstract

The Adept graphic icon editing utility EDITICON, writes data into a disk file. An application program can read the data from the disk and store it in graphic system memory by calling the subroutine **load.icon** with the appropriate argument list.

This section describes the procedure for including the subroutine in your application program. See the section on the icon editing utility, EDITICON, for information on how to create icon data files.

Requirements

The item listed below is needed to utilize the **load.icon** subroutine in your application program. All of these (except as indicated) were supplied with your Adept system.

- Adept graphics-based system controller with the AdeptVision option
- PC-style keyboard and high-resolution graphics monitor (system option)
  or
  AdeptWindows PC with Ethernet connection
- Your application program (not provided by Adept)

Procedure

1. Load the program:[1]

   load \util\loadicon

2. Use the system program editor to enter your application program into the system memory, or load your program from a disk file. Make sure the program calls the subroutine **load.icon** (see below).
3. Place a diskette to receive your application program in disk drive A.
4. Store your application program and the **load.icon** subroutine in a disk file with a command with the form

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

store a: *file_name = name_of _your _main_program*

where *file_name* represents the name you want to assign to your disk file. *name_of_your_main_program* r represents the name of the top-level program in your application program, which logically refers to all the other routines for the application.

Alternatively, use the command

store a:*file_name*

to store *all* the programs and data in the system memory into the specified file.

Programming Considerations

The **load.icon** subroutine reads the icon definition information stored in a disk file. If the data are found to be valid, the following instructions are executed for each icon found in the file.[1]

```
FOPEN (glun) $name , "/ICON /ARRAY", max.index,
             "/SIZE", dx, dy, "/POSITION", x, y
WRITE (glun) graphic_data_from_the_file
FCLOSE (glun)
```

After the routine has successfully processed the icon data file, the graphics system is ready to accept GICON instructions that reference the loaded icons.

Your application program should call the subroutine before the icons are referenced. The subroutine normally needs to be called only one time per operating session. The CALL instruction must have the following format:

```
CALL load.icon($file, dlun, glun, error)
```

where the program arguments have the interpretations listed below.

NOTE: The variable names used here are for explanation purposes only. Your application program can use any variable names you want.

$file Input parameter that passes the specification of the disk file containing the icon definition data. This parameter can be a string constant, variable or expression.

The file specification can have the form:

```
phy_device>unit:directory_path\name.extension
```

where the following elements are optional: *phys_device*>, *unit:*, *directory_path*\, and *.extension*. The defaults for the first three items are taken from the current DEFAULT setting. The default extension is .DAT.

dlun Input parameter that specifies the disk logical unit to use for opening the data file.

[1] The variables **$name**, **max.index**, **dx**, **dy**, **x**, and **y** are defined from the data in the file. All these variables are declared AUTO by **load.icon**, and thus cannot be accessed by other programs.

The disk logical unit must not be attached by the current program task. The program **load.icon** leaves the disk logical unit detached.

glun	Input parameter that specifies the graphics logical unit to use for writing the icon data into graphics memory. The graphics logical unit must be attached by the current program task, but no window may be open on the logical unit. The program **load.icon** leaves the graphics logical unit attached, with no open window.
error	Output parameter that receives the status of the load operation. This parameter must be a real-valued variable. If no error occurs in the load process, **error** returns zero. If an error occurs, **error** returns a standard V$^+$ error code describing the error. In that case, one or more of the icons defined in the disk file may not have been loaded. Thus, your calling program should check the variable **error** before proceeding.

Special Considerations

The subroutine **load.icon** assumes that the calibration data file has the specific format used by the Adept graphic icon editing utility, EDITICON. The icon editing utility writes a version number in the data file that must agree with the one expected by **load.icon**.

You can use the V$^+$ FLIST command to display information in an icon data file. The information includes the creation date of the file and a list of the icons defined in the file.

LOADTCH

Description

Load touchscreen calibration data for use by an application program

Disk File Name

LOADTCH.V2

Program Name

load.touch

Function

Load touchscreen calibration from a file created by the Adept touchscreen calibration program, TOUCHCAL.

Abstract

The Adept touchscreen calibration program TOUCHCAL, writes data to a disk file. An application program can load that data from the disk by calling the subroutine **load.touch** with the appropriate argument list.

This section describes the procedure for including the subroutine in your application program.

Requirements

The items listed below are needed to utilize the **load.touch** subroutine in your application program.

- Adept controller with VGB module in the system
- High-resolution graphics monitor (system option)
- Installed touchscreen (third-party option)
- PC-style keyboard (system option)
- The touchscreen rather than a trackball or mouse must be connected to the VGB module as the pointing device

Procedure

1. Load the program:[1]

   load \util\loadtch

2. Use the system program editor to enter your application program into the system memory, or load your program from a disk file. Make sure the program calls the subroutine **load.touch** (see below).

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

3. Store your application program and the **load.touch** subroutine in a disk file by typing a command with the form

   store a:*f**ile_name = name_of_your_main_program*

   where *file_name* represents the name you want to assign to your disk file. *name_of_your_main_program* represents the name of the top-level program in your application program, which must logically refer to all the other routines for the application.

   Alternatively, type the following command:

   store a:*file_name*

   to store *all* the programs and data in the system memory into the specified file.

Programming Considerations

The application should call the subroutine before interaction with the touchscreen is required. The subroutine needs to be called after system memory is ZEROed or the controller is first turned on. The CALL instruction must have the following format:

```
CALL load.touch($file, $error)
```

where the program arguments have the interpretations described below.

> **NOTE:** The variable names are for explanation purposes only. Your application program can use any variable names.

$file Input parameter specifying the disk file containing the touchscreen calibration data. This parameter can be a string constant, variable, or expression and can include a directory path. Its value can have the form:

```
unit:directory_path\name.extension
```

However, the file name and extension must have the form TOUCH*nnn*.DAT (where *nnn* is the data set number) if the data file was created by the Adept touchscreen calibration program TOUCHCAL.

$error If no error occurs during the loading of loadscreen calibration, **$error** returns an empty string (“”) and the touchscreen calibration is applied. If an error occurs, **$error** returns a string describing the error and the touchscreen calibration is not applied. The calling program should check the variable **$error** before proceeding.

Special Considerations

The subroutine **load.touch** assumes that the calibration data file has the specific format used by the Adept calibration program,TOUCHCAL. The version number in the data file must agree with the one expected by **load.touch**. Otherwise, the loading is not performed.

PROFILER

Description

Task profiler utility

Disk File Name

PROFILER.V2

Program Name

a.profiler

Function

Display the amount of CPU time being used by executing system and users tasks.

Abstract

The V$^+$ operating system is a multitasking system that concurrently executes system and user tasks. Each task is allowed to execute in one or more of sixteen time slices according to the priority assigned to each task in each of the sixteen time slices.

The default priorities assigned to each user task in each time slice may be defined by the user using the program CONFIG_C. The default priorities may be superseded with an **EXECUTE** instruction. The profiler displays a series of bar graphs showing how much time each task is taking in each time slice. This display helps you define a priority scheme that assures CPU time for your critical tasks and runs noncritical tasks on an as- time-is-available basis.

Using the task profiler requires a thorough understanding of V$^+$ task execution. See the section "Scheduling of Program Execution Tasks" in the *V$^+$ Language User's Guide*.

Requirements

You need the following items to run the PROFILER.

- Adept graphics based controller
- PC-style keyboard and high-resolution graphics monitor (system option)
  or
  AdeptWindows PC with Ethernet connection
- V$^+$ systems prior to version 11.2 must have the V$^+$ Version 11 Extensions option installed.

In order to display meaningful data, you should have application programs LOADed and ready to EXECUTE.

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 as a program task other than #0.

Procedure

1. Load the program:[1]

   load \util\profiler

2. Execute the program:

   execute a.profiler

   (Or type **execute 1 a.profiler** to execute the program as task #1. Any task can be used. See **Special Considerations** for related information.)

3. The profiler utility displays the default bar graph and pull-down menus. This display is active and shows current CPU usage.

4. To exit the program, choose the **Close** icon or **Exit Profiler** in the Profiler menu.

5. The program can be deleted from system memory with the following commands (type **kill 1** in place of **kill** if the program was executed in task #1):

   kill
   delete a.profiler

Program Operation

The profiler displays CPU usage on two different bar graphs, the % CPU Time graph or the Time Slice graph. The % CPU Time graph is a bar graph that shows the percent of time used per 16 ms major cycle by the system and user tasks (see Figure 2-1).

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

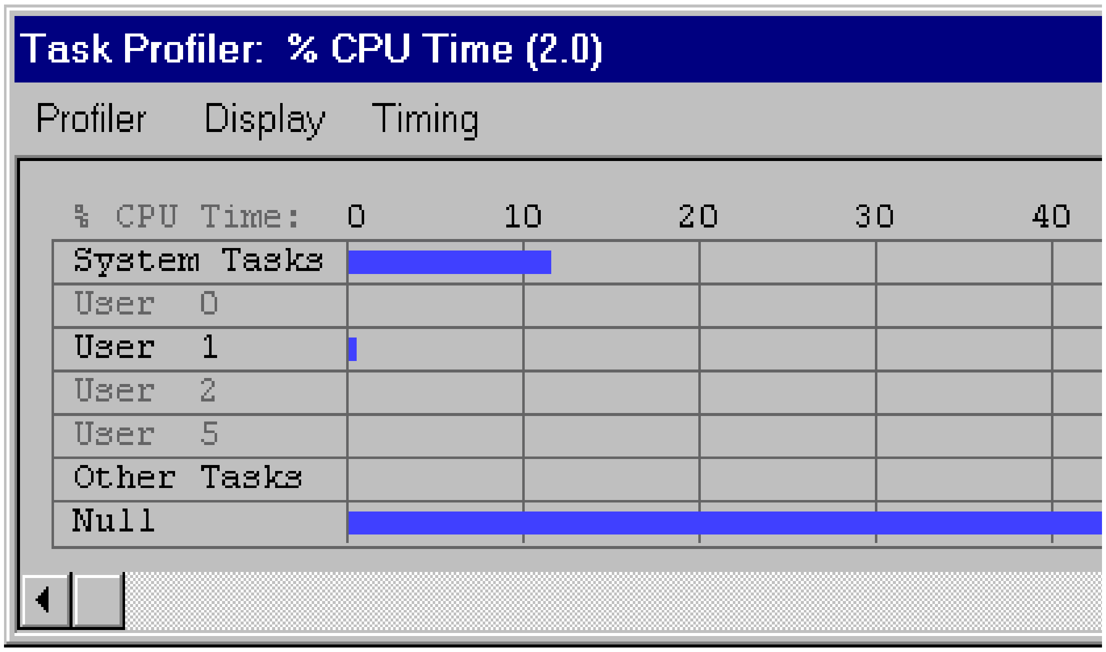


Figure 2-1. Percent (%) CPU Time Graph

Figure 2-1 shows that the running system is allocating approximately 12 percent of its time to system tasks, zero (0) percent to user task 0, 1percent to user task 1, zero (0) percent to user task 2, and 88 percent to the null task. (The null task runs whenever no other tasks are available to run.)

The Time Slice graph is a bar graph that shows the CPU time that each task has used in each time slice. Data is shown for the last three 16 ms major cycles. These three major cycles are shown as red, white, and blue bars on each task line (see Figure 2-2).

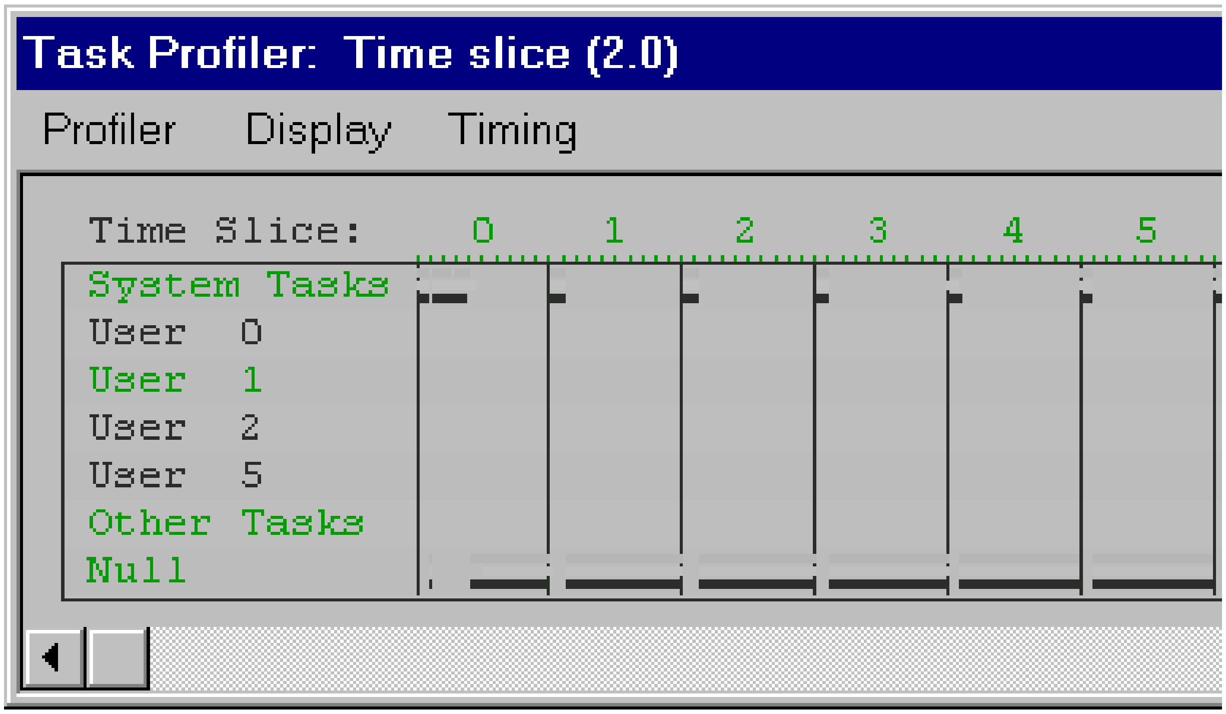


Figure 2-2. Time Slice Graph

Figure 2-2 shows a system running four concurrent user tasks. The red, white, and blue bars show the amount of time each task has used in each time slice for the previous three intervals. The interval time (2.0 seconds in Figure 2-2) is set with the Timing menu items.

The Time Slice graph appears after choosing the **Time Slice** menu item from the Display menu, and when the PROFILER is run in a second task. (When the time-slice graph is active, the **Time Slice** menu item changes to **%CPU Time**.)

A graph can be displayed in either *update* or *freeze* mode. In update mode, the graph is refreshed every 0.5, 1, 2, or 5 seconds. In freeze mode, the graph is held static. In freeze mode, the display can be updated by choosing the **Snapshot** item. (These items are selected from the Timing menu.)

Choosing Displayed Tasks

The Display menu allows you to specify which tasks are displayed. Choosing **All System Tasks** shows all the individual V^+ system tasks on the bar graph. If **All System Tasks** is not chosen, all system task usage is displayed on one line marked System Tasks.[1]

Choosing **All User Tasks** displays all the user tasks available to your system. If **All User Tasks** is not chosen, only tasks with a program on the execution stack are displayed.

[1] Selected options are marked by a ✔.

Program Menus

The Profiler Menu

The Profiler menu displays help for the profiler utility and exits the utility program. (To close an information pop-up window, click anywhere in the pop-up window.)

Profiler Menu Item	**Action**
About Profiler	Display program version/revision data.
Display Menu Help	Display basic information on the Display menu items.
Timing Menu Help	Display basic information on the Timing menu items.
Exit Profiler	Exit the profiler program. (Choosing the **Close** icon also exits the program.)

The Display Menu

The Display menu determines which display is presented and how many lines of information are graphed.

Display Menu Item	**Action**
Time Slice % CPU Time	This menu item toggles between the two display types. When **% CPU Time** is chosen, the program displays the graph shown in Figure 2-1. When **Time Slice** is chosen, the graph shown in Figure 2-2 is displayed.
All System Tasks	When chosen, each system task is displayed on its own line. When not chosen, all system task time is accumulated into one line labeled System Tasks.
All User Tasks	When chosen, all user tasks are graphed whether or not they are in use. When not chosen, only the tasks that have a program on the execution stack are displayed.
Refresh	Forces a redraw of the profiler graphs. (Use this, for example, if **All User Tasks** is not chosen and a new user task has become active.)
Display Values	When **% CPU Time** is displayed, this item can be chosen to add a column to the display to show the lengths of the graph bars.

The Timing Menu

The Timing menu determines how often the graphs are updated.

Timing Menu Item	Action
Snap Shot	Put the display in freeze mode and update the graphs.
Interval 0.5	Update graphs at 0.5-second intervals.
Interval 1	Update graphs at 1-second intervals.
Interval 2	Update the graphs every 2 seconds.
Interval 5	Update the graphs every 5 seconds.
Freeze	Do not update the graphs (freeze mode).

Special Considerations

Changing the task priority scheme can seriously degrade the performance of your system. Make sure you have a thorough understanding of how the Adept multitasking system works before changing task priorities. Read the first chapter of the *V$^+$ Language User's Guide* before making changes to the task priorities.

In order to get a true picture of CPU usage and minimize the affect of the executing PROFILER program, you will generally want to run the profiler in a low priority task or with a low priority specified as a parameter in the EXECUTE command. See the *V$^+$ Operating System Reference Guide* for details.

If you would like to see the Time Slice graph and % CPU Time graph at the same time, you can execute the PROFILER in two different tasks.

The PROFILER program defines may real-valued and string global variables with the prefix ‘pf.’ that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file PROFILER.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

SAFE_UTL

Description

Test program for the Adept robot Manual Mode Safety Package (MMSP) option

Disk File Name

SAFE_UTL.V2

Program Name

a.safe_utl

Function

Tests components of the Manual Mode Safety Package (MMSP) and sets a software flag to mark the robot as “commissioned” if all the tests pass.

Requirements

The items listed below are needed to run the SAFE_UTL program.

- Adept system controller
- Adept robot with the MMSP (Cat 3) option
- V^+ system version 11.3 or later

Procedure

Detailed procedures for running SAFE_UTL for a particular robot are given in the instruction handbook for that robot (e.g., the *AdeptThree-XL Robot Instruction Handbook* details commissioning an AdeptThree-XL Robot).

1. Load the program:[1]

   load \util\safe_utl

2. Execute the program in task #1 (or higher):

   execute 1 a.safe_utl

   The following menu is displayed:

```
*** Adept MMSP Test Program (Version 12.1X) ***

             Robot 1: xxx-xxxx

             Language Selection

             0 => Exit
             1 => English
             2 => Deutsch
             3 => Italiano
             4 => Francaise

             Enter Selection:
```

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

After a language selection is made, the following is displayed (in the selected language):

```
WARNING: The current robot will be marked as “not
commissioned", and will be marked as “commissioned" only
after passing “all tests".

Do you want to continue (Y/N)?
```

If the system has previously been "commissioned", responding Y marks the system as "not commissioned". The robot cannot be operated in manual mode until all tests are completed and passed.

If the response is N, the system is left unchanged and the program halts.

3. The program displays the following menu and prompts the user to select an option:

```
*** Adept MMSP Test Program (Version 12.1X) ***
 Copyright (c) 1997 by Adept Technology, Inc.

     Robot 1: xxx-xxxx   *Not Commissioned*

          0 => EXIT
          1 => All Tests
          2 => Accelerometer
          3 => B+ Amp Voltage Restrict
          4 => VFP Switches and Buttons
          5 => MCP E-STOP Functions
          6 => Brake Holding Force
       Enter Selection:
```

4. See the robot instruction handbook for your system for details on running the specific tests.
5. To exit the program, choose EXIT in the program menu.
6. The program can be deleted from system memory with the following commands :

   kill 1
   delete a.safe_utl

Special Considerations

The SAFE_UTL program defines may real-valued and string global variables with the prefix ‘ca.’, ‘opt.’, ‘ts.’ and ‘tt.’ that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file SAFE_UTL.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

SECURE

Description

Limit access to a file or a diskette

Disk File Name

SECURE.V2

Program Name

a.secure

Functions

Limit access to a diskette
Limit access to a file

Abstract

The SECURE utility works with two separate entities: disks and files.The process of securing a disk is entirely different from the process of protecting a file.

Securing a Disk The Adept disks are usually formatted to be IBM PC compatible. This can be a disadvantage if there is information on the disk that you want to restrict. If this is the case, you would want to secure the disk.

There are two objectives in securing a disk. One is to prevent someone from using an IBM PC to examine the contents of the disk. The other objective is to prevent someone from using the disk on more than one Adept system.

When a disk is secured, it is no longer IBM compatible; however, a secured disk can be read on any Adept system (if it is secured for all systems)—or only by a specific Adept system. If you secure to a specific system, you must provide the product type and the serial number of the controller.

> **NOTE:** Once a disk is secured for a specific system, it cannot be unsecured—or secured for another system. It can only be reformatted (with a complete loss of data).

Protecting a File Individual files can be protected from being changed or viewed. SECURE provides the ability to set either of two file attributes: read-only or protected.

A read-only file can be read, but can't be changed; these files have the R attribute in their directory listing. A read-only file can be used to allow operators to view a file, but not modify it; this allows users to become familiar with your programming practices without endangering the integrity of your programs.

A protected file cannot be viewed, it can only be LOADed or EXECUTEd. Protected files have the P attribute in their directory listing. The protected attribute not only applies to the file, it also applies to the contents of the file. Any programs in the file, after being loaded into memory, can't be changed. A protected file can be used to prevent users from examining or modifying a file (or its programs).

The DISKCOPY utility must be used to make copies of a protected file.

Procedure

1. Load the program:[1]

   load \util\secure

2. Execute the program:

   execute a.secure

   (Or type **execute 1 a.secure** to execute the program as task #1.)

3. Follow the prompts provided by the program.

4. The program can be deleted from system memory with the following commands (type **kill 1** in place of **kill** if the program was executed in task #1):

   kill
   delete a.secure

Special Considerations

Although a secured disk is totally compatible with the V+ system, you should be aware that securing a disk causes it to be no longer PC compatible.

The SECURE program defines may real-valued and string global variables with the prefix 'sec.' that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file SECURE.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

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

SFUTIL

Description

Utility for working with data for the MOVEF/MOVESF instructions

Disk File Name

SFUTIL.V2

Program Name

a.sfutil

Functions

Experimentally determine the parameters for the dynamic model of an AdeptOne-MV, AdeptThree-MV, AdeptOne-XL or AdeptThree-XL robot, which are used by the MOVEF and MOVESF instructions to predict the speed and acceleration of the robot.

Display which sets of parameters for the dynamic model are initialized.

Save and load all the configuration data for the robot (including the parameters for the dynamic model) to and from a disk file.

Save the dynamic model data for the robot to a V^+ system disk.

Abstract

For robot systems with the HyperDrive option, the MOVEF and MOVESF instructions drive the robot through a three-segment pick-and-place motion at the optimum speed and acceleration. These instructions rely on a built-in dynamic model of the robot in order to estimate the maximum speed and acceleration that drive the robot in the shortest amount of time without violating the maximum motor torque specifications. For robot applications that require high-speed motions, these instructions allow programs to be written that operate the robot at near-optimum cycle times without the need for manual speed tuning.

Although the form of the dynamic model of the robot is fixed by the design of the robot, the parameters for the model depend on the specific robot and its tooling and payload. The SFUTIL program provides an automatic method for experimentally deducing the parameters of the dynamic model, based upon the results of moving the robot at various speeds in a small region of the workspace.

Because of manufacturing variations between robots, this procedure must be performed on each robot. Also, the procedure must be done for each combination of tooling and payload that the robot will carry. This procedure is simple to execute, must be done only one time for each payload, and requires only about 10 minutes to determine one set of dynamic model parameters.

Once a set of dynamic model parameters is determined, this utility allows you to permanently store the computed values, along with a short description, on the system disk. This utility also displays the descriptions of the data sets that have been stored and provides for writing/reading the full set of configuration data for the robot to/from a disk file.

Requirements

You need the following items to run SFUTIL:

- AdeptOne-MV, AdeptThree-MV, AdeptOne-XL or AdeptThree-XL robot system with the HyperDrive option
- Robot end effectors and payloads that are to be modeled
- Copy of the system diskette if the system is booted from a floppy diskette. (If required, use the DISKCOPY utility program to make a copy of the distribution system diskette.)

Before you can run SFUTIL:

- The robot must be calibrated and COMP mode must be selected on the manual control pendant.
- No program must be running in the main execution task (#0) or in execution task #1.

Procedure

1. Load the program:[1]

   load \util\sfutil

2. Execute the program:

   execute a.sfutil

3. Select an option you want from the menu displayed (see below).
4. Follow the prompts provided by the program (see below).
5. The utility program can be deleted from system memory by typing the following commands:

   kill
   delete a.sfutil

Program Prompts

The SFUTIL utility program displays a menu of selections to the user. The following sections describe the menu items.

NOTE: To exit the program at any prompt, press CTRL+Z.

`0 => Exit to system monitor`

Choose this item to exit the program and return to the V$^+$ monitor.

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

`1 => Determine robot’s dynamic model parameters`

This menu selection initiates the procedure for automatically determining the parameters of the dynamic model of the robot. This procedure moves the robot through a series of predefined motions (relative to the user-defined starting location) and compares the actual performance of the robot to that predicted by the dynamic model.

NOTE: Before executing this procedure, the end effector and payload that are to be modeled must be attached to the robot.

The following sequence of events is initiated when this procedure is selected.

1. A message is displayed that describes the purpose of the procedure, and you are asked to confirm that the procedure should be performed.
2. A message is displayed describing how much each joint of the robot will be moved during the procedure, and you are asked to position the robot in a clear zone.
3. After confirmation, the robot moves at a slow speed through all the motions that will be performed by the automatic procedure. This allows you to verify that the required clear area is free of any obstacles.

CAUTION: If the robot moves close to an obstacle, you should press the E-Stop button on the Manual Control Pendant to stop the motion.

4. After confirmation, the calibration procedure is initiated and the robot is automatically driven through the motions previously demonstrated. In most cases, each motion is repeated more than once, at various speeds.

WARNING: During the automatic procedure, the robot is moved at both low speed and *very high speed*. Also, the robot may not move for several seconds (while a lengthy computation is performed). *Such delays are normal and do not indicate that the procedure has completed all of the robot motions.*

5. After the automatic process is completed, a table of computed dynamic model parameters is displayed. These values do not have to be written down. They are presented only for diagnostic purposes.
6. You are asked if you want to store the computed set of dynamic parameters in memory. If so, the list of parameter sets currently defined is displayed. After you specify a brief description and which set of parameter values is to be overwritten, the new values are stored in the internal data structures of the V^+ operating system.
7. Finally, you are asked if you wish to save the new data onto the system disk. If so, the new data is permanently saved on the system disk.

NOTE: Do not alter the original distribution diskette supplied by Adept. Instead, you should first make a copy of the distribution diskette (with the DISKCOPY utility), and then store your calibration data to that copy.

If the computed values are not written into V^+ memory, the results of the procedure are ignored. If the values are not written to the system disk, the results are lost when the controller is turned off.

`2 => Show description of defined dynamic models`

This selection shows which of the sets of dynamic parameters have been defined by executing the automatic procedure. A list of the possible parameter sets is displayed, followed by either a short description of each data set or the text "`*Not initialized*`".

`3 => Save robot specifications to a disk file`

This selection writes all the configuration data associated with the (primary) robot to a disk file. This data includes the sets of dynamic parameters. Depending on the type of robot, the data may also include other values that can be specified by the user, such as the software joint-limit stops.

The data is written in the disk file in a readable ASCII format that can be kept as a permanent record of the data and can be used to reload the data.

`4 => Load robot specifications from a disk file`

This selection reads a data file that was created by the previous menu selection, and overwrites the data in memory with the data contained in the disk file. This selection needs to be executed only when you are upgrading your robot operating system to a new version or revision and want to reinstate all the robot specifications generated previously, or when you want to change to different sets of parameters.

After the data has been read, the following menu selection must be executed in order to permanently store the data on the system disk.

`5 => Save system specifications to system disk`

This selection stores the robot specification information that is contained in the system memory to the system disk. If the specification information is contained only in the system memory, the information will be lost when the controller is turned off. After the system disk has been altered, the specification information is preserved even when the controller is turned off and the system is rebooted.

> **NOTE:** You should not alter the original distribution diskette supplied by Adept. Instead, you should first make a copy of the distribution diskette (with the DISKCOPY utility), and then store your calibration data to that copy.

Special Considerations

If you make any change to your V^+ system disk, you should clearly mark the system disk to indicate that it has been modified for your specific robot configuration. It is advisable to make a backup copy of the modified system disk. The DISKCOPY utility can be used for that purpose.

This utility defines many real-valued variables with the prefixes **sf.** and **sp.** that must not be modified by the user or by other programs. (All these variables are deleted from memory when the program completes execution normally, and when a DELETE command is used to delete the program from memory as described earlier.)

The disk file SFUTIL.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 to another with the FCOPY monitor command or the disk-copy utility program (DISKCOPY). The disk-copy program can be used to make a backup copy of the entire distribution diskette.

SPEC

Description

Robot configuration utility

Disk File Name

SPEC.V2

Program Name

a.spec

Functions

The SPEC program allows customizing robot parameters for a particular application. This description documents the restricted access mode of SPEC seen with Adept robots and password-protected 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 following robot parameters can be changed with the restricted access mode of SPEC:

- The digital signals that are used to control the robot hand or end-effector
- The positional tolerances to be used at the ends of motions
- The robot (software) joint limits
- The program-controlled motion speed above which accelerations are not to be scaled

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 can be used to make 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)

Before you run SPEC:

- The system must have been booted from the *actual* system disk that will be modified during the configuration procedure.

Procedure

1. Load the program:[1]

   load \util\spec

2. Execute the program:

   execute 1 a.spec

3. Choose an option from the menu displayed (see below).
4. Follow the prompts provided by the program (see below).
5. You can delete the SPEC program from the system memory with the following commands:

   kill 1
   delete a.spec

Menu Structure

```
Exit to system monitor
Enter password to increase access[2]
AdeptModules quick setup[3]
Perform hardware diagnostics
Edit robot specifications
    Exit to main menu
    Edit robot initialization specs
        Exit to robot menu
        Hand OPEN signal
        Hand CLOSE signal
    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 general motion specs
        Exit to robot menu
        Upper speed limit for SCALE.ACCEL (%)
Load robot specifications from a disk file
Save robot specifications to a disk file
Save ALL specifications to system disk
Change robot number[4]
Switch to external encoder specifications[5]
```

[1] Use the correct drive and path specification if the file is stored on another drive or subdirectory.
[2] Password-protected robots only
[3] Only for systems with AdeptModules devices
[4] Multiple-robot systems only
[5] Systems with external encoders only

Program Prompts

The SPEC robot configuration utility program displays the following screen (you may notice some differences due to your system configuration):

```
    ***  ADEPT ROBOT SPECIFICATION PROGRAM  ***
                 (Version 12.1 or later)

Copyright (c) 1988-1997 by Adept Technology, Inc.

Servo code version: 12.1 Servo rate: 1000 Hz

         ROBOT 1: Adept SCARA Robot Module.
     Restricted access mode (formerly CONFIG_R)

    0 => Exit to system monitor
    1 => Perform hardware diagnostics
    2 => Edit robot specifications
    3 => Load robot specifications from a disk file
    4 => Save robot specifications to a disk file
    5 => Save ALL specifications to system disk
    6 => Change robot number
    7 => Switch to external encoder specifications

         Enter selection:
```

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.

0 => Exit to System Monitor

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

1 => 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.

             Read-only mode access mode.

         Press P to toggle power, Q or 0 to quit.

Motor/Board/Channel    1/B1/C1   2/B1/C2   3/B1/C3  4/B1/C4
Encoder position            65        17     30914       10
Last zero-index          -1045      3188     36438      -88
```

```
Zero-index delta        2000      8000      7680         0
Home                      ON       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             0.00      0.00      0.00      0.00
DAC count                  0         0         0         0
```

`2 => Edit robot specifications`

The following menu items are displayed at this level:

```
0 => Exit to main menu
1 => Edit robot initialization specifications
2 => Edit joint motion specs
3 => Edit general motion specs
```

These menu item selections perform the following (refer to the next section for a detailed description of these menu items):

- Change the default gripper OPEN and CLOSE signals
- Change the joint limits and COARSE and FINE nulling tolerance for each joint
- Change the percentage speed above which the SCALE.ACCEL switch has no effect

`3 => 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.

`4 => 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 as a readable ASCII text file or as 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.

`5 => 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 turned on and booted from disk.

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, before you use this procedure, you should make a copy of your V$^+$ system disk (with the DISKCOPY utility). Then, you should 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 encoder).

`6 => Change robot number`

If your system has multiple robots, selecting this item allows you to access data for another robot.

`7 => Switch to external encoder specifications`

If your system supports external encoders for conveyor tracking, you can select this item to edit the encoder parameters. See the *AdeptMotion VME Developer's Guide* for more information.

Detailed Description of Program Operations

The following items can be edited by moving through the SPEC menu as documented below. After changing any of these items, the data can be written to the system disk or to a disk data file that can be used later to apply your specific configuration to a new V$^+$ system disk.

```
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 if your application requires special signals to control the robot gripper.

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), any of the 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.

```
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 in the *V$^+$ Language Reference Guide* of the COARSE and FINE program instructions.

```
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 prevent the robot from moving into undesired regions.

```
Upper speed limit for SCALE.ACCEL (%)
```

Defines the percentage of maximum program speed below which the SCALE.ACCEL switch takes 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, refer to the *V$^+$ Operating System Reference Guide*.

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. (All the string 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.)

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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

SQUEEZE

Definition

Disk file compression utility

Disk File Name

SQUEEZE.V2

Program Name

a.squeeze

Function

Compress a disk file by deleting all the comments and blank lines in the V$^+$ programs in the file.

Abstract

When a V$^+$ program is loaded into the system memory from a disk file, all the comments and blank lines in the program take up system memory. The squeeze program eliminates all the comments and blank lines from program files so the programs will require less memory (and so the progam file will require less disk space).

Requirements

You will need the following to run SQUEEZE:

- Adept system controller
- Program file(s) to be compressed
- Disk to receive compressed file(s)

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 as a program task other than #0.

Procedure

1. Load the program[1]

   load \util\squeeze

2. Execute the program

   execute a.squeeze

   (Or type **execute 1 a.squeeze** to execute the program as task #1.)

3. Enter the disk unit of the input file. Or you can simply press ENTER to select the default unit when one is shown.

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

4. Enter the name of the input file to be compressed. The file extension ".V2" is assumed if no extension is specified.

   An optional subdirectory specification can be included in the file specification. If no subdirectory specification is included, the file is assumed to be in the default subdirectory on the specified input disk.

5. Enter the disk unit for the output file or press ENTER to accept the default unit.

6. Enter the name of the output file to be created. If you simply press ENTER, the input file name with the extension ".SQU" is used.

   An optional subdirectory specification can be included in the file specification. If no subdirectory specification is included, the file will be created in the default subdirectory on the specified output disk.

7. If you want to stop, enter **n** (or just press ENTER) in response to the prompt

   ```
   Squeeze another file (Y/N)?
   ```

   If you want to process another file, enter **y** and repeat steps 3 through 7 for the next file to be compressed.

8. To delete the file compression program from the system memory type the commands:

   kill
   delete a.squeeze

   (You must type **kill 1** in place of **kill** if the program had been executed as task #1.)

Special Considerations

The compression program defines several real-valued and string variables with the prefix **sq.** that must not be modified by the user, or by other programs. (All the variables are deleted from memory when a DELETE command is used to delete the program from memory, as described above.)

The disk file SQUEEZE.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

TOUCHCAL

Definition

Touchscreen calibration utility

Disk File Name

TOUCHCAL.V2

Program Name

a.touchcal

Function

Calibrate an installed touchscreen for use as the system pointing device.

Abstract

A touchscreen can be used as the system pointing device, instead of a mouse or trackball. Since the resolution of a touchscreen is different from the standard trackball, a simple calibration procedure must be performed. After the calibration procedure has been completed, the data can be stored in a disk file for access by an application program.

Requirements

You must have the following items to run TOUCHCAL:

- Adept controller with a VGB module in the system
- High-resolution graphics monitor (system option)
- Installed touchscreen (third party option)
- PC-style keyboard (system option)

Before you can run TOUCHCAL:

- The touchscreen rather than a trackball or mouse must be connected to the VGB module. (You must know the type of touchscreen you are using.)
- 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 as a program task other than #0.

Procedure

1. Load the program:[1]

   load \util\touchcal

2. Execute the program:

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

execute a.touchcal

(Or type **execute 1 a.touchcal** to execute the program as task #1.)

3. The monitor window will display the touchscreen calibration main menu (see below).
4. After you exit the program, it can be deleted from the system memory by typing the following commands (type **kill 1** in place of **kill** if the program was executed as task #1):

   kill
   delete a.touchcal

Menu Options

`0 => EXIT to system monitor`

Select this option when you are finished with the calibration utility.

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

`1 => LOAD calibration data from disk`

Before calibration data can be loaded, a valid calibration file must have been created using option 2 or 3 (see below). The calibration file must have the name TOUCH*nnn*.DAT where *nnn* is the number of the stored calibration data set.

The program asks for a disk drive letter (A or C).

After you enter a drive designation, the program asks for a subdirectory specification.

After you enter the subdirectory specification the program asks for the number of the touchscreen calibration (0 to 999).

After you supply a calibration file number, the program displays the directory path it will use to load a calibration file. If the path and file name are correct, respond **y** and the file will be loaded.

`2 => STORE calibration data to disk`

Before calibration data can be stored, a valid calibration file must have been loaded using option 1, or new calibration data must have been created using option 3.

The program asks for a disk drive letter (A or C).

After you enter a drive designation, the program asks for a subdirectory specification.

After you enter the subdirectory specification, the program asks for a touchscreen calibration number. This number is used to create a file with the name TOUCH*nnn*.DAT, where *nnn* is the number you specify.

After you supply a calibration file number, the program displays the directory path it will use to store the calibration file. If the path and file name are correct, respond **y** and the file is created.

3 => CALIBRATE the touchscreen

The following sub-menu is shown:

```
0 => EXIT to MAIN MENU
1 => Elographics: Intelli-Touch
2 => Elographics: Accu-Touch

Enter Selection and Press Enter:
```

The Intelli-Touch screen uses Surface-Acoustic Wave (SAW) technology; the Accu-Touch is a resistive screen.

After you make your selection, the program presents a square in each corner of the monitor and asks you to touch the center of each square. After touching all the squares, you are then asked to check the screen calibration:

```
Move your finger around on the screen.

The pointer should track your finger within 0.25 inch (6 mm).

Do you want to calibrate again to improve the results (Y/N)?
```

If you respond with **n**, the program will proceed to store the calibration data in a disk file. See menu option 2 (STORE calibration data to disk) above.

Programming Considerations

The touchscreen calibration data must be loaded each time the controller is turned on. After the calibrating procedure has been done, the data can be stored in a disk file for access by an application program.

Your application program can CALL the routine in the file LOADTCH.V2 to load the touchscreen calibration.

The TOUCHCAL program defines may real-valued and string global variables with the prefix ‘tc.’ that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file TOUCHCAL.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

XMODEM

Description

File transfer utility

Disk File Name

XMODEM.V2

Program Name

a.xmodem

Function

Transfer disk files using the XMODEM protocol.

Abstract

XMODEM is a popular serial communication protocol that incorporates basic error checking, handshaking, and retransmission of corrupted data. This utility allows for XMODEM transfers between Adept controllers, between Adept controllers and other computers, and between Adept controllers and modem devices.

> **NOTE:** The Kermit file-transfer protocol is built into V^+, and can be enabled for a serial port with the utility CONFIG_C.

Requirements

You will need the following items to run XMODEM:

- Adept system controller
- Serial connection between the controller and the device being communicated with
- Correct configuration of communication parameters on the receiving and transmitting ends. (The utility CONFIG_C specifies the default serial configurationof the serial ports on the Adept controller. The FSET instruction can be used to change that configuration.)

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 as a program task other than #0.

Procedure

1. Load the program:[1]

   load \util\xmodem

2. Execute the program:

   execute a.xmodem

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

(Or type **execute 1 a.xmodem** to execute the program as task #1.)

3. The program asks you for the serial port to use for the file transfer (see the *V⁺ Operating System User's Guide* and the *Adept MV Controller User's Guide* for details.)
4. Next the program asks whether you want to send or receive a file.
5. After you indicate the direction of the transfer, the system asks you the name of the file to transfer or the name to give the file that is being received. The file name can include a complete directory specification, including drive designation.
6. After entering the file specification the program will make sure the remote system is ready to send/receive a file and then begin the file transfer.

   If the remote system does not respond after several tries the program will time out and halt.
7. After a successful file transfer, you are asked if you want to transfer another file.
8. The file transfer program can be deleted from the system memory by typing the commands:

   kill
   delete a.xmodem

   (You must type **kill 1** in place of **kill** if the program had been executed as task #1.)

Special Considerations

The XMODEM program defines may real-valued and string global variables with the prefix 'x.' that must not be modified by the user, or by other programs. (All these variables are deleted from memory when the program exits normally, or when a DELETE command is used to delete the program from memory, as described earlier.)

The disk file XMODEM.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 itself. The DISKCOPY program can be used to make a backup copy of the entire distribution diskette.

Index

Adept User's Manual Comment Form

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

NAME____ DATE________________________

COMPANY ____

ADDRESS____

PHONE____

MANUAL TITLE: ____

PART NUMBER and REV level:____

COMMENTS:

____

____

____

____

____

____

____

____