CROS and System Shell

UMI-R3-171

CROS and System Shell

Revision	History	Date
001	First release as UMI-R3-171. CROS 1.16 for C500C.	99-05
002	Updated for C500C	00-05

Contents

CHAPTER 1

Introduction

This Guide describes the CROS operating system and the system shell. CROS handles the controller resources and permits different processes to have access to these resources. CROS monitors point of control to the robot, so that, for safety reasons, only one process or application can command the robot at one time.

The system shell provides user access to the CROS operating system. From a computer running Robcomm3 or a suitable terminal emulator, you can access a system shell which permits communication directly with the CROS operating system. The system shell gives you command line access to the CROS operating system on the controller, so that you can work with system directories and files, system memory, and robot applications.

If you have a POLARA lab system, CROS for Windows NT is installed on your PC so that you can run RAPL 3 programs on the PC as well as the controller. Note however, that if CROS is installed on your computer and on a controller, the versions must be compatible in order to ensure proper operation of your applications.

CHAPTER 2

The Basics

This chapter describes how to use the CROS Operating System and the Shell (ash). There are nine sections:

- The Operating System: CROS
- The System Shell
- Running CROS
- Accessing the System Shell
- Using the Shell: Basics
- Working With Directories
- Working With Files
- Accessing System Tools and Applications
- Installing CROS.

The Operating System: CROS

The operating system of a computer manages the system resources and provides services to the various processes or programs that run on the system. Examples of operating systems for personal computers are DOS, Windows, and UNIX.

The CROS (CRS Robotics Robot Operating System) operating system on the controller performs the same functions for the controller. The CROS operating system:

- controls processes
- manages memory
- manages storage
- controls input and output.

CROS directs the interpretation of instructions for each process. CROS controls the resources of the controller and allows tasks to use the CPU, memory, and input/output devices. CROS schedules processes according to priorities, allows processes to wait for events, and handles the termination of processes.

CROS allocates memory for the processes being executed. It also manages the storage of files, the system's own files, RAPL-3 libraries needed by user programs, and the user's application programs and application variable files.

CROS also controls the transfer of data, including data coming in to the controller from the keyboard and going out to the screen during a terminal session.

As a user, you can monitor and direct the activity of the operating system by giving commands to the operating system using the system shell interface.

The System Shell

The system shell provides a command line interface between the user and the operating system. Consider the system shell as a tool which lets you access the CROS operating system. The shell interprets input from the keyboard and sends it to the controller, and takes output from the controller and interprets it for display to the screen. You access the system shell using a terminal emulator such as the terminal feature of Robcomm3.

The system shell is started up by the init (first) process when the system is started. You can create another system shell at any time from within a system shell or an application shell.

Note: If you exit from the only existing shell, the init process opens another shell. This ensures that there is always a shell open to interact with CROS.

Running CROS

When the controller is powered on, CROS is started up. In CROS, the init process starts a system shell. You do not have to start a system shell yourself to access CROS.

Starting CROS

CROS is started by the controller's start up sequence. You can power on the controller before opening a terminal session on your computer, but if you start a terminal session before powering on the controller, you can follow the controller boot sequence. The terminal window buffer captures all CROS start up messages which can be checked if there is a problem at start up.

1. Start a terminal session. (If you have Robcomm3, start Robcomm3, and open the terminal window. On the tool bar, click the terminal icon, or from the C500 drop-down menu, select the Terminal command.)
2. At the controller, turn on power at the controller's main power switch.

Shutting Down CROS

Use the shutdown now command or the front panel shutdown sequence to shut down the operating system in an orderly manner. If you bypass the shutdown command and just turn off the controller at its main power switch, the file system on the controller could be corrupted. All data on the controller would be lost.

To shut down CROS from a development computer

1. From a terminal window on the development computer, enter:

   ```
   $ shutdown now
   ```
2. Wait until the controller LCD screen displays the message:

   ```
   System Halted
   ```

If you do not have a development computer connected, you can shut down CROS manually from the front panel.

Note: The front panel shutdown sequence is only supported in CROS versions 2.6 and later.

To shut down CROS from the front panel

1. While holding down the Home button on the front panel, press and release the Pause/Continue button.
2. Release the Home button. The controller will begin shutting down.

   Note: You must complete steps 1 and 2 within a second or two. If nothing happens, simply try again a little faster or a little slower.
3. Wait until the controller LCD screen displays the message:

   ```
   System Halted
   ```

Accessing the System Shell

CROS and the system shell, running on the controller, are independent of any terminal session, running on the computer. The system shell is running even if a terminal session is not. If the controller is on and it started up normally, both CROS and the system shell are running. You can access the system shell by opening a terminal session at any time when CROS is running. If the system shell $ prompt is not displayed, press the enter key and it will appear.

A system shell is considered a process by CROS, and all processes including the system shells are listed in the process table. The process table can be viewed by issuing the ps (process status) command from a running system shell.

Starting With CROS Not Yet Running

If you open a terminal session before starting up CROS, you have access to the shell at the terminal window when CROS starts. The start up messages are displayed at the terminal window and finally the system shell $ prompt is displayed. If the system shell $ prompt is not displayed, press the enter key, and it will appear.

Accessing With CROS Already Running

You can start CROS and the shell (by starting the controller) without having a terminal session open. You can work with the shell at a terminal session and close the terminal session. You can also disconnect your PC from the controller and later re-connect.

CROS and the shell continue to run on the controller as long as the controller is powered.

To access the shell, open a terminal session. You may need to press Enter to reach the system shell prompt.

Starting Additional System Shells

You can have more than one system shell running at one time. The number of system shells and application shells is limited by available memory. You can start another system shell from an existing system shell or from an application shell.

To start another system shell from an existing system shell or an application shell, enter the command:

```
shell
```

To confirm the existence of the system shell, use the process status command ps to view the process table.

```
ps
```

The existing system shells are listed in the process table displayed. Refer to the *System Shell Commands* section for details regarding the shell and ps commands.

Exiting Out of a Shell

To exit out of the current system shell, enter the exit command:

```
exit
```

The exit command terminates the current system shell and returns you to your starting point. If you only have one system shell active, the exit command terminates your only active shell but the init process automatically starts up a new shell. As a result, you always have a shell (an interface) to the system running.

The application shell also has an exit command which exits you out of the application shell. If you exit an application shell (ash), you are returned to the system shell from which you opened the application shell.

Checking System Shell Version Number

To determine which version of the system shell you are running, enter the version command at the system shell $ prompt:

```
ver
```

The application shell, available at the ash prompt > , also has a version command, ver, to determine which version of the application shell you are running.

Using the Shell: Basics

From the system shell prompt you can issue commands to the controller operating system, CROS. A complete list of the available system shell commands and the required format and parameters is provided in the *System Shell Commands* section. Some on-line, command specific help is available. If you enter a system shell command with incorrect format or parameters, the system shell displays the proper command usage.

Entering Commands

Shell commands and parameters are entered as strings of characters separated by spaces:

```
command file_name
```

If a string is recognized as a valid shell command with properly specified parameters, then the corresponding CROS command is called and executed. If the command string is not recognized as a valid shell command, the shell assumes that the string is calling an executable program (application) and searches for that executable in the current directory and then in the bin directory. If an executable is not found in either directory, a message is displayed to the effect that the string is not recognized.

If the command is recognized but the parameters given are incorrect, the command usage is displayed.

Command History

The command line remembers the previous 10 commands. To review or repeat previous commands, you can move through the list of commands. To re-issue a previous command when it is displayed, press the Enter key.

There are also other command line editing features which can be used in conjunction with the standard keyboard editing functions to construct command line inputs.

The following table shows how to display the remembered commands:

Ctrl + p or Up arrow key	displays the previous command issued
Ctrl + n or Down arrow key	displays the next command issued, if you have already displayed the previous command(s) issued
Ctrl + f or Right arrow key	moves the cursor forward in the command string
Ctrl + b or Left arrow key	moves the cursor backward in the command string
Ctrl + d	deletes character

Working With Directories

When issuing system shell commands which specify files or directories, you must specify not only the file name but also the specific path to the directory in which the file is saved. This applies also to executable files (applications).

There are two simple exceptions to this rule:

1. If the file is in the active directory (the directory you are currently in), then the path can be omitted. CROS automatically looks in the active directory for the specified file or application.
2. If an executable file is in the bin directory, the path to the bin directory is optional. CROS looks in the bin directory for commands which are not recognized as system shell commands. If an executable file or application is placed in the bin directory, it can be accessed from any directory without the path.

Note: If a file with the same name exists in the active directory, CROS assumes that it is the specified file. In other words, CROS first looks in the active directory for the specified file before searching in the bin directory.

To display the current active directory, issue the pwd (**p**rint **w**orking **d**irectory) command:

```
$ pwd
```

The active directory is displayed. All directories and files accessible with system shell commands are stored in the controller memory. System shell commands do not provide access to the files stored in host computer memory.

Changing the Current Directory

Change the current active directory with the cd (**c**hange **d**irectory) command.

Tip: Use the pwd command to display the currently active directory.

To change to a Lower Level Directory

Name the sub-directory for each step. The names of the sub-directories are available with the ls (**li**s**t** directory) or dir (list **dir**ectory) command.

One Step at a Time

Name the sub-directory.

```
$ cd app
$ cd examples
$ cd lab
```

All Steps at Once

Name all sub-directories. Use a forward slash or a backslash.

```
$ cd app/examples/lab
$ cd app\examples\lab
```

To change to a Higher Level Directory

Use the cd command with **..** (dot, dot). The **..** (dot, dot) characters as parameters to the cd command move up one level to the parent directory of the current active directory.

One Step at a Time

Two dots.

```
$ cd ..
$ cd ..
```

All Steps at Once

Use a forward slash or a backslash with two dots for each level.

```
$ cd ../../..
$ cd ..\..\..
```

Tip: You can move to a higher level directory and then down a different directory path in a single cd command. For example:

```
$ cd ..\..\app\test
```

Moves up two directory levels and then down to the test directory in the app directory.

Listing the Contents of a Directory

You can list the names of sub-directories and files of a directory with the ls (**l**i**s**t directory) or dir (list **dir**ectory) command.

1. Move to the directory you want to list using the cd command.
2. Enter ls or dir.

```
$ ls
$ dir
```

The sub-directories and files of the current working directory are displayed.

With the -R parameter, you can list all directories and files in the file system from anywhere in the system. Refer to the ls command listing in the *System Shell Commands* chapter.

Creating a Directory

You can create a new directory or sub-directory with the md command. A new directory is created in the active directory.

1. Change to the directory you want as the parent directory with the cd command.
2. Use md and name the new directory.

```
$ md newname
$ md examples
```

The directory specified is created as a sub-directory of the working directory. You can check that your directory was created with the ls or dir command.

Note: You cannot make a directory and sub-directory at the same time, but you can make a sub-directory of an existing directory using the directory path. For example, to make a new directory 'lab' with the sub directory 'analysis', enter:

```
$ md  lab
$ md  lab\analysis
```

This is equivalent to:

```
$ md  lab
$ cd  lab
$ md  analysis
```

Working with Files

Viewing the Contents of a File

With the system shell commands, you can view the contents of CROS system files. Some files (binary files), such as executable files and variable files (.v3), may be unintelligible when displayed in the terminal window.

To view the contents of a file, use the more command. As an example, you can display the error codes for a subsystem using the more command. In the lib/errors CROS directory, there are 8 files with the name in the format sys001.err. Each file is a list of errors for a specific subsystem. For more information about system errors and error messages, refer to the *RAPL-3 Reference Guide* for a description of error descriptors.

To display the errors for the sub system 1 (Kernel), from the root directory, enter:

```
$ more lib/errors/sys001.err
```

The following list of errors is the response:

```
Kernel
000 no error
001 general error
002 not found
003 no such process
004 interrupted system call
005 i/o error
006 no device/operation not supported
007 too many arguments
008 not an executable
009 bad file descriptor
010 no child process
011 permission denied
012 out of memory
013 access denied
016 resource busy
017 file exists
018 cross device link
019 not supported by device
020 not a directory
021 is a directory
22 invalid argument
023 too many files on the system
024 too many files
025 not a tty
026 text busy
028 filesystem full
029 illegal operation on pipe or socket
034 result out of range
035 resource temporarily unavailable
037 timed out
039 not a socket
040 no server
041 no client
042 device is being reset
043 directory is not empty
045 operation not supported
```

Deleting a File

To delete files from CROS, use the system shell rm (**re**m**m**ove) command. Refer to the system shell commands.

Note: A deleted file cannot be recovered. To set aside a file until you are certain you will not need to recover it, create a directory with a name such as trash and move the file to that directory using the mv (**m**o**v**e) command. Later, delete the contents of trash when you are certain you do not need the file.

As an example, if you want to replace an existing variable file for your 'trim' application stored in app/trim directory, enter the following commands (assuming the root directory is the active directory):

```
$ md trash
$ mv /app/trim/trim.v3  /trash/trim_old.v3
```

When you want to delete the file from the root directory, enter:

```
$ rm /trash/trim_old.v3
```

Refer to the *System Shell Commands* chapter of this guide for details of the mv and rm commands.

Accessing System Tools and Applications

One of the key tasks you will want to perform from the system shell is access your applications and other system tools. You can access and run your applications directly from the system shell, or you can open either the teach pendant or the application shell and run your applications.

It is a more efficient use of your controller resources to run applications directly from the system shell rather than from the teach pendant or the application shell. When the teach pendant or application shell is running, it is using memory and related resources. However, you need to access the teach pendant and/or the application shell when you develop your applications.

Accessing the Application Shell (ash)

To access the application shell from the system shell, enter:

```
ash <application name>
```

If you omit the application name, the available application shells created are listed and you are prompted to specify which application you want to open. For details on the application shell, refer to the *Application Shell* section in this *Application Development Guide*.

From ash, you can return to the system shell by entering the exit command at the ash 'application name'> prompt. You can also open another system shell by entering:

```
$ shell
```

at the ash 'application name'> prompt.

Accessing the Teach Pendant

From the system shell you can also activate (or just give point of control to) the teach pendant.

Note: In boot-up sequence, point of control is given to the teach pendant.

Executables can be executed or run from the system shell. A RAPL-3 program, when successfully compiled, creates an executable (object) file. The object file must be on the controller before it can be run. Refer to the other sections of this *Application Development Guide* for details about creating applications.

Compiled RAPL-3 object files have no extensions in their file names and are executable files. The RAPL-3 source files (.r3 extensions) and variable files (.v3 extensions) are not executable.

When an executable is run, a process is created. You can check the status of the process using the ps (process status) command, which displays the process table listing all processes currently running on the system. Refer to the *System Shell Commands* section in this chapter for details on the ps command.

RAPL-3 programs can be run from the teach pendant, the application shell, or the system shell.

In fact, running a robot application from the system shell is more efficient than running it from the application shell. Without the application shell loaded into memory, the system can use that memory during execution. During application development, you must have the application shell to teach locations and move the arm.

Running From the File's Directory

One approach to running a file is to first change the current directory to the directory that contains the file to be run.

1. Move to the directory containing the program using the cd command.

```
$ ls
  app/  bin/  conf/  dev/  lib/  log/  sbin/  tmp/
  $ cd app
  $ ls
  load/  prepare/  sample/  test/
  $ cd test
  $ ls
  test     test.v3
```

2. Enter the name of the file:

```
$ test
```

To check on the status of the file while it is running, use the **ps** command.

Running From a Parent Directory

You do not have to be in the file's directory to run the file. You can run the file from a parent directory or any directory which is at a higher level up the directory tree from the program.

1. Remain in the higher level directory.

```
$ ls
  app/  bin/  conf/  dev/  lib/  log/  sbin/  tmp/
```

2. Enter the path down to the file:

```
$ app/test/test
```

Running From Another Directory

The current working directory is the default where the system searches for the executable file. You can specify a path to another directory.

You do not have to be in the file's directory or a parent directory to run the file. You can run the file from any other directory. However, you must specify a path to the executable file. You must provide the path to a common shared parent directory (which could be the root directory) and then the path down to the file's directory.

1. After changing to another directory, remain in that directory.

```
$ ls
app/  bin/  conf/  dev/  lib/  log/  sbin/  tmp/
cd bin
```

2. Enter the path up, using **..** (dot, dot) for each step, to the directory common to both the current directory and the file's directory, and the path back down to the file.

```
$ ../app/test/test
```

Running an Executable in the Background

When you run an application from the system shell, the program executes and, when the program has completed, command returns automatically to the system shell. With CROS, you can start a file and put it in the background, and while it is still running, return to the system shell prompt in order to enter other commands.

1. To run an application test in the background, enter the name of the program, a space, and an ampersand.

```
$ test &
```

The program runs and the system shell $ prompt returns immediately. To check on the status of the program while it is running, use the ps command.

Note: Never run an interactive program in the background. An interactive program requires user input.

Note: If the executable process running in the background requires and receives access to the robot, no other process running can control the robot.

Installing CROS on the PC

If you are compiling applications on your development computer, or you want to upgrade the firmware on your controller, you must install CROS on the development computer.

Before you Begin

If you have previously installed CROS on the computer, remove the previous version with the Windows Uninstall feature available from the Control panel.

To install CROS on the computer.

1. Insert your Robot System Documentation and Software CD in the CD-ROM drive on the development computer.
2. Wait until the startup screen appears.
3. On the startup screen, click **Install CROS for the C500C** and follow the instructions provided on screen.

Installing CROS on the Controller

Although CROS is pre-installed on your controller at the factory, you can upgrade or re-install CROS on the controller if necessary.

To install CROS on the controller

1. Install CROS on the development computer. The Firmware Download Utility is automatically installed as part of the CROS installation.
2. Connect the development computer to your controller.
3. Using Robcomm's file transfer, copy all files in the /app and /conf directories to a safe location on the development computer.

   Note: As part of the firmware download procedure, the file system on the controller is completely erased and rebuilt.
4. Follow the instructions in the Firmware Download Utility Guide (provided on the Documentation and Software CD) to install CROS on the controller.

Testing CROS on the Controller

You can test to ensure that CROS is running on the controller by running a RAPL-3 program. Run the application at low speed and ensure you have immediate access to an e-stop button in the event that the application locations are no longer accurate.

CHAPTER 3

System Shell Commands

This chapter describes the commands that you can use through the system shell. There are three sections:

- Categories of commands
- Similarities to UNIX and DOS
- Detailed descriptions of all commands listed alphabetically

Categories of Commands

Details of the commands are given in the alphabetical listing.

Start and Exit	
shell	start new system shell
exit	exit current system shell
ver	display version of system shell
crosver	display version of operating system (CROS)
msleep	put system shell to sleep
shutdown	shut down operating system (CROS)
ash	start new application shell
Terminal	
echo	echo a message to the console
pause	wait for the user to type <enter>
Shells	
do	execute a shell script
auto	update the startup shell file
/diag/setup	setup (configure) the robot
Maintenance	
siocfg	reconfigure serial port
Memory	
mem	display space in memory
df	display space on file system
sync	flush filesystem buffers; defragment memory on C500
Program Editing, Compiling, Teaching	
edit	start the terminal editor to edit a file
pedit	start the pendant editor to edit a file
r3c compile	invoke the RAPL-3 compiler
rc1	provide the main pass of the RAPL-3 compiler
touch	change the modification time of a file to the current time
v3make	make or update a v3 file for a program
rpp	handles .define macros and .ifdef conditionals
pendant	run the teach pendant

Directories, Files, Devices	
pwd	display current working directory
cd	change current working directory
ls dir	list directory contents
mkdir md	make new directory
rmdir	remove/delete directory
ln	make link to file
mv ren	move/rename file
rm del	remove/delete or unlink file
cp copy	copy file
more	display contents of file
type	display contents of file
cksum	calculate checksum of file
chmod	change protection mode
mkdev	make device
mkfifo	make fifo
mksock	make socket
mount	mount a file system on a directory
unmount	unmount a file system from a directory
Processes	
ps	display status of processes
kill	terminate a process
Time	
date	display or set date and time
Front Panel	
panel	provide a menu for selecting and executing shell commands
lcd	display characters at front panel lcd

Robot Related Commands	
axst†	display low-level status information for the robot axes
calrdy†	move the robot to the calibration ready position
gtype†	set the robot gripper type
home†	home the robot
joint†	move a robot joint
limp†	limp robot axis
motor†	move a robot motor
nolimp†	unlimp robot axes
ready†	move the robot to the ready position
pendant†	run the teach pendant
speed†	set the robot speed
wact† w0†	display the robot actual position
Robot Configuration and Maintenance Commands	
odometer	display the robot armpower on-time odometer
/diag/cal	calibrate robot axes
/diag/calgrip	calibrate the servo gripper
/diag/encres	reset the joint position encoders (F series only)
/diag/f3diag	perform F3 specific diagnostics
/diag/f3pack	move an F3 robot into its packing position
/diag/setup	master configuration program for setting up the robot
/diag/xzero	zero a particular motor position register
/diag/zero	zero all motor position registers
Help	
help	get descriptions of system commands

† These commands are identical to the like-named ash commands, except that parameters are separated by spaces, not commas. See the *Application Shell (ash)* section of this *Application Development Guide* for details.

Similarities to UNIX and DOS

CROS has many commands and features similar to UNIX and DOS.

If you are already familiar with one of them, you can use a CROS command that is similar. For example, to list a directory in CROS, you can use ls (a UNIX-like command for list directory) or dir (a DOS-like command for list directory).

Command Names: CROS, UNIX, and DOS

The following table lists all CROS commands with similar commands in UNIX and DOS. CROS, UNIX, and DOS commands are not always identical. Check the alphabetical listings for full descriptions.

CROS	UNIX	DOS
ash	—	—
cd	cd	cd chdir
chmod	chmod	attrib
cksum	sum	—
copy	cp	copy
cp	cp	copy
crosver	—	ver
date	date	date + time
del	rm	del erase
dir	ls	dir tree
df	df	chkdsk mem dir
edit	ed	edit
exit	exit	exit
help	man	help
kill	kill	—
ln	ln	—
ls	ls	dir tree
md	mkdir	mkdir md
mem	vmstat	mem
mkdev	mknod	—
mkdir	mkdir	mkdir md
mkfifo	mknod	—
mksock	mknod	—

CROS	UNIX	DOS
more	more	more
mount	mount	—
msleep	sleep	—
mv	mv	rename ren move
ps	ps	—
pwd	pwd	cd
ren	mv	move rename ren
rm	rm	del erase
rmdir	rmdir	rmdir rd
shell	sh csh ksh	command
shutdown	shutdown -i0	—
siocfg	stty	mode
sync	sync	defrag
type	cat	type
unmount	unmount	—
ver	—	—
&	&	—

Path Separators

CROS allows both the / (forward slash) familiar to UNIX users and the \ (backslash) familiar to DOS users as separators between directories and files in a path. Since they are equally valid, they could be mixed in a path.

CROS (UNIX-style)	CROS (DOS-style)
/app/test/test.r	\app\test\test.r

Wildcards

The CROS system shell can handle simple wildcards. The special characters recognized are:

*****	matches zero or more of any character
?	matches exactly one of any character

For example, the string “bob*” matches “bob” “bobby” and “bobbobbobbing”. The string “b?b” matches “bib”, “bob”, “byb” but not “bxxb” or “bobs.”

Wildcard expansion works the same as in UNIX. When a command line is typed, the shell expands wildcards before executing the command. When the shell detects a ‘*’ or a ‘?’ wildcard character in the command line, it searches for files that match the specified pattern and replaces the pattern with the actual file names.

As an example, suppose that the following files are in our current working directory:

bob.v3	bib.v3	babe.v3
bib.r	temp.txt	test/
test/temp.temp	test/t.v3	test/q.v3

Then these command lines are expanded by the shell as follows:

ls b*3	ls babe.v3 bib.v3 bob.v3
ls t*/*	ls test/t.v3 test/temp.temp test/q.v3
ls b?b*	ls babe.v3 bib.r bib.v3 bob.v3

A good way to better understand wildcard expansion is to play with the echo command; for example, “echo *” will echo the names of all files in the current working directory.

General Command Format

Most CROS commands have the following general command line format:

command [-options…] param1 param2 param3 …

Where:

command	the command name
-options…	the options list; introduced by a – (dash) character. Options may be appended together (like “-aRl”) or listed individually (like “-a –R –l”).
param1…	the other parameters to the command, separated by spaces.

Detailed Descriptions

These are detailed descriptions of all system shell commands listed alphabetically.

Where a command has two names, there is an entry for each. For example, there is an entry for ls and an entry for dir. Details are only at one entry. A cross-reference directs you from the other entry to the one with details.

file_name

Description

Runs the specified file. Running a robot application program from the system shell is more efficient than running it from the application shell.

Format

The following short-forms are used in the next table:

xpath	the path to the executable program file
xname	the executable program file name
vpath	the path to the variable file
vname	the variable file name

File names can be entered according to any of the following formats:

xname	program file name only (uses variable file of same name)
xname**:*****vname***	program file name with variable file name
xname**:*****vpath/vname***	program file name with variable file path and variable file name
xpath/xname	program file path and program file name (uses variable file of same name)
xpath/xname**:*****vname***	program file path and program file name with variable file name
xpath/xname**:*****vpath/vname***	program file path and program file name with variable file path and variable file name

The .v3 extension is optional. The current working directory is the default where the system begins searching for the file unless you specify a path to another directory.

Examples

test1	test1 with test1.v3
test1:alpha	test1 with alpha.v3
test1:samples/beta	test1 with (from samples directory) beta.v3
test/prep	(from test directory) prep with prep.v3
test/prep:alpha	(from test directory) prep with alpha.v3
test/prep:samples/beta	(from test directory) prep with (from samples directory) beta.v3

ash

application **sh**ell

Description

Starts up a new application shell.

The application shell interprets application commands.

Syntax

ash [*application_name* [*variable_file_name*]]

Parameters

Takes an optional application name parameter. If the application is specified, then the variable file name can be optionally specified.

application_name	the application to open when ash loads
variable_file_name	the variable file to load into the database with or without the .v3 extension

If no application name is specified, the shell prompts for the application name. If no variable file name is specified, the shell loads the variable file name with the same name as the application.

Examples

ash test alpha	use application “test” with v3 file “alpha.v3”
ash test test	use application “test” with v3 file “test.v3”
ash test	use application “test” with v3 file “test.v3”
ash	prompt the user for which application to use

See Also

exit (in application shell) exits from application shell

auto

set **auto**start commands

Description

The **auto** command is used to update the **/conf/startup.sh** file. This file is a list of commands that gets executed by the shell after the rc (initialization) file, when the system first starts up. The **auto** command can be used to set the system to automatically launch a program at start up for turnkey applications.

Syntax

auto *[-options] [command]*
auto *[-options]*

Parameters

The command takes two optional sets of parameters:

options	An option. See the options list below.
command	The system shell command to add to the conf/startup.sh file. If *command* is omitted, then the **auto** command calls up the editor (**edit**).

These are the valid options for the **auto** command:

Option	Description	
-a	append	append *command* to the end of the startup.sh file, or simply edit the existing file if *command* is not specified. (default)
-d	delete	simply delete the **startup.sh** file. Used to cancel a previously defined set of startup commands.
-h **-?**	help	display a usage message
-x	overwrite	discard the existing **startup.sh** file and create a new one.

Examples

```
auto ls       appends the "ls" command to the startup.sh file
auto          edit the existing startup.sh file
auto –d       delete the startup.sh file
auto –x ls    create a new startup.sh file with "ls" in it
```

axst

axis **st**atus

Description

Continually displays low-level axis status information. This information is displayed as a set of 8 hexadecimal numbers, one per axis. The data is of diagnostic use only, and is not useful to general users.

Syntax

```
axst
```

Parameters

This command takes no parameters.

Example

```
axst
```

cd

change **d**irectory

Description

Changes the current working directory to the directory specified.

Syntax

cd ***path***

Parameters

Takes one required parameter, specifying the absolute or relative path to the directory to change to.

path any absolute or relative directory path

Paths can be absolute or relative; absolute paths start with either "/" or "\" and specify the location of a directory from the root of the directory tree.

Relative paths start at the current working directory. The following special directory names are often useful in relative paths:

- `.` refers to the current directory
- `..` refers to the parent of the current directory

Examples

cd test	**cd** into the **test** directory, (which must be a child of the current working directory.)
cd /dev	**cd** into the **dev** directory, (which must be a child of the root of the directory tree.)
**cd **	**cd** into the **root** directory
cd app\test	**cd** from the current directory into the **app** directory, and from there into the **test** directory.
cd ..	**cd** one level up in the directory tree into the parent of the current working directory.
cd ../..	**cd** two levels up in the directory tree.

See Also

pwd	print working directory
ls (or dir)	list contents of directory

chmod

change **mod**es

Description

Changes the access modes of an object (file, device, fifo, or socket).

Syntax

chmod *modes object_names…*

Parameters

This command accepts two arguments: *modes*, which specifies how the modes of the listed objects are to be changed, and *object_names*... which is a list of object names.
The *modes* argument consists of a list of the following key letters, possibly with + or – characters before them. The mode key letters are:

r	permit **r**eading from the object
w	permit **w**riting to the object
x	permit e**x**ecuting the object
s	mark this as a **s**ystem object

The '+' sign indicates that the following mode bits are to be set for (added to) the object; the '-' indicates that the mode bits are to be cleared (removed from) the object. If no '+' or '-' appears, then the access modes are simply set as indicated. For example:

+w	add the **w** flag to the object
-xw	remove the **x** and **w** flags from the object

rw	set only the **r** and **w** flags; clear all of the others
-s+rw-x	set the **r** and **w** flags; clear the **s** and **x** flags

Examples

```
chmod rwx this_file
chmod +w that_fifo
```

See Also ls -l lists access modes of entities

cksum

chec**ksum**

Description Calculates and prints the checksum of a file. The checksum is a number determined by a mathematical calculation on the bits of the file, and can be used to compare files to ensure that they are identical. Prints the hexadecimal checksum, the decimal integer number of bytes of the file, and the filename. At present, the checksum is a simple 32-bit sum of each of the 8-bit bytes in the file.

Syntax `cksum` *`file_name`* [*`file_name_2`*] [*`file_name_3`*]

Parameters Takes one or more parameters:

`file_name`	the name of the file to check

Example

```
cksum sieve.r
```

Result

```
0x00006b8a 416 sieve.r
```

Example

```
cksum teachflo.r teachint.r teachloc.r
```

Result

```
0x000085db 788 teachflo.r
0x00009e23 912 teachint.r
0x0000704c 628 teachloc.r
```

compile

invoke the RAPL-3 **Compile**r

Allows the user to recompile RAPL-3 programs.

See **r3c**

copy

copy objects

Copies the contents of one object to the contents of a second object.

See **cp**

cp
copy

co**p**y objects
copy objects

Description — Copies the contents of one object to the contents of a second object. If the second object does not exist, then it is created. If the second object already exists, over-writes the existing contents of the second object. This is similar to the operation of the **copy** command in DOS and the **cp** command in Unix. Typically, copy is used to copy files; it can also be used to copy a file to a device or to copy a file to a fifo.

Syntax

```
cp source_object_name destination_object_name

cp source_object_name [source_object_name_2...]
                                    destination_directory_name
```

Parameters — To copy one object to another, cp takes two required parameters:

source_object_name	the object to copy from (can include a path)
destination_object_name	the object to copy to (can include a path)

To copy one or more object to another directory, **cp** takes at least two required parameters:

source_object_name	the object to copy from
source_object_name_2	one or more additional objects to copy from
destination_directory_name	the directory to copy to (can include a path)

Example

```
cp  lab_test.r lab.r
copy alpha.r alpha.v3 \app\trash
```

See Also

ln	makes a link to an object
rm (or del)	removes (deletes) an object or link
mv (or move)	moves or renames an object

crosver

CROS version

Description — Displays the version of the operating system (CROS) being used.

Syntax — **crosver**

Parameters — No parameters.

Example

```
crosver
```

Result

```
System type:  'CROS on a C500'
Version:      1.11.424
Click size:   64
msec/tick:    10
```

See Also

ver	(in the system shell)	displays version of system shell
ver	(in the application shell)	displays version of application shell

date

date

Description — Displays or sets the current date and time.

Syntax
date
date [*yyyy mo dd hh mi ss*]

Parameters — To display the date, takes no parameters.
To set the date, takes six required space-separated parameters:

yyyy	year	1970, 1971, ... 2037
mo	month	1, 2, ... 12
dd	day	1, 2, ... 31
hh	hour	0, 1, ... 23
mi	minutes	0, 1, ... 59
ss	seconds	0, 1, ... 59

All parameters are integer numbers. A first leading zero is optional (00, 01, 02, ...). The date command performs some error checking for correct combination of day with month or year.

Example
date
Result
Wed Jan l 00:00:00 1997

Example
date 1997 07 01 15 30 00
Result
the date is set to **Tue Jul 1 15:30:02 1997**

del

delete

Deletes (removes) an object (file, device, fifo, or socket).

See — **rm**

df

disk **f**ree space

Description — Displays the number of free bytes on the file system.

Single File System

For systems with a single file system like CROS on a C500, no parameter is necessary.

Syntax
df

Parameter — This command takes no parameters.

Example
df

Result
```
. (mfs): 101440 bytes free (block size = 64)
```

Multiple File Systems

For systems with multiple file systems, CROS on Windows NT, any directory in the desired file system must be specified.

Syntax **df *name***

Parameter Takes an optional parameter:

name For the host file system or a mounted file system, any directory name or file name in the desired file system

If no parameter is given, the default is used which is . (dot) for the current directory.

Example
```
df  /app
```
Result
```
. /app(hostfs): 72276992 bytes free (block size = 512)
```

See Also mount mounts a file system on a directory

/diag/cal

re**cal**ibrate specified axes of the robot.

Description This command is used to recalibrate and home selected axes of the robot. **WARNING: the cal command will overwrite your existing calibration files!** The axes in question should be properly zeroed before performing this procedure. Under normal operation you will never have to recalibrate the robot.

Syntax **/diag/cal** *[axes…]*

Parameters Optionally takes a list of axes to recalibrate and home; if the list is omitted, then all robot axes are recalibrated.

Example
```
/diag/cal          recalibrate and home all axes
/diag/cal 5        recalibrate and home axis 5
```

See Also home (in the application shell) home specified robot axes
home (in the system shell) home specified robot axes

/diag/calgrip

re**cal**ibrate the servo**grip**per

Description

This command is used to recalibrate the servo gripper. **WARNING: the calgrip command will overwrite your existing servo gripper calibration file!** Under normal conditions, you will never have to recalibrate the servogripper; this command is generally used to set up the gripper when it is first installed.

The calgrip program opens the servo gripper and prompts for you to enter the distance that the jaws are open (typical is 2.0 inches); then the program closes the gripper and asks what distance the jaws are open (typically 0.0 inches.) The program then calibrates the gripper and exits.

Syntax

`/diag/calgrip`

Parameters

This command takes no parameters.

Example

`/diag/calgrip`

See Also

/diag/cal	(in the system shell)	calibrates and homes robot axes
/diag/cal	(in the application shell)	calibrates and homes robot axes
gtype	(in the application shell)	sets the type of gripper

/diag/configur

configure robot

In earlier versions of CROS, the name for /diag/setup.

See

/diag/setup

/diag/encres

F3 **enc**oder **res**et

Description

Resets the absolute encoders on an F3 robot.

WARNING: this command is generally used in preparation to calibrating the robot and is not used in normal operations. Please refer to the F3 operation manual for details on the use of this command.

Syntax

`/diag/encres`

Parameters

This command takes no parameters.

Example

`/diag/encres`

Result

```
The F3 encoders are reset.
```

/diag/f3diag

F3 diagnostics

Description — This program allows the user to perform several F3 specific diagnostics.

Syntax — **f3diag**

Parameters — The command takes no parameters.

Example — `f3diag`

Result —

```
F3diag v. 2.19
Commands available:
(M)odule Detection (N)etwork Scan
(C)alibration Info (J)oint Limits
(H)elp             (Q)uit
Command:M
Waist amp detected.
Wrist amp detected.
EOA-IO or SG-IO board detected.
A total of 3 modules detected.
Command:Q
```

/diag/f3pack

F3 packing position.

Description — This command is used prior to shipping an F3 robot, to place the robot in its packing (curled-up) position.

Syntax — **f3pack**

Parameters — This command takes no parameters.

Example — **f3pack**

/diag/setup

robot **setup**.

Description — This command asks the user a series of questions about the system and robot and writes a new /conf/robot.cfg file based on the answers.

This command was called configur in early versions of CROS.

Syntax — **/diag/setup**

Parameters — This command takes no parameters.

Example — Note that what the computer prints is in regular type; sample user input is underlined.)

```
$configur
```

Result

```
Robot Configuration

Are your coordinates in English (0) or Metric (1) units?
0
Your robot has 6 axes. How many additional axes are
connected to your controller?
1
Is the robot mounted on a track?(1=yes, 0=no)
1
Enter the positive travel limit of the track in inches.
35.5
Enter the negative travel limit of the track in inches.
-0.5
Do you have a servo gripper installed ?(1=yes, 0=no)
0

Robot is configured.
This may cause a PIC TIMEOUT ERROR and a LOSS OF ARM
POWER
Please restart controller to allow new settings to take
effect.

$
```

See Also

/diag/cal	(in the system shell)	calibrates and homes robot axes
/diag/cal	(in the application shell)	calibrates and homes robot axes
gtype	(in the application shell)	sets the type of gripper

/diag/xzero

a**x**is **zero** position registers

Description

Zeros the position registers of a particular robot axis.

WARNING: this command is generally used in preparation to calibrating the robot and is not used in normal operations.

Syntax

/diag/xzero *axis_number*

Parameters

There is one required parameter:

axis_number	Which robot axis to zero the position registers of.

Example

/diag/xzero 7

Result

```
The position registers of axis 7 are set to zero.
```

See Also

/diag/zero	zeros the position registers of all robot axes
/diag/cal	calibrate and home robot axes

/diag/zero

zero position registers

Description — Zeros the position registers of all robot axes.

WARNING: this command is generally used in preparation to calibrating the robot and is not used in normal operations.

Syntax — **/diag/zero**

Parameters — This command takes no parameters.

Example — **/diag/zero**

Result — `The position registers of all robot axes are set to zero.`

See Also

/diag/xzero	zeros the position registers of all robot axes
/diag/cal	calibrate and home robot axes

dir

list **dir**ectory

Lists the contents of a directory

See — **ls**

do

do shell script

Description — This command tells the shell to execute a text file containing a list of system shell commands. The file should consist of shell commands, one command per line, possibly interspersed with '#' delimited comments.

Syntax — **do** *script_file_name*

Parameters — There is one parameter:

script_file_name	The name of the text file containing the shell commands. The system will search for this file in the current working directory and then in /bin.

Example — Given a file test.sh that contains this:

```
# This is a comment
echo Hello
pause Hit any key to continue
echo Done.
# This is another comment
```

The "**do test.sh**" will print out the message "Hello" and "Hit any key to continue", then wait for the user to hit a key, and print out the message "Done."

See Also **shell** start up the system shell

echo

echo a message to the console

Description Echoes a message to the console. This is useful inside scripts.

Syntax `echo [-n] [text…]`

Parameters There are two optional parameters:

-n	Do not print, go to a new line after printing the message; leave the cursor right at the end of the message.
text…	The text of the message to display.

Example `echo This is a test`

Results `Displays the message "This is a test", and moves the cursor to the next line.`

See Also **do** do a shell script
pause wait for user input

edit

edit a text file

Description This command invokes the command line text editor. **edit** can be used to create and modify configuration and source files.

Syntax **edit** *[-options] filename*

Parameters The command takes one optional sets of parameters and one required parameter:

options	an option (see the list below)
filename	the name of the file to edit

These are the valid options for the **edit** command:

Option	Description	
-n	no menu	do not display the editor help menu on startup.
-h **-?**	help	display a usage message
-l	use default command log	use the default command history log file (/log/command.log) for inserting. This allows a program to dump out its command history to a file and permits the editor to select lines from that history list to insert.
-L *filename*	use the specified	same as **–l**, but loads a specific command

	command log	history log file.

Examples

```
edit /conf/rc          Edit the startup script file.
edit –n myprog.r3      Create and edit myprog.r3.
                        Don't display the editor help menu.
```

See Also

pedit	starts the pendant editor

exit

exit the shell

Description

Terminates the current shell. Takes one optional parameter, which is the exit code to return to the parent process.

If you have only one system shell running, the exit command exits from that shell, but the init process starts up a new shell to ensure that you always have communication with the system.

Syntax

exit [*code*]

Parameters

There is one optional parameter:

code	The exit code to return to the parent process.

See Also

shell	starts a new system shell
exit (in the application shell)	exits from the application shell

help

help

Description

Displays help on system shell commands. Displays the command name, its parameters, and a brief description.

You must be in the system shell to get help on commands that are accessed only from the system shell. In the application shell, the help command gives help on commands that are accessed from the application shell.

Syntax

help [*command_name*]

Parameters

Takes one optional parameter:

command_name	the command for which you want help.

No parameter gives a list of all system shell commands.

Examples

```
help ls
help shutdown
help
```

See Also

help (in the application shell)	displays help on ash accessible commands.

kill

kill

Description

Sends a specified signal to a set of processes. Typically this can be used to terminate (kill) the processes.
By default, an INT (interrupt) signal (7) is sent, which is the equivalent of pressing Ctrl E. The kill signal (1) is the only signal that cannot be masked or caught by the target process; this signal can be sent in extreme cases to terminate an errant process.

Syntax

kill *[-signal]* ***process_number…***

Parameters

signal the signal name or number
process_number... a list of process id numbers

To obtain the process number and the process state, use the ps command.

Signals				
Number	**Name**	**RAPL-3 Symbol**	**Description**	**Default Action**
1	KILL	SIGKILL	kill; cannot be caught or masked	terminate
2	SEGV	SIGSEGV	segmentation violation	terminate
3	SIGILL	SIGILL	illegal instruction	terminate
4	FPE	SIGFPE	floating point exception	terminate
5	SYS	SIGSYS	bad argument to system call	terminate
6	ABRT	SIGABRT	abort	terminate
7	INT	SIGINT	interrupt	terminate
8	ALRM	SIGALRM	alarm clock	terminate
9	HUP	SIGHUP	hang up	terminate
10	PIPE	SIGPIPE	write to pipe, but no process to read it	terminate
11	SOCK	SIGSOCK	write to socket, but no process to read it	terminate
12	RPWR	SIGRPWR	robot power failed	terminate
13	13	SIG13	user defined	terminate
14	14	SIG14	user defined	terminate
15	15	SIG15	user defined	terminate
16	16	SIG16	user defined	terminate
17	CHLD	SIG17	child process died	ignore
18	18	SIG18	user defined	ignore
19	19	SIG19	user defined	ignore
20	20	SIG20	user defined	ignore
21	21	SIG21	user defined	ignore
22	22	SIG22	user defined	ignore
23	23	SIG23	reserved for CRS use	ignore
24	24	SIG24	reserved for CRS use	ignore

The effect of a signal on a process depends on what state the process is in:

Signal Number	State				
	RUN / SLEEP / WAIT	WIO (Wait for I/O)	IWIO (Interruptible Wait for I/O)	WSEM (Wait for SEMaphore)	WSOCK (Wait for SOCKet)
SIGKILL (1) to SIGALRM (8)	interrupt	no	interrupt	interrupt	interrupt
SIGHUP (9) SIGPIPE (10)	interrupt	no	no	no	no
SIGSOCK (11)	interrupt	no	no	no	interrupt
SIGRPWR (12) to SIG22 (22)	interrupt	no	no	no	no
SIG23 (23) SIG24 (34)	interrupt	no	no	no	interrupt

The "interrupt" entries denote that the operation can be interrupted by the given signal. For example, an I/O read (IWIO) can be interrupted by a SIGALRM.
The behaviour of signals SIG23 and SIG24 may change in the future since they are reserved for CRS use.

Examples

```
kill  -9  64 65      send a SIGHUP to processes 64 and 65
kill –HUP 22         send a SIGHUP to process 22
```

See Also

ps displays process numbers and process states

lcd

display text on or clear the front panel **LCD** display.

Description

The **lcd** command allows messages to be displayed on the C500C front panel LCD (liquid crystal display). It also allows clearing the display.

Syntax

lcd *[first_line [second_line]]*

Parameters

There are two optional parameters to this command:

first_line	text to appear on the first (top) line of the display.
second_line	text to appear on the second (bottom) line of the display.

If no parameters are given, **lcd** clears the lcd display.

Examples

```
lcd
```
clear the lcd display
```
lcd "Hello, World!"
```
display the string "Hello, World!" on the first line of the display, leaving the second line unchanged.
```
lcd Hello, World!
```
display "Hello," on the first line and

"World!" on the second line of the display.

ln

lin**k**

Description

Makes a hard link to an object (file, device, fifo, or socket). Can be used to create another name for an object. Can be used to rename an object, if the **ln** command is followed by an **rm** of the original name.

If all links to an object (like a file) are removed (with **rm**, for example), then the object ceases to exist.

Hard links are presently supported only within a CROS-500 mfs filesystem or a CROSnt CFS filesystem; in particular, the CROSnt hostfs (which allows access the host filesystem) does not support hard links, as Windows NT™ itself does not support hard links.

Syntax

ln *source_object target_object_name*

Parameters

Takes two required parameters:

source_object	the object identified by an existing name
target_object_name	the new name

Examples

```
ln  sample_alpha.v3  sample_beta.v3
```

See Also

rm (or del)	breaks a link
cp (or copy)	copies an object
mv (or move)	moves or renames an object

ls
dir

list directory
list **dir**ectory

Description

Lists the directory, sub-directories, entities, and/or information about them depending on optional parameters.

Syntax

`ls [-options] [directory_name]`

Parameters

Takes two optional parameters:

options	the options list (see below)
directory_name	the name of a specified directory

If no option is given, lists the contents by name.
If no directory name is given, lists the current directory.

Option	Description	
-a	all	lists all including any normally hidden files which begin with a . (dot), the current directory, and the parent directory
-I	inode number	lists the inode (**i**nternal **node**; the index of where object information is actually stored) for each sub-directory or object
-l (the letter l)	long	lists the current or specified directory with details as described in the output sample below
–R	Recursive	lists recursively (lists the contents of all sub-directories, sub-sub-directories, etc. starting at the specified directory)
–1 (the number 1)	1 (single)	lists the current or specified directory, one sub-directory or object per line

Note that the options can be combined:

Examples

```
ls  -l
ls  -R
ls  -aRl /temp
ls  -R  -i /dev
```

Long Option

The long option is described below with example command, resulting output, and description of output categories.

Example

```
ls  -l
```

Result

The output displays without any column names.

file mode	links	size or device	date	name
d---rwx	2	160	Jul 1 12:00	./
d---rwx	12	416	Jul 1 12:00	../
-f-Sr-x	1	9320	Jul 1 12:00	fastacid
-f-Sr-x	1	35468	Jul 1 12:00	robotsrv
-f--r-x	1	47716	Jul 1 12:00	stpv3

Description

The long option gives five columns of information: file mode, links, size or device, date, and name.

file mode

File mode contains seven sub-columns of information: type of object, flash location (primary or secondary), system ownership, and mode of protection (read, write, or execute).

File Mode						
Type of Object	Flash Location		System owner-ship	Mode of Protection		
	Primary	Secondary		Read	Write	Execute
d directory	**f** in primary flash	**F** in secondary flash	**S** a system object	**r** can read	**w** can write	**x** can execute
– file						
v device	**–** not in primary flash	**–** not in secondary flash	**–** not a system object	**–** cannot read	**–** cannot write	**–** cannot execute
p pipe						
s socket						

Type of object is always indicated (each object is one of the listed types). Other indicators use a letter for yes/on and a – (dash) for no/off.

links

Indicator of the number of directory entries that refer to this object.

size or drivers

Size of file or identifiers for driver.

Type of Object	Information Given
file or directory	size in bytes
device	major and minor driver codes for the device driver

date

Date and time of last modification.

name

Name of object or directory.

See Also

cd	changes current working directory
chmod	changes access mode of an object

md

make **d**irectory

Makes a new directory.

See **mkdir**

mem

memory available

Description — Displays a summary of free space in memory and the percentage of fragmentation of that space. Zero percent indicates that free memory is completely contiguous.

Syntax — **mem**

Parameters — This command takes no parameters.

Example — mem

Result —
```
152512 bytes free
5% fragmentation
```

See Also — sync flushes file buffers and defragments memory

mkdev

ma**k**e **dev**ice

Description — Makes a new device. A device is an object in the file system (usually in the device directory) which is necessary to communicate with peripherals. These peripherals are also called devices.

Most devices correspond to external hardware components like those connected through the GPIO or the front communication port. In addition, CROSnt includes some special purpose device drivers supporting communication between CROSnt programs and native WindowsNT™ programs.

Every device has a device driver that instructs CROS how to perform various communication functions with the external device. The device entry in the file system, created by mkdev, tells CROS which device driver to use when communicating with the external device through the internal device driver software.

Syntax — **mkdev *device_name major minor***

Parameters — Takes three required parameters:

device_name	the name of the device
major	the major identifier of the device driver
minor	the minor identifier of the device driver

Examples — `mkdev \dev\terminal 1 1`

Same As RAPL-3 — mknod() makes device, fifo, or socket

See Also — chmod changes protection mode

mkdir
md

ma**k**e **dir**ectory
make **d**irectory

Description — Makes new directories.

Syntax — **mkdir *directory_name…***

Parameters — Where:

directory_name the name of the directory

Examples —
```
mkdir  lab
mkdir  lab\analysis
```

See Also — rmdir removes (deletes) a directory

mkfifo

ma**k**e **fifo**

Description — Makes a new fifo. A fifo is a queue-like object for one-way communication between processes. It causes items to be taken out of the queue in the order that they were put in (**f**irst **i**n, **f**irst **o**ut).

Syntax — **mkfifo *fifo_name***

Parameters — Takes one required parameter:

fifo_name the name of the fifo

Examples —
```
mkfifo  \dev\propos
```

Same As RAPL-3

mknod()	makes fifo, device, or socket
pipe()	creates a private set of fifos

See Also — chmod changes protection mode

mksock

ma**k**e **sock**et

Description — Makes a new socket. A socket is an object for two-way communication between processes.

A socket supports a client-server configuration for one server and many clients with the sockets like a set of two-way pipes connected to the server hub and the clients unable to talk to each other.

Syntax — **mksock *socket_name***

Parameters — Takes one required parameter:

socket_name	the name of the socket

Examples

```
mksock  \dev\carousel
```

Same As RAPL-3

mknod()	makes socket, fifo, or device
socketpair()	makes a private pair of sockets

See Also

chmod	changes protection mode

more

display **more** and more of a file

Description — Displays the contents of the file at the terminal window 20 lines at a time. More accurately, copies the contents of the file to the console screen.

Syntax — **more *file_name***

Parameters — Takes one required parameter:

file_name	the name of the file (absolute or relative path)

Examples

```
more  \log\acid.log
```

mount

mount file system

Description

Mounts a second file system at a point on the first file system. Mounting a file system allows access to its files which are on a different drive, device, or computer from the primary file system. The mount point becomes the root of the second file system.

The mount command can only be used with systems using multiple file systems, like those running CROS-NT.

Syntax

mount [–r] hostfs *mount_point filesystem_to_mount*

Parameters

Takes one optional and two required parameters:

–r	set mode to read only; CROS will not be able to write into the mounted filesystem.
hostfs	this keyword denotes that a host filesystem is being mounted under the CROS filesystem.
mount_point	where under the CROS filesystem to mount the hostfs filesystem. The mounted filesystem will be visible under this directory, but any objects currently in the directory will become inaccessible as long as the filesystem is mounted.
filesystem_to_mount	this parameter specifies what part of the host filesystem to mount. Typically a full DOS-style path.

Examples

```
mount -r hostfs/mnt c:/tmp
mount    hostfs/mnt c:/mydir/results
```

See Also

unmount unmounts file system

msleep

millisecond **sleep**

Description

Puts the shell to sleep for a number of milliseconds. Similar to the effect of an msleep() call or delay() call within a program.

Syntax

msleep *number*

Parameters

Takes one required parameter:

number	an integer specifying the number of milliseconds

Examples

```
msleep 1000          # sleep for 1 second
```

mv
ren

mo**v**e object
rename object

Description — Moves or renames an object. At present you cannot move a directory.

Syntax
```
mv  old_object_name  new_object_name
mv  object_names… directory
```

Parameters — To rename an object takes two required parameters. If the new name already exists, the system displays a message and does not overwrite it.

old_object_name	the name being deleted
new_object_name	the name being created

To move entities from one directory to another, takes at least two required parameters:

object_names…	a space-separated list of the objects being moved
directory	the directory to move the objects into

Examples
```
mv    lab_test  lab_1
move  \app\test\dispense.r  \app\dispense\dispense.r
mv    beta.r  beta.v3  \app\test
```

See Also

cp (or copy)	copies an object
ln	makes a link to an object
rm (or del)	removes (deletes) an object or link

odometer

robot armpower **odometer**

Description — This command allows the user to examine the robot armpower odometer, which gives how long the robot armpower has been on since firmware installation.

Syntax
```
odometer
```

Parameters — None.

Example
```
odometer
```
Result
```
The robot has been running for 123.4 hours.
```

panel

front **panel** menu shell

Description — This command uses the front panel F1 and F2 buttons and the lcd display to provide a simple menu for selecting and executing shell commands.

Syntax

```
panel [-options] selection1 [selection2 …]
panel [-options] –f filename
```

Parameters

The arguments to **panel** consist of a (possibly absent) set of options and a required list of menu selections. The menu selections can be stated on the command line (*selection1*…) or may be found in a file (-f *filename*).

The *selection* items all look like this:

label=command

or

label:*command*

In both cases, the *label* is the title shown on the lcd display for the selection; the *command* is the shell command that is executed if the item is selected by the user. If the *selection* uses the '=' symbol (the first case) then **panel** will permit the user to abort the command from the front panel via the F1 key. If the ':' symbol is used, then the command cannot be aborted from the panel. This can be used to allow **panel** to call itself for nested menus.

There is one special *command* symbol, **exit**, which, if chosen by the user, causes **panel** to terminate.

The valid options are:

Option	Description	
-d *label*	specify default menu item	Specifies that if the front panel buttons have not been touched, the menu item with label ***label*** should be executed after the timeout specified with the **–T** option has elapsed.
-f *filename*	read items from file	Reads the list of selection items from a file. It is expected that they are listed one item per line in the file.
-h **-?**	help	Display a usage message.
-n	don't take control of the robot	Normally, **panel** takes control of the robot when it starts up and after each command has executed (for safety reasons.) The **–n** option disables this behaviour.
-s	sort	Sort all of the *selection* entries alphabetically, by their *label*s.
-T *nnnn*	set timeout	Specifies that if the panel has not been touched, then after *nnnn* seconds the default action will be taken if the **–d** option has been supplied.
-t *title*	set title	Sets the title shown on the lcd display when **panel** first starts up.

Examples

```
panel –t test Ready=ready Home=home "Rotate=motor 1 100"
Exit=exit
```

This will display a menu with entries labelled "Ready", "Home" "Rotate" and "Exit". If the user selects "Ready", the **ready** command will be executed, and so forth. Note that the selection item with the **motor** command must be

placed in quotes so that **panel** knows that it is just one item altogether.

`panel –T 120 –d Startup Startup=myapp Home=home Exit:exit`
This displays a 3 item menu with “Startup”, “Home” and “Exit” as the labels. If the front panel is not touched, after 120 seconds the “Startup” selection will execute (running the “myapp” command.)

`panel –f /conf/panel.cfg`
Load menu items from the specifed file (which is, in this case, a sample file provided with the system.)

---

ps

process **s**tatus

Description — Displays the current status of every process on the system.

Syntax — **ps**

Parameters — none.

Example — **ps**

Sample Result

pid	ppid	status	flags	prio	time	mem	command
52	8	RUN	---r	2	0.120	7.43750K	ps
7	6	READY	t-pr	2	1.72e+3	11.3750K	/sbin/robotsrv
6	1	READY	t-pr	2	4.66e+3	11.3750K	/sbin/robotsrv
5	3	IWIO	--pr	2	3.28	13.4375K	/sbin/fastacid
4	3	READY	--pr	2	17.3	13.4375K	/sbin/fastacid
3	1	WSEM	--pr	2	0.683	13.4375K	/sbin/fastacid
8	1	WAIT	--pr	2	12.8	10.5000K	shell
1	0	WAIT	--p-	2	0.138	4.00000K	(init)

System has been running for 1933 seconds

Description — The ps command gives nine columns of information: pid, ppid, status, flags, prio, time, slip, mem, and command.

pid

process **id**entification number
The identification number of the process. Assigned by the operating system. During a session, each new process is assigned a new number.

ppid

parent **p**rocess **id**entification number
The identification number of the parent process. The parent process is the process that initiated the process identified by pid.

status

process **status**
The process is in one of the following states.

IWIO	**i**nterruptible, **w**aiting for **i**nput/**o**utput
READY	**ready** to run, not currently running
RUN	currently **run**ning (executing)
SLEEP	waiting for **sleep** (time delay) to elapse
STOP	execution **stop**ped for diagnostic purpose
WAIT	**wait**ing for child process to finish running
WIO	**w**aiting for **i**nput/**o**utput
WSEM	**w**aiting for **sem**aphore
WSOCK	**w**aiting to send or receive a message on a **sock**et
ZOMB	**zomb**ie: process has terminated, but the table entry exists until the parent task deletes it from the process table

flags

attribute **flags**
Indicators of attributes of the process. The first two are used for CRS testing.

t	**t**imed out of wait; not yet revived
I	**i**nterrupted; not yet re-started
p	**p**rivileged: can change its priority above normal; can mount and unmount directories
r	**R**APL-3 process, not a binary process

prio

priority level
The indicator of the level of priority for processing. Higher priority processes get CPU time before lower priority processes.

1	high
2	normal
3	low

time

process **time**
The total time the process has been executing.

mem

memory used
Amount of memory used in Kbytes.

command

command name
The name of the process when called by the user at the prompt or by another process.

system running time

The total time the system has been running during this session, expressed in seconds. Large numbers of seconds are expressed in exponential notation, like 2.3e+5

See Also kill terminate a process

pause

pause for the user to hit the return key

Description — Optionally displays a prompt on the console and waits for the user to hit the return key. This is useful inside scripts.

Syntax — **pause** *[-n] [text…]*

Parameters — There are two optional parameters:

-n	Do not print. Go to a new line after printing the message; leave the cursor right at the end of the message.
text…	The text of the message to display before waiting for user input.

Example — `pause –n Press ENTER to continue:`

Results — `Displays the message "Press ENTER to continue:", and leaves the cursor next to the ":". Waits for the user to hit the return key, then exits.`

See Also

do	do a shell script
echo	echo a message to the console

pedit

pendant **edit**

Description — This command invokes the teach pendant text editor. **pedit** can be used to create and modify configuration and source files.

If no filename is specified, the pendant displays files to edit.

Syntax — **pedit** *[filename]*

Parameters — The command takes one optional parameter:

filename	the name of the file to edit.

Examples

`pedit /conf/rc`	Edit the startup script file.
`pedit`	Start the pendant editor.

See Also — edit — starts the terminal editor.

pwd

print **w**orking **d**irectory

Description — Displays (prints to terminal screen) the current working directory. Displays the full absolute path.

Syntax — **pwd**

Parameters — Takes no arguments.

Example — pwd

Results —
```
/
/dev
/app/test
```

See Also — cd changes current working directory
ls (or dir) lists contents of directory

r3c
compile

invoke the **R**APL-**3** **c**ompiler

Description — The **r3c** command allows the user to recompile RAPL-3 programs. There are a great number of command line options, many of which are of interest only in very special circumstances.

Syntax — **r3c** *[-options} file1.r3 [file2.r3…]*

Parameters — There are two sets of parameters:

options	options, from the table below.
file1.r3…	a list of files to compile.

The most common options are:

Option	Description	
-h **-?**	help	Display a complete usage message for the compiler, listing all options.
-L *libname*	use library	Search the specified library for
-l	line numbers	Force line number information to be included in the output file, even if stripping all symbols (see **–s** and **–x**, below.)
-o *filename*	specify output file name	Causes the compiled file to be written to *filename* instead of to the default name. (The default output name for file “x.r3” is “x”.)
-P	pipe	Use pipes instead of temporary files when compiling (saves file space during

Option	Description	
		the compile.)
-s	strip symbols	Strip all symbols from the output file (to save space)
-v	verbose	Be verbose when compiling.
-Wall	warn on all	Generate all possible warnings about questionable code.
-Wmax	maximum warnings	Warn on even remotely questionable code.
-Wnone	warn on none	Generate no questionable code warnings.
-x	exclude symbols	Exclude all symbols that are neither global nor exported. Used to minimize the size of a library.

Examples

```
r3c myprog.r3
   compile myprog.r3, producing output file “myprog”

r3c –Wall –o test fred.r3
   compile fred.r3 with all warnings turned on, producing
output file “test”
```

rc1

RAPL-3 **c**ompiler pass **1**

Description

The **rc1** program is the main pass of the RAPL-3 compiler. It is not generally accessible from the command line, but is instead called by the **r3c** compiler driver program.

Syntax

rc1 *[-options][filename]*

Parameters

There are two optional parameters:

options	options, from the table below.
filename	name of the source file. If this is omitted, then the compiler reads source from the standard input.

The most common options are:

Option	Description	
-?	help	Display a complete usage message for the compiler, listing all options.
-L *libname*	use library	Search the specified library for
-l	line numbers	Force line number information to be included in the output file, even if stripping all symbols (see **–s** and **–x**, below.)

Option	Description	
-o *filename*	specify output file name	Causes the compiled file to be written to *filename* instead of to the default name. (The default output name for file "x.r3" is "x".)
-s	strip symbols	Strip all symbols from the output file (to save space)
-v	verbose	Be verbose when compiling.
-Wall	warn on all	Generate all possible warnings about questionable code.
-Wmax	maximum warnings	Warn on even remotely questionable code.
-Wnone	warn on none	Generate no questionable code warnings.
-x	exclude symbols	Exclude all symbols that are neither global nor exported. Used to minimize the size of a library.

Example
```
rc1 –L/lib/syslib.r –L/lib/robotlib.r –Wall –o myprog
myprog.r3
```

Result
```
The program myprog.r3 is compiled with warnings turned on
and with the libraries syslib and robotlib available to
the program.
```

See Also `r3c, rpp`

---

ren

rename object

Moves or renames an object.

See **mv**

---

rm
del

rem**o**ve
delete

Description Removes (deletes) a reference to a set of objects (files, devices, fifos, or sockets). When all references to an object have been deleted, the object ceases to exist.

Syntax **`rm`** ***`object_names…`***

Parameters Takes one parameter:

`object_names…` a space-separated list of objects to be removed

Examples

```
rm  test.r
```

See Also

ln	makes a link to an object
cp (or copy)	copies an object
mv (or move)	moves or renames an object

rmdir

re**m**ove **dir**ectory

Description

Removes (deletes) a directory. Only empty directories may be deleted; an error message will be displayed if one attempts to remove a directory that is not empty.

Syntax

rmdir ***directory_names…***

Parameters

Takes one required parameter:

directory_names… a space separated list of the directories to be deleted.

Examples

```
rmdir  temp_test
```

See Also

mkdir (or md)	makes a directory

rpp

RAPL-3 **p**re-**p**rocessor

Description

The **rpp** program is the part of the RAPL-3 compiler that handles .define macros and .ifdef conditionals. It is not generally accessible from the command line, but is instead called by the **r3c** compiler driver program.

Syntax

rpp *[-options] [infile [outfile]]*

Parameters

There are three options parameters:

options	a set of options, from the table below
infile	input file name (input is taken from the standard input if this is omitted.)
outfile	output file name (output is sent to the standard output if this is omitted.)

The most common options are:

Option	Description	
-? **–h** **–H**	help	Display a complete usage message for the compiler, listing all options.
-L	no line numbers	Disable the automatic inclusion of line number information in the output file.
-D*name* -**D***name=value*	define symbol	Has the same effect as placing a ".define *name* 1" or ".define *name value*" directive at the start if the input file.

Option	Description
	Note that if no value is given, the value is defined to be “1”.

Example
```
rpp myprog.r3 myprog.out
```
Result
The RAPL-3 program **myprog.r3** will be preprocessed and the output placed in **myprog.out**.

shell

system **shell**

Description
Starts a new system shell from a system shell or from an application shell. The system shell interprets system commands.

Syntax
shell

Parameters
Takes no parameters.

Examples
```
shell
```

See Also

exit	terminates a shell
ash	starts a new application shell

shutdown

shut down

Description
Shuts down the system in a controlled fashion. The system should **always** be shut down before turning off the controller power.

Failure to shut down before power off can result in loss of all data on the controller.

Syntax
shutdown *[-option] when*

Parameters
Takes one optional parameter and one required parameter:

option	optional actions to take when shutting down
-rebuild	invalidates the current file system and rebuilds the file system at the next start up. This erases all data in the memory file system and forces its contents to be reset to factory defaults. **Do not execute this option unless absolutely necessary.**
when	the time when the system will be shut down
now	immediately

Examples

shutdown now	shutdown normally
shutdown -rebuild now	shutdown & erase memory contents

siocfg

serial **i**nput/**o**utput **c**on**f**i**g**uration

Description

Changes the configuration of the serial ports.

The C500C controller has a total of four configurable serial ports: ports 0 and 1 (/dev/sio1 and /dev/sio1) are available for application use, port 2 (/dev/sio2) is reserved for the teach pendant, and port 3 (/dev/sio3) is used for the console.

The C500 controller only has two configurable serial ports: port 0 (/dev/sio0) is reserved for the teach pendant and port 1 (/dev/sio1) is used for communications with the console.

CROSnt supports up to 64 serial ports: com1 corresponds to port 0 (/dev/sio0), com2 corresponds to port 1 (/dev/sio1), etc.

Be careful when changing the console port; if this gets set so that you can no longer communicate with the robot, then the only way to recover is to restore the C500C controller to its most basic port settings by holding down the F1, F2, Pause/Continue, and Home buttons during the controller's boot-up cycle. If you perform a reset to basic settings, the console port reverts to SIO0 on the back of the controller. You will need a null modem serial cable to connect your console to SIO0.

serial port	**port association and baud rate**		
	C500	**C500C (standard)**	**C500C (reset)**
sio0	pendant, 19200	57600	console, 57600
sio1	console, 38400	57600	57600
sio2	N/A	pendant, 19200	pendant, 19200
sio3	N/A	console, 57600	57600

Syntax

siocfg -c *port [***-b** *baud] [***-d** *data] [***-p** *parity] [***-s** *stop] [***-v***]*

Parameters

Only the –c parameter is required. The parameters are:

-c	*port*	com port	identification number of port to be configured
-b	*baud*	baud rate	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 (C500C), or 115200 (C500C)
-d	*data*	data length	5, 6, 7, or 8 bits
-p	*parity*	parity	0 = none; 1 = odd; 2 = even
-s	*stop*	stop bits	1 = 1; 2 = 2
-v		be verbose	

Examples

```
siocfg  -c 1  -b 38400  -d 8  -p 0  -s 1
siocfg  -c 8  -b 9600
siocfg  -c 9  -d 7
```

sync

synchronize filesystem buffers

Description

Synchronizes (flushes) filesystem buffers in CROS-NT and defragments memory in CROS-NT and CROS-500.

For CROS on a C500, primarily used to defragment memory.

For CROS on NT, the system buffers are flushed. In other words, any information that CROS has not written to storage is written out.

Syntax

sync

Parameters

Takes no parameters.

Examples

sync

See Also

mem displays fragmentation of free space in memory

touch

touch file modification times

Description

The **touch** command changes the modification time of a file to the current time. It can be used, for example, to force programs like ash to rebuild the v3 file for an executable.

Syntax

touch *file1 [file2…]*

Parameters

The **touch** command accepts a list of files to set the timestamps of.

Example

```
touch myprog
```

Result

The timestamp of **myprog** is set to the current time. If **ash** is used to run **myprog,** ash will detect that **myprog**'s v3 file is older than the program, and will use **v3make** to rebuild the v3 file.

unmount

unmount file system

Description

Unmounts a second file system from the mount point.

The mount and unmount commands can only be used with systems using multiple file systems, like those running under CROS-NT.

Syntax

unmount ***mount_point***

Parameters

Takes one required parameter:

mount_point the mount point of the file system to be dismounted

Examples

unmount /mnt

See Also

mount mounts a file system

v3make

make or update a **v3** file for a program

Description

The **v3make** command creates or updates (if the v3 file already exists) the v3 file for a RAPL-3 executable. It is generally not used directly by the user, but is instead run automatically from **ash.**

Syntax

v3make *[-options] exec_filename*

Parameters

There are two sets of parameters, one of which is required:

options	a set of options, from the table below
exec_filename	the name of the executable to construct / update the v3 file for.

The valid options are:

Option	Description	
-? **–h**	help	Display a usage message for program.
-an	**a**dd missing / **n**o adjust	Add any missing variables to the v3 file; do not adjust any variables whose types have changed. (this is the default mode.)
-af	**a**dd missing / **f**ix changed	Add any missing variables to the v3 file; attempt to fix any variables whose types have changed. [currently unimplemented]
-ai	**a**dd missing / **i**nteractive fix	Add any missing variables to the v3 file; ask the user about fixing any variables whose types have changed. [currently unimplemented]
-ar	**a**dd missing / **r**eplace	Add any missing variables to the v3 file; replace variables whose types have changed (causing their old values to be lost.)
-d	**d**elete	Delete any v3 variables that aren't needed.
-o *filename*	**o**utput file	Use *filename* as the v3 file instead of automatically constructing the name from the *exec_filename.*
-r	**r**eplace	Just replace any v3 file that currently exists; all data in the old file is lost.
-v	**v**erbose	be verbose
-V	**v**ersion	print v3update's version string and exit.
-w	**w**arnings	give extra warnings.

Example

```
v3make –an myprog
```

Result — Scans **myprog.v3** and checks it against the program **myprog**. Any missing variables will be added to **myprog.v3**, while variables with changed types will result in error messages.

See Also — ash

ver

version

Description — Displays the version of the system shell being used.

Syntax — ver

Parameters — Takes no parameters.

Examples — ver

Result — **CROS System Shell -- $Revision: 1.15 $**

See Also

crosver	displays version of CROS
ver (in the application shell)	displays version of application shell