Application Shell
ASH

UMI-R3-161

Application Shell (ASH)

Revision		
Number	**History**	**Date**
001	First release as UMI-R3-161. CROS 1.16 for C500C.	99-05

Contents

CHAPTER 1

Introduction

With the application shell (ASH), you can teach locations, modify variables and values, monitor arm status, and move the arm with robot motion commands. You can also run an application.

The application shell provides a command line interface, interpreting input from the keyboard and output to the terminal screen. It is the command-line equivalent of the teach pendant.

CHAPTER 2

ASH Basics

This is a quick summary of the basics of ASH.

Topic	Command	Comments
Starting	ASH	Starts the application shell.
Applications		Create a separate application for each pair of program and variable files, unless you know how to handle multiple files in an application.
Updating	refresh	Updates the database from a new program file. Use this after sending a new program file.
Motions	move	Caution. Use small increments and slow speed. The arm will try to complete any command as specified.
Teaching	here set	Teach locations with **here** after positioning the arm. Set values to non-location variables with **set**.
Control	pendant	Transfers control from ASH to the teach pendant. At the pendant, Shift + ESC transfers control back to ASH.
Running	run	Runs the default application: default program file with default variable file.
Getting Help	help	Lists all commands and descriptions.
Exiting	exit	Exits the application shell.

ASH Commands

This chapter, *The Application Shell (ASH)*, outlines the functionality of the application shell and many of the typical procedures you perform using it.

The next chapter, *Application Shell Commands*, details each command.

Repeating Commands

The application shell remembers the last 25 lines used at the command line prompt. To re-display previously typed lines, press the up arrow for earlier lines and the down arrow for later lines.

Note: If you have more than one shell running at one time, each shell remembers the lines used for that shell.

Command Completion

The application shell has command completion. Pressing the tab key multiple times causes a list of possible completions to be shown. If you type enough letters to distinguish a specific command, pressing the tab key completes the command.

Getting Help

The application shell has built-in help. To access ASH help, you must be in ASH. The ASH help cannot be accessed from the system shell.

Listing Help Topics
help

To display the list of help topics, type the help command with no parameters.

```
help
```

Displaying Help on a Command

To display help on a specific command, type help followed by the command name.

```
help  set
help  joint
```

There may be a delay of a few seconds until help is displayed.

Reading the Entire Help File

The ASH help file is a text file in the \lib directory and can be viewed with the system shell's type or more command.

```
more  \lib\ash.help
```

In the file, the @@ symbols separate sections displayed by the help function.

Understanding the Application Shell

This section explains some of the basic concepts of the application shell and how the shell works.

Applications

An application is a specific set of tasks that you have programmed the robot to perform.

Since you can program the robot for different sets of tasks, you can have more than one application. With ASH, each application is identified by a name. For example, an application that only checks locations and motions could be called “test”, while an application that actually dispenses material on a work piece could be called “dispense”.

Files

An application has a program file and a variable file. The program file contains the step-by-step instructions written in RAPL-3. The variable file contains all teachable variables.

Teachable Variables

Most variables, including locations, can be made teachable. A teachable variable is a variable that can be accessed outside the program. Its value can be changed without having to change the program file.

Teachable variables are stored in the variable file. When you run an application, the operating system takes the variable file and uses its values to initialize the variables in the program file just before running.

You change teachable variables with the database.

Database

When you start an application shell, ASH creates a database and loads all variables and their values from the variable file into the database. If you are an advanced user and have more than one variable file, you can specify which variable file to load into the database.

While in the database, you can create or erase variables, change values of variables, and teach locations.

When you finish modifying a variable and its value, this data is saved from the database to the variable file. The data must be stored in the variable file for it to be used with the program file when it is run as an application. The application shell automatically saves the data to a variable file. If you are an advanced user with more than one variable file, you can specify which variable file to save to.

Variable Files

You can create a variable file in a number of ways:

- Refreshing from the Program File: When your program file is on the controller, ASH's refresh command reviews the program and adds any teachable variables to the database. After working with the teachables in the database, you save the new data to the variable file. This method is used if you write your program before teaching your locations.
- Building on the Controller: You can build a variable file entirely on the controller using ASH. In the database, create variables and work with them. When you are finished, save this data to a variable file. This method is used if you teach your locations before writing your program.

When the variable file is saved, it is saved to a specific directory.

Directories

On the controller, the \app directory contains all applications.

Within \app, each application has its own directory. For example, the application named "test" has \app\test while the application named "dispense" has \app\dispense.

When you start an application shell, you must name a specific application, for example "test" or "dispense". When an application shell is running, the current directory for that shell is the directory with the specified application name. The current directory, the specific application, cannot be changed within an application shell. If you want to access another application, you must run another application shell.

Each application has a program file and a variable file. For example, the application named "test" has "test" and "test.v3" which are stored at \app\test\test and \app\test\test.v3. When you send your program file from Robcomm3, you must specify the correct directory. When you save your variable file from ASH, ASH automatically saves to the current directory.

Multiple Applications

It is good practice to keep applications separate. For each application (a set of tasks that solves an automation problem) create an application (a directory in \app containing a specific program and its variable file).

For example: for preparation, create the application "prep" containing the program "prep" and the variable file "prep.v3"; for loading part 220, "load_220" with "load_220" and "load_220.v3"; for loading part 440, "load_440" with "load_440" and "load_440.v3", and for cleaning up, "clean_up" with "clean_up" and "clean_up.v3".

For variations of programs and variables, you can have multiple files in a directory.

Multiple Files

You can store variations of your program file and variations of your variable file in the same directory, under the same application name. For example, "test1", "test2", "alpha.v3", and "beta.v3" can be stored in the same directory.

When you start an application shell, you can specify which variable file to load into the database. You can also merge data from another variable file into the database. When you save, save to any variable file.

When you run an application, specify which program file and which variable file to use when running.

Caution. Use multiple files carefully. *It is easy to confuse one file with another, or confuse your filenames with the default filename. Whenever possible, use a separate application for each pair of program and variable files.*

Robot Motion

The application shell is designed as a tool for developing applications in an architecture where teachable variables are stored in a variable file separate from a program file. The database of ASH is used to modify teachable variables including locations. Before teaching a location, the arm must be moved with the teach pendant or ASH. To do this, the most common robot motion commands are accessible from ASH.

Running an Application Shell

To use the application shell to teach locations, teach other variables, and move the arm, you must have an application shell running.

This section describes how to:

- start an application shell
- exit from an application shell
- start a new system shell
- check the shells that you have running
- display the version of the application shell software.

In this part of the *Application Development Guide*:

the expression	is the same as
Starting an application shell	Opening an application
Having an application shell running	Having an application open

Starting an Application Shell ASH

You can start an application shell from any system shell prompt (the $ prompt).

When you start an application shell, you must specify the application by either selecting an existing application or creating a new application.

When starting an application shell, you can list all existing applications and then select one, or by-pass the listing and just select an application.

The application shell will not start if the pendant program is running. This is a safety feature to prevent accidental removal of point of control from the pendant. To terminate the pendant, hit the Esc key until the termination screen is displayed, and then press the F1 key.

Listing All Existing Applications

1. At the prompt, type:
   ```
   ash
   ```
2. The application shell displays the message “Existing applications are:”, lists all existing applications, and displays an “Application name >” prompt.
3. Type the name of the application.
4. The application shell responds in one of two ways:
   - If it is an existing application, ASH loads the default variable file into the database, displays the message “Loading v3 file ‘*application_name*.v3’ . . . done.” and displays a prompt with the application name in it.
   - If it is a new application, ASH displays the message “Application ‘*application_name*’ not found -- try to create it? If you respond **y** for yes, ASH creates the new application, creates a variable file “Creating

v3 file '*application_name*.v3' . . . done.", loads it into the database, displays "Loading v3 file '*application_name*.v3' . . . done." and displays a prompt with the application name in it.

If you do not want to start a new ASH session, but typed **ASH** by mistake, you cannot back out of the start-up procedure half-way through. Name any application, such as "test", to complete the start-up procedure and, once ASH is started, exit from it.

By-Passing the List

You can specify an application when you type the ASH command. This by-passes the listing of existing applications.

1. At the prompt, type
   ash *application_name*
   for example:
   ash test
   ash dispense

   - If it is an existing application, ASH displays the message "Loading teachables from '*application_name*.v3' . . . done." and displays a prompt with the application name in it.
   - If it is a new application, ASH displays the message "Application '*application_name*' not found -- try to create it? If you respond **y** for yes, ASH creates the application, displays the messages for creating and loading the variable file, and displays a prompt with the application name in it.

If you do not want to start a new ASH session, but typed ASH and the application name by mistake, you cannot back out of the start-up procedure. Complete the start-up procedure and, once ASH is started, exit from it.

Exiting From an Application Shell
exit, quit

To exit from the current application shell, use the exit command.

The exit command terminates the current application shell and returns you to the point where you started the application shell. If you start ASH from the system shell, the system shell is the parent process and ASH is its child process. When you terminate from ASH, you are returned to its parent process, the system shell.

Any process started by ASH is a child process of ASH. If you terminate ASH (exit from ASH), any child process of ASH is sent a SIGHUP signal. Any child process that does not either mask SIGHUP or have an installed signal handler for it will be terminated by CROS.

The application shell will not allow you to exit if the pendant has point of control. At the teach pendant keypad, press Shift + ESC to transfer control to ASH. The transfer function can also be reached by repeatedly pressing ESC to move up the hierarchy of screens to the Terminate Pendant screen.

The quit command is an alias of the exit command. Remember that the system shell also has an exit command which exits you out of the system shell.

Running Another System Shell
shell

You can have only one application shell running at one time.

You can have more than one system shell running at one time. The total number of shells that you can have running at one time is limited by available memory. An application shell with its database takes far more memory than a system shell. The system limits you to one application shell.

From the application shell, you can access a system shell in one of two ways:

- You can exit from the application shell. This terminates that application shell and any of its child processes that do not handle or mask the SIGHUP signal. Alternatively, you can start a new system shell. This keeps the existing application shell, and all of its child processes, active in the background and places you in the new system shell. At the application shell prompt, use the shell command.
- If you have more than one shell running, you cannot jump from one shell to another. You must exit from the shell that is the child process to return to its parent.

Checking Application Shells
ps

You can check the status of an application shell with the system shell's process status (ps) command. The application shell (and each system shell) is a process displayed in the process table.

Although it is a system shell command, the ps command is available from the application shell.

Checking the Version of ASH
ver

To display which version of ASH you are running, use the version command, ver.

Remember that the system shell also has a version command which displays the version of the system shell you are running.

Loading and Refreshing the Database

When you start an application shell, what happens in the database depends on the variable file.

If you are creating a new application, ASH creates a variable file with the default name, the same name as the application. This file is empty. Next, ASH loads the database with this file and the contents of the database remain empty. Any default saving is done to this default file.

If you specify an existing application, ASH loads that application's default variable file (the same name as the application) into the database. If that file is empty, the database remains empty. If that file has data from previous activity, those variables and values are loaded into the database.

If you specify an existing application and specify one of the multiple variable files of that application, ASH loads that specific variable file.

Loading the Default Variable File

If you do not specify a variable file, ASH loads the variable file with the same name as the application. For example, if you are in the application "test", ASH loads "test.v3", or in the application "dispense", ASH loads "dispense.v3". Even if you have multiple files stored with one application, if you do not specify a variable file, ASH loads the file with the same name. For example, if you have "test.v3", "test1.v3", "test2.v3", and "alpha.v3" with the application "test", ASH loads "test.v3".

Loading a Specific Variable File

If your application has more than one variable file, you can specify which file to load.

Listing Files

To list the variable files, use the dir or ls command and specify the directory. Changing directories with the cd command, and listing directory contents with the ls or dir command, is described in the chapters on the system shell.

For example:

```
ls  -R  /app
ls  /app/test
```

Loading a File

To load a specific variable file:

1. At the prompt, type
   ash *application_name variable_file_name*
   For example:
   ash test test1
   ash dispense alpha.v3
   The .v3 extension is optional.

The application shell displays the message "Loading v3 file '*variable_file_name*.v3' ... done.", and displays a prompt with the application name in it, *application_name* >.

Loading Another File

When you save from the database to a variable file, ASH copies the data to the file, but the data is also still in the database. You can add some or all of the data from another variable file with the merge command.

You can erase some or all of the current data from the database with the erase or eraseall command and then merge some or all of the data from another file.

For further details, see the sections Working with Variables: Deleting Variables, and Merging Data From Another File.

Refreshing the Database refresh

When you are developing your application, you are likely in a repeating process of editing your program file, compiling it, and sending it to a \app directory. If you add teachables to your program, you need those new teachables in the database. Update the database with the refresh command.

The refresh command reviews the program file's time stamp. If the program file is newer, the application shell makes a new .v3 file and adds any new variables to the database.

Working with Variables

Once ASH loads variables from a variable file into the database, you can work with the variables. Commands are available to: list existing variables, make new variables, erase variables, change the values of variables, and print the values of variables.

In this section, variables include locations, integers, floats, and strings.

Listing Variables
list

To list variables in the database, use the list command. The list command without any parameters lists all variables of all data types. The list command with a parameter specifying a data type lists all variables of that data type. Possible data types are: int, float, string, cloc, ploc, gloc.

```
list

list  cloc
```

The list displays: the data type, name, whether it is taught or not, and the values of simple types like floats, ints and strings. An asterisk indicates that the variable is not yet taught, i.e. no value has been assigned to the variable.

```
Variables:   (* indicates not yet taught)
      int            number_of_loops      = 10
      int            counter              = 1
      ploc        *  pick_1
      cloc        *  place_1
```

To display the value of any variable type, use the print command.

Remember, this list command of the application shell is different from the ls command of the system shell that lists a directory.

Making New Variables
new

To make a new variable, use the new command. Using this command is similar to a declaration in a RAPL-3 program.

```
new   counter
```

Identifiers

The variable name follows the rules for RAPL-3 identifiers:
- begins with a letter
- has one or more letters, digits, or _ (underscore) characters
- has any combination of uppercase (ABCDE) or lowercase (abcde)

Data Types

A prefix, identical to the RAPL-3 implicit declaration prefix, is used to indicate the data type.

Example	Prefix Character		Data Type	
new counter		none	int	integer
new %difference	%	percent sign	float	floating point number
new $message[20]	$	dollar sign	string[]	string of characters
new _safe	_	underscore	cloc	cartesian location
new #dispense	#	number sign	ploc	precision location

For a string variable, you must specify its length in characters.

You cannot create a gloc with the new command.

Arrays (One Dimension)

You can make arrays by specifying a size in square brackets. The size is a positive integer, but the indexing begins at zero.

Example	Description
`new counter[3]`	a one-dimensional array of integers counter[0], counter[1], and counter[2]
`new %diff[5]`	a one-dimensional array of floats diff[0], diff[1], diff[2], diff[3], and diff[4]
`new $message[20][2]`	a one-dimensional array of strings message[0] and message[1] each with a length of 20 characters
`new _safe[16]`	a one-dimensional array of cartesian locations safe[0] to safe[15]
`new #dispense[24]`	a one-dimensional array of precision locations dispense[0] to dispense[23]

In the example of an array of strings, the string length in characters is specified first and then the number of strings. Compare this to the single string in the previous table.

You can make a one-dimensional array of any data type: int, float, string, cloc, or ploc.

Arrays (Two Dimensions)

You can also make two-dimensional arrays. There are two methods to make a two-dimensional array: top-down and bottom-up.

Top-Down Method

The top-down method follows the same format used by ASH to display the values in an array.

First, specify the higher-level element. Second, specify the number of lower-level elements in each of the higher-level elements. With the top-down method, the two dimensions are separated by a comma within one set of square brackets.

Example	Description

`new counter[2,2]`	a two-dimensional array of integers
	`counter[0,0], counter[0,1],`
	`counter[1,0], counter[1,1]`

`new %diff[2,5]`	a two-dimensional array of floats
	`diff[0,0], diff[0,1], diff[0,2],`
	`diff[0,3], diff[0,4]`
	`diff[1,0], diff[1,1], diff[1,2],`
	`diff[1,3], diff[1,4]`

`new _safe[5,16]`	a two-dimensional array of cartesian locations
	`safe[0,0] ... safe[0,15]`
	`... ...`
	`safe[4,0] ... safe[4,15]`

`new #dispense[10,24]`	a two-dimensional array of precision locations
	`dispense[0,0] ... dispense[0,23]`
	`... ...`
	`dispense[9,0] ... dispense[9,23]`

Bottom-Up Method

The bottom-up method is similar to the method used to make a one-dimensional array of strings.

First, specify the size of the lower-level element in the array. Second, specify the higher-level number of these elements you want in the array. With the bottom-up method, each dimension is written in its own complete set of square brackets.

Example	Description

`new counter[2][2]`	a two-dimensional array of integers
	`counter[0,0], counter[0,1],`
	`counter[1,0], counter[1,1]`

`new %diff[5][2]`	a two-dimensional array of floats
	`diff[0,0], diff[0,1], diff[0,2],`
	`diff[0,3], diff[0,4]`
	`diff[1,0], diff[1,1], diff[1,2],`
	`diff[1,3], diff[1,4]`

`new _safe[16][5]`	a two-dimensional array of cartesian locations
	`safe[0,0] ... safe[0,15]`
	`... ...`
	`safe[4,0] ... safe[4,15]`

`new #dispense[24][10]`	a two-dimensional array of precision locations
	`dispense[0,0] ... dispense[0,23]`
	`... ...`
	`dispense[9,0] ... dispense[9,23]`

You can make a two-dimensional array of int, float, cloc, or ploc, but not string.

You cannot make an array of teachables with more than two dimensions.

Excess Variables

You can make any variable. If, when you run the application, the variable is not used by the program, the system displays a message that the variable is being ignored.

Teaching Variables here, set

When you use the new command, or when you use ASH's (or the compiler's) .v3 file generator, variables are created but have no values assigned to them.

To assign values to variables or "teach" these teachable variables, there are two commands: here and set. With the here command, you can pack position data into a location variable. With the set command, you can set a value with a constant or set a value with another variable.

Using the here Command With Locations

The here command is used with locations. This command obtains data about the current position of the arm and assigns that data to the location variable.

Since the here command obtains current position data, you must move the arm to the desired position before using the here command.

Simple locations.

```
here  safe_loc
here  point1
```

Elements of arrays.

```
here  place[2][3]
here  a[4,10]
```

Using the set Command With Ints, Floats, and Strings

The set command is used with ints, floats, and strings. With the set command, you specify the value to be assigned: an integer value, a floating point value, or a string of characters. You can also use the set command to initialize locations, but they have limited uses.

Integer and float constants can be positive or negative. String constants are enclosed in double quotes.

Simple Variables

Integer constants and integer variables.

```
set  count_step  =  -2
set  number_of_loops  =  100
set  number_of_loops  =  number_of_samples
```

Float constants and float variables.

```
set  factor  =  -0.5
set  x_increment  =  1.66666
set  y_increment  =  x_increment
```

String constant and string variable.

```
set  message_pause  =  "Waiting for input."
set  message_1  =  message_2
```

Arrays

You must set each element of the array separately. You cannot set the entire array at once.

Array of integers with constants.

```
set  a[0]  =  32
set  a[1]  =  64
set  a[2]  =  128
set  a[3]  =  256
```

Array of integers with variables.

```
set  b[0]  =  a[0]
set  b[1]  =  a[1]
set  b[2]  =  a[2]
set  b[3]  =  a[3]
```

Array of strings with constants. Use double quotes around the string constant.

```
set  error_string[0]  =  "No errors"
set  error_string[1]  =  "Missing part 1"
set  error_string[2]  =  "Missing part 2"
set  error_string[3]  =  "Incomplete assembly"
```

Array of strings with variables.

```
set  error_string[0]  =  message[10]
set  error_string[1]  =  message[11]
set  error_string[2]  =  message[12]
set  error_string[3]  =  message[13]
```

String Limit

Any string that has been declared as a specific size can take only that number of characters. If you try to set a larger number of characters into the string variable, the extra characters are lost.

Example with constant:

```
new  $message[20]
set  message  =  "Re-set counter to 1000."
print  message
  = "Re-set counter to 10"
```

Example with variable:

```
new  $message1[25]
set  message1  =  "Re-set counter to 1000."
print  message1
  = "Re-set counter to 1000."
new  $message2[20]
set  message2  =  message1
print  message2
  = "Re-set counter to 10"
```

This problem can exist whether the variable is in the database and variable file as a result of the new command or as a result of ASH's v3 file generator reviewing a program with a declaration such as

```
teachable  string[20]  message2
```

You can display the size of a string with the list command.

Values From Other Variables

If you set a value using another variable, the current value is used and any further changes to one variable have no affect on the other variable. For example:

new alpha	create an integer
new beta	create an integer
set alpha = 5	set alpha to the value of the constant 5
set beta = alpha	set beta to current value of variable alpha which is 5
set alpha = 10	set alpha to the value of the constant 10
print alpha	display the value of alpha
= 10	the value when it was last set
print beta	display the value of beta
= 5	the value when it was last set

Values From New Variables

When you set a value using a second variable, that second variable must already be in the database. If you try to set a value and that second variable does not exist, the system asks if you want to create that variable. Even if you respond "yes" and the system makes that second variable, the system takes its unset value (zero) and uses that in your original set command. You must set the value of the second variable and then use that in setting the first variable. For example:

set y = z	try to set y to the value of z
Variable z not found.	Variable z does not exist
-- create it?	system prompts to create it
yes	respond with yes
	variable z is created
	variable z has not been set by user and is zero
	original command is executed: sets y to value of z (zero)
print z	display the value of z
= 0	the value when it was created
print y	display the value of y
= 0	the value when it was set to equal z
set z = 5	z is set to 5
set y = z	original command now works: y is set to 5
print z	display the value of z
= 5	the value when it was last set
print y	display the value of y
= 5	the value when it was last set

Using the set Command With Locations

In the same way that you can set a value for an int, float, or string variable, you can set the value for a location variable with the set command.

Variables

A location variable can be set with another location variable of the same type.

```
set  safe_point_9  =  safe_point_1
set  path_out[0]  =  path_in[0]
```

Constants

When you use the set command with constants, you must specify the exact value to be assigned to the variable.

With a cloc location variable, you must specify exact cartesian axis distances and rotational orientations.

A cloc is composed of five to eight floats for cartesian axis distances, rotational orientations, and extra axes. The application shell promotes integers to floats where necessary and ignores unneeded extra axis values. For example:

```
set  plate_xfrm  = {12.0,0.0,42.0,0.0,0.0,0.0,30.0,0.0}
set  stack_xfrm  = {20,15,5,0,-90,0,0,0}
```

You cannot move to the resulting cloc. You can only use it to modify a coordinate system or a location, such as specifying a base offset, a tool transform, a world shift (wshift), or a tool shift (tshift).

Normally location variables are modified with the here command.

Displaying Values of Variables print or ?

After loading variables into the database from a file or after teaching a variable with the here or set command, you can display the value of a variable with the print command. You must specify the variable.

```
print  number_of_cycles
= 10
```

The ? (question mark) is an alias for the print command.

```
?  number_of_cycles
= 10
```

If you specify an array, ASH displays all elements of the array.

```
print  difference
[0,0]  =  33.3333
[0,1]  =  16.6667
[1,0]  =  25.0000
[1,1]  =  12.5000
```

Deleting Variables
erase, eraseall

To erase a variable and its value from the database, use the erase command. You must specify the variable to erase. You can erase an array, but not part of an array.

```
erase   number_of_loops
erase   pick_1
erase   safe
erase   dispense
```

To erase all variables and their values from the database, use the eraseall command.

```
eraseall
```

With both the erase command and the eraseall command, ASH prompts you to confirm the action.

Merging Data

When you work in the database, you are modifying data to save to a variable file. While working in the database, you may want to include data from an existing variable file. For example, you may have taught several locations and saved them to a file, and now want these same locations in the database to include them in a new variable file.

To get data from an existing file, use the merge command.

When you merge data, you usually use two other commands. You often erase unneeded data from the database before or after merging data into the database. Also, you often save data from the database to variable files as well as merge data from files to the database.

Preparing the Database
erase & eraseall

You may work on a series of variable files with very different content. If this is the case, you may want to erase all data in the database with the eraseall command. You can then load the contents of the next variable file with the merge command. Since there are no variables in the database to conflict, the merge command loads the entire contents of the file into the database without prompting for confirmation.

Alternatively, you may be working with data in the database and, after saving to one file, want some, but not all, of that data for the second file. If this is the case, you can erase unwanted data from the database with the erase command.

Adding Teachables From Another File
merge

You can load data from any variable file into the database with the merge command.

As it prepares to load data, the merge command checks for name conflicts where a variable in the file has the same name as a variable in the database. If a conflict occurs, ASH asks you whether you want to accept or reject the value from the file into the database. If you accept, the file value over-writes the existing database value. If you reject, the existing database value remains. You can do this for each variable, one at a time, or for all variables, all at once. Each time ASH asks, you have four options.

	single variable	every variable
accept	yes	all
reject	no	ignore

At the first conflict, a message with the variable name is displayed, such as:

`Variable` *`name`*`: accept new value (yes/no/all/ignore)?`

If you respond for that single variable (with yes or no), ASH accepts or rejects that variable and then displays a similar message for the next conflicting variable.

If you respond for every conflicting variable (with all or ignore), ASH accepts or rejects the current specific variable and all remaining conflicting variables.

After you have merged some or all of the variables from a second file into the database, work with those variables and then save them to a file.

Cleaning Up the Database

erase

When loading data from a file into the database, the merge command prompts for a response (yes, no, all, ignore) only if there is a name conflict. Any variable with a unique name is automatically loaded into the database from the file. You may get variables in the database that you do not need. You can erase these from the database with the erase command.

Saving Data

When you set values to variables, teach locations, or make other modifications, you make changes in the database. Data is saved from the database to the variable file. When you run a program, the system uses the data in the variable file to initialize the variables in the program.

Whether saving is done automatically or by the save command, the new data over-writes the existing .v3 file. To keep an old .v3 file, use the system shell's commands for copying, renaming, or moving files.

Automatic Saving

Whenever you make a change in the database, ASH automatically saves to the variable file. You do not need to use the save command.

The application shell saves to the default variable file, set when ASH is started.

If you do not specify a variable file when you start ASH, the default variable file is the file with the same name as the application. For example, if you started the application “test” without specifying a file, the default variable file is “test.v3”. The automatic save feature does not save to any other file.

If you specify a variable file when you start ASH, the default variable file is the file that you specified. For example, if you started the application “test” with the file “alpha.v3”, the automatic save feature saves the data to “alpha.v3”.

To save to a different file you must use the save command.

Saving to a Specific File
save

You can save to any file with the save command.

If you do not specify a file, ASH saves to the default variable file.

You can specify any file, existing or new, and can specify the path, absolute or relative, to the file.

```
save   alpha.v3
save   \app\test\alpha.v3
save   ..\test\alpha.v3
```

If you do not specify the .v3 extension, ASH adds it to the file name.

```
save   alpha
save   \app\test\alpha
save   ..\test\alpha
```

Once the data has been saved to a variable file, that variable file can be used with a program file.

Saving to Multiple Files

When you save to a file, the data is copied into the file, but the data is still in the database until you exit from the application or power down. You can further modify the data in the database and save that to another file. In this way, you can build several similar variable files.

Configuring the Arm

Before moving the arm to teach locations, several settings may need to be made.

The commands listed below are described in detail in the next chapter, *Application Shell Commands*.

Homing the Arm
home

home	home the axes

Getting Arm Data
w0, wcmd, w1, w2, wact, w3, w4, wend, w5

w0	wcmd	display commanded position
w1		continually display actual position
w2	wact	display actual position
w3		continually display commanded position
w4	wend	display the next motion end-point
w5		display velocity command

Preparing to Move
base, tool, griptype_set, speed

These commands have equivalents in RAPL-3. If you set one of these with ASH and do not set it in the program, when you run the program, the ASH setting is maintained. If you set one of these with ASH and then set it in the program, when the program is run, the ASH setting is over-written.

You need to set the tool transform before teaching locations in ASH.

base		base offset, re-define the world coordinate system
tool	tool_set	tool transform, re-define the tool coordinate system
griptype_set		displays/sets the type of gripper used (air, servo, none)
speed		displays/sets the current speed setting

Moving the Arm

The application shell with its database modifies teachable variables. Most importantly, it modifies locations. Before a location can be taught with the here command, the arm must be moved to a position.

You can move the arm with the teach pendant or with ASH. The application shell contains the most common robot motion commands.

Motion using ASH can be dangerous. *The arm tries to complete the motion command as specified. If the increment is too large, a collision could result, damaging the arm, work pieces, or other equipment. Use small increments and slow speed. Be prepared to hit an e-stop. Use the teach pendant where releasing a motion key stops motion.*

The motion commands are described in detail in the next chapter, *Application Shell Commands*.

Transferring Control

The application shell and the teach pendant are similar devices. One is a command-line interpreter using keyboard and monitor, and the other is a hand-held device using a keypad and LCD display. Through either device, you can create variables, set values to variables, teach locations, and move the arm.

Additionally, the program can move the arm.

Unsafe operation would result if more than one of these had control of the arm. The system is designed for only one to have control of the arm at one time.

Opening an Application

When the application is opened from ASH — by starting ASH and specifying an application by name — ASH is the parent process. When the pendant is started from ASH, it does so as a child process of ASH.

You can check the status of the pendant process with the system shell's ps command. The pendant is labeled "stpv3".

Securing Control at the Pendant

With both ASH and the pendant running, either can be used to move the arm or perform other operations such as modifying variables. If you use the pendant and successfully move the arm, control is secured at the pendant. Control remains there until you explicitly transfer it to ASH or run a program.

Transferring from ASH to the Pendant pendant

You can start the pendant process and transfer control from ASH to it with the pendant command.

application_name>pendant

Explicitly Transferring Control to the Pendant

If the pendant process is already running, the pendant command transfers control from ASH to the pendant.

Transferring from the Pendant to ASH

If the pendant has control, it must explicitly give control to another process. Another process, such as ASH, cannot take control.

At the teach pendant keypad, press Shift + ESC to transfer control to ASH. The transfer function can also be reached by repeatedly pressing ESC to move up the hierarchy of screens to the Terminate Pendant screen.

Securing Control by a Program

When a program is run, control is readied to be transferred to the program. If the program successfully moves the arm, control is secured by the program. Control remains there until the program explicitly transfers control with the ctl_give() command or releases it with ctl_rel() command.

Having the program use the terminal or the pendant for taking input from the keyboard or keypad, is just using that device for standard input/output and does not affect point of control of the robot.

Understanding Control

For more on points of control and transferring control, see the chapter on P*oints of Control.*

Running an Application

You can run an application from ASH. You can run the application that you have open, or run any other application.

Running the Default Program run

You can run the application with the run command.

application_name>run

The run command executes the program file with the same name as the application using the variable file with the same name as the application name.

For example, if you are in the application “test”, the run command executes “test” with “test.v3”.

Running Any Program *filename*

Normally, with one program file and one variable file under one application, the run command is sufficient.

If needed, you can run any application by specifying the program. You can optionally specify the variable file.

Use this method if you have multiple program files or multiple variable files, or want to test run a file in another directory without exiting out of the current application shell.

```
test>test1
test>test1:test1.v3
test>test1:alpha.v3
dispense>..\test\test1:..\test\alpha.v3
```

Running from the System Shell

You can run any application from the system shell.

It can be more efficient to run your application without ASH. Your application needs memory to run and ASH takes up some memory. If you are finished developing your application, you can exit from ASH (which frees the memory) and run your application from the system shell prompt.

To run from the system shell, you must use the filename, either specifying the path to the file or first changing the working directory to the application’s directory. If you specify only the program file, the system uses the variable file with the same name as the program file. You can specify the variable file with the : (full colon) as you can in ASH.

Running in the Background
&

You can run an application in the background. This gives you back the prompt to enter other commands while the program is executing. Use an & (ampersand) after the executable.

```
test>run &
test>test1 &
test>test1:test1.v3 &
```

Once the program successfully moves the robot, the program has control of robot motion and you cannot use any robot motion commands of ASH.

If the program is run from ASH, the program is a child process of ASH. If you exit ASH, which terminates the ASH process, all child processes of ASH, including the program, are sent a SIGHUP signal. If a child process does not mask or handle SIGHUP, then it is terminated by CROS.

CHAPTER 3

Application Shell Commands

This chapter describes the commands that you can use through the application shell (ASH). There are four sections:

- categories of ASH commands
- details of individual ASH commands, listed alphabetically
- command line features
- system shell commands available in ASH

Almost all robot commands (motion, gripper, calibration, coordinate systems, etc.) call RAPL-3 commands from the libraries. Further details of these commands are in the *RAPL-3 Language Reference Guide*.

Categories of Commands

Details of the commands are given in the alphabetical listing.

Start Up and Exit	
ash	start the application shell
exit	exit the current application shell
ver	display the version of the application shell
Variables, Values, and Locations	
erase	erase variable and value
eraseall	erase all variables and values
here	store current coordinates into location variable
list	list variable types and names and values for int, float and string types
merge	merge variable records from a .v3 file
new	create a new variable
print, ?	display the value of a variable
refresh	refresh .v3 file and database from program
save	save variables to a .v3 file
set, !	set (assign) a value to a variable
Coordinate Systems	
base, base_set	base offset, re-define the world coordinate system
tool, tool_set	tool transform, re-define the tool coordinate system
tshift	modify location in tool coordinate system
wshift	modify location in world coordinate system
Calibration, Homing, and Status	
ampstat	obtain the status of F3 amplifiers
arm_status, armstat	display robot arm status information
armpower, arm	enable/disable arm power
calrdy	move to calrdy position
clrerror	clear motion errors on F-series robots
home	home the axes
robotver	display robot version
w0, wcmd	display commanded position
w1	continually display actual position
w2, wact	display actual position
w3	continually display commanded position
w4, wend	display end-point position
w5	display position error
Arm Configuration	

accel	display/set accelerations
cfg_save	save robot configuration
linacc_set, linacc	display/set linear acceleration
linspd_set, linspd	display/set linear speed
rotacc_set, rotacc	display/set the maximum rotational acceleration
rotspd_set, rotspd	display/set the maximum rotational speed
servoerr	display the servo error detection parameters
Machine	
use	display/select robot to use (if more than one)
Motion	
align	align to world axis
appro, appros	move to approach position
calrdy	move to calrdy position
depart, departs	move to depart position
finish	finish motion
halt	halt (stop) motion
joint	rotate a joint
limp	limp joint(s)
lock	lock joint(s)
motor	rotate a motor
move, moves	move to a location
nolimp	unlimp joint(s)
online	set online mode
pitch, pitchs	pitch tool centre point (in tool coordinate system)
ready	move to ready position
roll, rolls	roll tool centre point (in tool coordinate system)
speed	set speed
stance, stance_set	place arm in pose
tx, txs	jog along the tool X axis
ty, tys	jog along the tool Y axis
tz, tzs	jog along the tool Z axis
unlock	unlock joint(s)
wx, wxs	jog along the world X axis
wy, wys	jog along the world Y axis
wz, wzs	jog along the world Z axis
xrot, xrots	rotate around the world X axis
yaw, yaws	yaw the tool centre point (in tool coordinate system)
yrot, yrots	rotate around the world Y axis

zrot, zrots	rotate around the world Z axis
Gripper	
gripdist_set, grip	move servo gripper fingers to a distance
grip_close, gclose, gc	close the gripper
grip_open, gopen, go	open the gripper
griptype_set, gtype	display/set the type of gripper used (air, servo, none)
wgrip	display servo gripper finger distance
Input/Output	
input	display state of channel
output	set state of channel
Pendant	
pendant	start the pendant and transfer control to pendant
Application Execution	
run	run (execute) a program
file_name	run (execute) a program
Help	
help	display help on commands

Detailed Descriptions

These are detailed descriptions of all ASH commands listed alphabetically.

file_name

Description

Runs the specified program file with the specified variable file.

Starting the application shell changes the working directory to the directory with the application's name. If you want to run a file in another directory, you can either change the working directory (with the cd command) and enter the filename, or enter the full path to that file. You can use either the relative path from the working directory or the absolute path.

If you have multiple files in the application directory, specify by filename to run a file.

If you run from the system shell, you must specify by filename. If you are finished developing your application, you can exit out of ASH and run your program from the system shell. Exiting from ASH frees space in memory that could be used by larger programs.

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. The separator between program file and variable file is : (full colon).

xname	program file name (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.

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

See Also
run runs the program with the same name as the application
& (in Features) places the program in the background

Category
Execution

?

Displays the value of the specified variable in the current database.

See
print

!

Sets (assigns) a value to a variable.

See
set

accel

Description
Displays or sets the current **accel**eration settings of the robot. The units are degrees/sec² for rotation joints and inches or mm/sec² for linear joints such as tracks.

Warning
Setting the accelerations to large values can cause mechanical damage to the robot.

Display

Syntax
`accel`

Parameters
none

Example
`accel`

Result
```
Current Accelerations are:
J1=499.9999, J2=499.9999, J3=499.9999, J4=2250, J5=499.9999...
```

Set

Syntax
`accel` *`j1, j2, j3, j4, j5, j6`*

Parameters
Six required parameters.

j1	joint 1 acceleration
. . .	. . .
j6	joint 6 acceleration

Example
`accel 0.3, 0.2, 0.2, 0.2, 0.2, 0.2`

Sets acceleration values of joint 1 to 0.3 and joints 2 through 6 to 0.2.

RAPL-3 Language
accels_set(), accels_get().

RAPL-II
Similar to @@ACCEL

See Also **linacc_set** displays/sets linear acceleration
rotacc_set displays/sets rotational acceleration

Category Arm Configuration

align

Description **Align**s the tool parallel to the nearest or a specific world axis.

Syntax `align` *`axis`*

Parameter One required parameter.
axis the axis to align to, one of:

n	the nearest world axis
x	the world X axis
-x	the world -X axis
y	the world Y axis
-y	the world -Y axis
z	the world Z axis
-z	the world -Z axis

Examples
```
align x
align -z
```

RAPL-3 Language Same as align().

RAPL-II Similar to ALIGN.

Category Motion Arm Configuration

ampstat

Display F3 amplifier status information.

See **amp_status**

amp_status

Alias **ampstat**

Description Display F3 amplifier status information. Displays nothing for A Series robots.

Syntax `amp_status`

Parameter There are no parameters.

Examples `amp_status`

Result
```
Waist module:
      DSP code version 16
      Intel I196 code version 292
      Temperature is : 25.8 degrees
```

```
Wrist module:
      DSP code version 16
      Intel I196 code version 292
      Temperature is : 27.3 degrees

Track module:
      DSP code version 18
      Intel I196 code version 292
      Temperature is : 27.3 degrees

Arm Power is OFF

Amplifier Status
1......OK
2......OK
3......OK
4......OK
5......OK
6......OK
7......OK
```

RAPL-3 Language — No direct equivalent. The commands robotispowered() and robot_servo_stat() provide arm power and amplifier status information.

RAPL-II — No equivalent.

See Also — **arm_status**

Category — Calibration, Homing, and Status

appro, appros

Description — Moves the tool point to an **appro**ach position. An approach position is a position near a location, but a specified distance away from the location along the "approach/depart" tool axis. Used as a preliminary position before moving carefully to the exact location.

The motion from the current position is joint interpolated with appro and cartesian interpolated, **s**traight line, with appro**s**.

Syntax

```
appro  location_name , distance
appros location_name , distance
```

Parameters — Two required parameters.

Location_name	the name of the cartesian location to approach
Distance	the distance away from the location, along the "approach/depart" tool axis, in current units (mm or in)

Examples

```
appro  place_1,  2
appros place_2,  1
```

RAPL-3 Language — Same as appro() and appros().

RAPL-II — Same as APPRO, without and with the s parameter.

See Also

depart	moves away from the current position
departs	moves away from the current position in a straight line
tool	re-defines the tool axis

Category — Motion

arm

Enables or disables robot arm power.

See **armpower**

arm_status

Alias **armstat**

Description Displays robot arm status information.

Syntax `arm_status`

Parameter There are no parameters.

Examples `arm_status`

Result

```
Robot arm is of type F3
Arm power is OFF     Robot is calibrated
Units are Metric     Online is OFF
Physical stance: forward up noflip

Current Tool Transform is:
tx=0, ty=0, tz=0, yaw=0, pitch=0, roll=0

Current Base Transform is:
wx=0, wy=0, wz=0, xrot=0, yrot=0, zrot=0

Axis    locked   done   limped
 1        N       Y       Y
 2        N       Y       Y
 3        N       Y       Y
 4        N       Y       Y
 5        N       Y       Y
 6        N       Y       Y
 7        N       Y       Y
```

RAPL-3 Language No direct equivalent. The commands server_info(), robot_info(), units_get() and axis_status() provide the same information.

RAPL-II No equivalent.

See Also **ampstat**

Category Calibration, Homing, and Status

armpower

Alias

arm

Description

Enables or disables robot **arm power**.

The disable command immediately shuts off arm power, if currently on. The arm power switch on the front panel has no effect.

The enable command allows the arm power to be turned on. The arm power switch must be manually operated to turn on arm power.

Syntax

`armpower` *`onoff`*

Parameters

One required parameter.

onoff the flag, one of:

0	disable
1	enable

Examples

```
armpower 0
armpower 1
arm 0
arm 1
```

RAPL-3 Language

Same as armpower().

RAPL-II

Same as ARM.

Category

Calibration, Homing, and Status

---

armstat

Displays robot arm status information.

See

arm_status

---

base

Alias

base_set

Description

Displays or sets the robot **base** offset; re-definition of the world coordinate system.

If a base offset is set, then the cfg_save command must be used in order to save it as part of the robot power on configuration. Do not run cfg_save if the base offset being set is not the one that you want the robot to power on with.

Display

Syntax

`base`

Parameters

There are no parameters.

Example

`base`

Result

```
Current Base Transform is:
wx=0, wy=0, wz=0, zrot=0, yrot=0, xrot=0
```

Set

Syntax `base x, y, z, zrot, yrot, xrot`

Parameters Six required parameters.

x	distance along the world X axis
y	distance along the world Y axis
z	distance along the world Z axis
zrot	rotation about the world Z axis
yrot	rotation about the world Y axis
xrot	rotation about the world X axis

Example
```
base  1, 2, 3, 45, 10, 15
```

RAPL-3 Language Same as base_get(), base_set().

RAPL-II Similar to BASE.

See Also **cfg_save** save robot configuration
tool displays/sets a tool transform

Category Coordinate Systems

base_set

Sets a base offset.

See **base**

calrdy

Description Moves the robot to the **cal**ibration **r**ea**dy** position.

For an F3 or A465 robot, the calrdy position is straight up, the same as the ready position with joint 3 rotated an addition 90°.

For an A255 robot, the calrdy position is straight out with the arm links from shoulder (joint 2) to wrist (joint 5) aligned to the world X axis.

Syntax `calrdy`

Parameters There are no parameters.

Example
```
calrdy
```

RAPL-3 Language Same as calrdy().

RAPL-II Same as CALRDY.

See Also **ready** moves to the ready position.

Category Motion

cfg_save

Description

Saves the current robot configuration information. This consists of:

- Whether the robot is on a track or not. This can be set in RAPL-3 with track_spec_set().
- The total number of axes in the system. Maximum of 8. This can be set in a RAPL-3 program with the axes_set() command.
- The tool offset. This can be set in the application shell by using the tool command or a RAPL-3 program using the tool_set() command.
- The base offset. This can be set in the application shell by using the base_set command or a RAPL-3 program using the base_set() command.
- If a track is present, the positive and negative travel lengths from the zero position. This can be set in RAPL-3 with the jointlim_set() command.
- The gripper type: air, servo or none. This can be set in the application shell by using the griptype_set command or in a RAPL-3 program with the griptype_set() command.
- The engineering units to be used: Metric (mm) or English (inches). This can be set in the application shell by using the /diag/setup command or in a RAPL-3 program with the units_set() command.

With the exception of the base and tool offset all of the above parameters can also be set using the /diag/configur utility in the system shell.

Syntax

```
cfg_save
```

Parameters

There are no parameters.

Example

```
cfg_save()
```

RAPL-3 Language

No direct equivalent. The combination of track_spec_set(), axes_set(), tool_set(), base_set(), jointlim_set(), griptype_set() and units_set() provides equivalent functionality.

RAPL-II

@@SETUP, @TRACK and @XLIMITS provided some of the same functionality.

See Also

tool_set tool transform, re-define the tool coordinate system
gtype displays/sets gripper type used (air, servo, none)
base_set base offset, re-define the world coordinate system

Category

Arm Configuration

clrerror

Description

Clears persistent error bits on the F3 amplifier. This includes runaways, collisions, overspeeds, and encoder faults. After an error of this type, the clear_error() command must be invoked before arm power can be re-engaged.

Note: This is command only works with the F-series arms.

Syntax

```
clrerror
```

Parameters

There are no parameters.

Example `clrerror()`

Result `Error state has been cleared.`

RAPL-3 Language Same as clear_error().

RAPL-II No equivalent.

Category Calibration, Homing, and Status

depart, departs

Description **Depart**s from a position. Moves the tool point from the current position to a position that is a specified distance away, along the "approach/depart" tool axis. Often used to slowly and carefully move the tool away from an exact location before moving quickly elsewhere.

Can be used from any position, not only locations.

A positive value moves away from the current position, in the negative direction of the tool axis. A negative value moves in the opposite direction, forward, past the position.

The motion from the current position is joint interpolated with depart and cartesian interpolated, **s**traight line, with depart**s**.

Syntax
```
depart distance
departs distance
```

Parameters Takes one required parameter.

distance the distance from the position, in current units

Examples
```
depart   100
departs  3
```

RAPL-3 Same as depart() and departs().

RAPL-II Same as DEPART, without and with the s parameter.

See Also **appro/appros** moves to an appro position
tool re-defines the tool axes

Category Motion

erase

Description **Erase**s (deletes) the specified variable and its value from the current database.

Syntax `erase variable_name`

Parameters Takes one required parameter:

variable_name the name of the variable to erase

Examples
```
erase  a
erase  place_1
```

Confirmation `Erase variable 'name' and its value?`

Response Takes one response:

	y or **yes**	erase the variable and value
	n or **no**	do not erase the variable and value

Any other response displays a prompt for a correct response.

See Also **eraseall** erases all variables and values from database

Category Variables, Values, and Locations

eraseall

Description **Erase**s (deletes) **all** variables and their values from the current database.

Syntax `eraseall`

Parameters There are no parameters.

Example `eraseall`

Confirmation `Erase ALL teachable variable values: are you sure?`

Response Takes one response:

y or **yes**		Erase all variables and values
n or **no**		do not erase any variables and value

Any other response displays a prompt for a correct response.

See Also **erase** erases a single variable and value from database

Category Variables, Values, and Locations

exit

Alias **quit**

Description **Exit**s the current application shell.

Returns to the parent process of this application shell, normally a system shell.

The application shell will not allow you to exit if the pendant has point of control. At the teach pendant keypad, press Shift + ESC to transfer control to ASH. The transfer function can also be reached by repeatedly pressing ESC to move up the hierarchy of screens to the Terminate Pendant screen.

Warning Exiting from ASH sends a SIGHUP signal to all child processes of ASH, including any programs started from ASH. A child process that does not mask or handle a SIGHUP will be terminated by CROS.

Syntax `exit`

Parameters There are no parameters.

See Also

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

Category Start Up and Exit

finish

Description **Finish**es arm motion before further program execution. **Note:** finish does not wait for gripper motions to finish.

Syntax `finish`

Parameters There are no parameters.

Example `finish`

RAPL-3 Language Same as finish().

RAPL-II Same as FINISH.

See Also **online** sets online mode on or off

Category Motion

gc

Closes the gripper.

See **grip_close**

gclose

Closes the gripper.

See **grip_close**

go

Opens the gripper.

See **grip_open**

gopen

Opens the gripper.

See **grip_open**

grip

Sets the gripper distance.

See **gripdist_set**

gripdist_set

Alias

grip

Description

Moves the servo-**grip**per fingers to a specified **dist**ance apart from each other. Fingers move in an opening or closing direction depending on the starting position. Distance is in currently set units: metric or English.

Gripper type must be set to 2 (servo) for the gripdist_set command to function.


Warning

This command operates at 100% force. Do not use this command to hold an object. Use grip_close or grip_open which operates with the servo loop.

Syntax

```
gripdist_set distance
grip distance
```

Parameters

One required parameter.

distance	the distance between fingers in current units: a float

Examples

```
gripdist_set  1.0
grip  10.5
```

RAPL-3 Language

Same as gripdist_set().

RAPL-II

Same as GRIP.

See Also

grip_close	closes the gripper
grip_open	opens the gripper
griptype_set	displays/sets the type of gripper used (air, servo, none)

Category

Gripper

grip_close

Aliases

gclose, gc

Description

Closes the **grip**per fingers with an optionally specified servo force.

Syntax

```
grip_close [force]
gclose [force]
gc [force]
```

Parameters

One optional parameter.

force	the percentage of force (servo-gripper only)

If no parameter is given, the last force setting is used (servo-gripper only).

Examples

```
grip_close  60
grip_close
gclose  20
gclose
gc 70
gc
```

RAPL-3

Same as grip_close().

RAPL-II Same as CLOSE.

See Also
grip_open opens the gripper fingers: opposite of grip_close
gripdist_set moves the fingers to a specified separation distance
griptype_set displays/sets the type of gripper used (air, servo, none)

Category Gripper

---

grip_open

Aliases **gopen, go**

Description **Open**s the **grip**per fingers with an optionally specified servo force.

Syntax
```
grip_open [force]
gopen [force]
go [force]
```

Parameters One optional parameter.

force the percentage of force (servo-gripper only)

If no parameter is given, the last force setting is used (servo-gripper only).

Examples
```
grip_open  25
grip_open
gopen  10
gopen
go 75
go
```

RAPL-3 Language Same as grip_open().

RAPL-II Same as OPEN.

See Also
grip_close closes the gripper fingers: opposite of grip_open
gripdist_set moves the fingers to a specified separation distance
griptype_set displays/sets the type of gripper used (air, servo, none)

Category Gripper

---

griptype_set

Alias **gtype**

Description Displays or **set**s the **grip**per **type** attached to the robot.

Note that the gripper type must be set to 2 (servo) for the **gripdist_set** command to work.

Syntax
```
griptype_set [type]
```

Parameters There is one optional parameter:

type The type of gripper the robot has attached. Valid values are "none", "air" or "servo" (equivalent to codes 0, 1 or 2) If this argument is omitted, then the gtype command displays what the gripper is currently set to.

Example1
```
griptype_set
```

Possible Responses
```
Gripper type set to 1--AIR
Gripper type set to 2--SERVO
Gripper type not set
```

Example2
```
griptype_set servo
```

Result
The robot gripper type is set to servo. Note: for this setting to persist when the controller is rebooted, the **cfg_save** command must be used to record the setting.

RAPL-3 Language
Same as **griptype_set()** and **griptype_get()**

RAPL-II
Same as @@SETUP: response to gripper type query

See Also
wgrip displays the finger separation distance of a servo gripper
cfg_save save current robot configuration

Category
Gripper

---

gtype

Displays or sets the gripper type.

See
griptype_set

---

help

Description
Displays **help** on application shell commands. Displays: command name, parameters, brief description.

Although many system shell commands are accessible from the application shell, help on system shell commands is available only from the system shell.

Syntax
`help [command_name]`

Parameters
One optional parameter:

command_name the command for which you want help

No parameter gives a list of all application shell commands.

Examples
```
help
help list
help move
```

See Also
help (in the system shell) displays help on system commands

Category
Help

---

here

Description
Stores the current arm coordinates into a specified location variable. Used to teach locations. This location is **here** at these coordinates.

If the location variable does not exist, makes a new location variable. The type of location variable (cloc or ploc) is specified by a type prefix. The default type is cloc, if no prefix is provided.

Also, the here command displays the coordinates of the current position.

Syntax `here [ [type_prefix]location_name ]`

Parameters No parameter, displays the current position.
One optional parameter. The parameter has an optional prefix.

location_name the name of the location

type_prefix the prefix indicating data type

The location name follows the rules for RAPL-3 identifiers.

- begins with a letter
- one or more letters, digits, or _ (underscore) characters
- any combination of uppercase (ABCDE) or lowercase (abcde)

The type prefix indicates one of the two location data types.

Type Prefix Character		Location Data Type	
Example	Description		
_	underscore	cloc	cartesian location
#	number sign	ploc	precision location

Examples
```
here _point2
here #dispense9
here thisloc                ;; defaults to cartesian
```

Example `here`

Result
```
NAME      X          Y          Z         ZROT       YROT       XROT
        TRACKX     TRACKY
WORLD   (mm)       (mm)       (mm)      (deg)      (deg)      (deg)

      0.000      0.000      0.000     0.000      0.000      0.000
      0.000      0.000
```

RAPL-3 Language Same as here().

RAPL-II Same as HERE.

See Also **new** makes a new variable
wact displays robot position

Category Variables, Values, and Locations

home

Description **Home**s all axes or specified axes.

Syntax `home [axis [, axis [, axis . . .]]]`

Parameters A list of optional parameters

axis which axis to home

If no parameters are given, homes all axes.

Examples
```
home
home 7
home 2,3,4
```

RAPL-3 Language Similar to home().

RAPL-II Same as HOME.

See Also **cal** calibrates axes

Category Calibration, Homing, and Status

input

Description Examines the state of a parallel I/O (**input**/output) channel. Displays the state or stores the state in an integer variable.

Syntax `input` *`channel`* `[,` *`variable`*`]`

Parameters One required and one optional parameter.

channel the number of the parallel I/O line to check, 1 . . . 16

variable the variable to store the result

If no variable is used, the result is displayed instead of stored in a variable.

Examples
```
input 2
input 4, x
```

Example & Result
```
input 2
input 2 = 0
```

RAPL-3 Language Same as input().

RAPL-II Same as IFSIG.

See Also **output** sets the state of an output channel

Category Input/Output

joint

Description Rotates a rotational **joint** (of an articulated arm) by a specified number of degrees, or moves a linear **joint** (of a track or gantry) by a specified number of current units (metric or English).

Syntax `joint` *`axis, distance`*

Parameters Two required parameters.

axis the axis being moved: an int

distance the distance of travel: a float (positive or negative)
an integer is converted to a float

Examples
```
joint  1   22.5
joint  2  +30
joint  3  -15.0
joint  7  200
```

RAPL-3 Language Same as joint().

RAPL-II Same as JOINT.

See Also **motor** rotates a motor by specified encoder pulses

Category Motion

limp

Description **Limp**s one joint, more than one joint, or all joints.

Warning Limping releases the joint and the link can fall due to gravity. Carelessly limping axes, especially joints 2 (shoulder) and 3 (elbow), can cause a fall which can cause damage.

F3 joints will move little or none, although starting from a straight-out position, joints 2 and 3 will move slowly. You can safely limp from the calrdy position.

A465 and A255 joints will fall quickly. Support the arm. It is not safe to limp the arm from any position without adequate support.

Syntax `limp [axis] [, axis] ...`

Parameters One or more optional parameter. If no parameter is given then all axes are limped.

axis the axis to be limped

If no parameter is given then all axes are limped.

Examples
```
limp
limp  1
limp  4,  5,  6
limp  7
```

RAPL-3 Language Same as limp().

RAPL-II Same as LIMP.

See Also **nolimp** unlimps joint(s)
calrdy moves to the calibration ready (zero) position

Category Motion

linacc

Displays or sets the linear acceleration.

See **linacc_set**

linacc_set

Alias **linacc**

Description Displays the current value of the robot's linear acceleration or sets it to the value specified.

Display

Syntax `linacc_set`

Parameters none

Example `linacc_set`

Result `Current Linear Acceleration is 100.000`

Set

Syntax `linacc_set` *`acc`*

Parameter Takes one parameter.

`acc`	the acceleration in the current engineering unit system (English or Metric); a float

Examples
```
linacc_set  95.5
linacc_set  120
```

RAPL-3 Language Same as linacc_set(), linacc_get().

RAPL-II Same as @CLINACC

See Also

accel	displays/sets acceleration
linspd_set	displays/sets linear speed
speed	displays/sets speed
rotacc_set	displays/sets rotational acceleration

Category Arm Configuration

linspd

Displays or sets the linear speed.

See **linspd_set**

linspd_set

Alias **linspd**

Description Displays the current value of the robot's linear speed or sets it to the value specified.

Display

Syntax `linspd_set`

Parameters none

Example `linspd_set`

Result `Current Linear Speed is 10.0000`

Set

Syntax `linspd_set` *speed*

Parameter Takes one parameter.

speed the speed in the current engineering unit system (English or Metric); a float

Examples
```
linspd_set  25.5
linspd_set  15
```

RAPL-3 Language Same as linspd_set(), linspd_get().

RAPL-II Same as @CLINSPD

See Also
accel displays/sets acceleration
linacc_set displays/sets linear acceleration
speed displays/sets speed
rotspd_set displays/sets rotational acceleration

Category Arm Configuration

list

Description **List**s variables in the current database. Lists: data type, whether taught or not, name and value for simple types.

Syntax `list [`*type*`]`

Parameters One optional parameter.

type the data type to list only variables of that type, any one of:

`int`	integer
`float`	floating point number
`string`	string
`cloc`	cartesian location
`ploc`	precision location
`gloc`	generic location

Examples
```
list
list  cloc
list  int
```

Example & Result
```
list
Variables:  (* indicates not yet taught)
  int                 number_of_loops        = 10
  int                 counter                = 1
  ploc              * pick_1
  cloc              * place_1
```

See Also
erase erases a variable and value
eraseall erases all variables and values
new makes a new variable
print prints the value of a variable

set set a value to a variable
ls (system shell) lists directory contents

Category Variables, Values, and Locations

lock

Description **Lock**s one or more joints.

Caution Motion with move or moves can cause unexpected arm motion.

Syntax `lock a`*`xis [, axis] ...`*

Parameters One required and other optional parameters.

`axis` the axis to be locked

Examples
```
lock 7
lock 2, 3, 4, 5
```

RAPL-3 Language Same as lock().

RAPL-II Same as LOCK.

See Also **unlock** unlocks joint(s)

Category Motion

merge

Description **Merge**s variables and their values from a file into the current database.

A conflict occurs when a variable in the file has the same name as a variable in the database. If a conflict occurs, the system prompts to accept (copy) the value from the file into the database, over-writing the existing database value.

Syntax `merge` *`file_name`*

Parameter One required parameter.

`file_name` the name of the file to merge from

Responses In cases of conflict, the message displays:

`Variable` *`name`*`: accept new value (yes/no/all/ignore)?`

Responses are:

Response		Description
Letter	Word	
`y`	`yes`	Accept the file value for this variable. Lose the existing database value.
`n`	`no`	Reject the file value for this variable. Keep the existing database value.
`a`	`all`	Accept the file values for all variables. Lose any conflicting existing database values.

I	ignore	Reject the file values for all variables. Keep the existing database values.

See Also **save** saves variables and values from database to a file

Category Variables, Values, and Locations

motor

Description Rotates a **motor** by a specified number of encoder pulses.

Syntax `motor` *`axis, pulses`*

Parameters Two required parameters.

`axis` the axis being rotated: an int

`pulses` the number of encoder pulses: an int (positive or negative)

Examples
```
motor  1,   4500
motor  2,  +1500
motor  3,  -2500
motor  7,   10500
```

RAPL-3 Language Same as motor() without third parameter of condition.

RAPL-II Same as MOTOR without third parameter of condition.

See Also **joint** rotates joint by value given in degrees

Category Motion

move, moves

Description **Move**s the tool tip to a specified location.

The motion from the current position is joint interpolated with move and cartesian interpolated, **s**traight line, with move**s**.

Syntax
```
move  location_name
moves location_name
```

Parameters One required parameter.

`location_name` the destination location

Examples
```
move  point1
moves place_2
```

RAPL-3 Language Same as move(), moves().

RAPL-II Same as MOVE without and with the s parameter.

See Also **appro/appros** moves to an approach position
depart/departs moves to a depart position
finish finishes current motion

Category Motion

new

Description

Creates a **new** variable in the current database. Similar to a declaration in a RAPL-3 program using an implicit declaration prefix.

Syntax

```
new type_prefix-
variable_name[dimension_size][dimension_size]...
```

Parameters

One required parameter which has two parts and optional dimension(s).

type_prefix	the prefix indicating data type
variable_name	the name of the variable
dimension_size	the size of a dimension for an array

The variable name follows the rules for RAPL-3 identifiers.

- begins with a letter
- one or more letters, digits, or _ (underscore) characters
- any combination of uppercase (ABCDE) or lowercase (abcde)

The type prefix indicates one of the five data types.

Type Prefix Character		Data Type	
Example	Description		
	none	int	integer
%	percent sign	float	floating point number
$	dollar sign	string[]	string of characters
_	underscore	cloc	cartesian location
#	number sign	ploc	precision location

Arrays are made by giving dimensions. The square brackets are necessary. The dimension size in the new command is a positive integer. The variable's index numbering begins at zero.

`new %calc[3]`	a one-dimensional array of floats calc[0], calc[1], and calc[2]
`new $message[20][5]`	a one-dimensional array of strings, each string able to hold 20 characters, message[0] to message [4]
`new #pallet[6][12]`	a two-dimensional array of plocs, pallet[0][0] to pallet[5][11]

The limits on dimensions of arrays are: one dimension of string and two dimension of int, float, cloc, and ploc.

Limits on Dimensions of Arrays	
Data Type	Limit
int	two dimensions

float	two dimensions
string[]	one dimension
cloc	two dimensions
ploc	two dimensions

Examples

```
new counter
new %difference
new $message[20]
new _safe
new #dispense
new #pallet[12][8]
```

See Also

erase	erases a variable and value
eraseall	erases all variables and values
list	lists variables and values
print	prints the value of a variable
set	set a value to a variable

Category Variables, Values, and Locations

nolimp

Description Unlimps one, more than one, or all joints.

Syntax `nolimp [axis [, axis]…]`

Parameter Zero or more optional parameters. If no parameter is given then all axes are unlimped.

axis	the axis to be unlimped

Examples

```
nolimp  1
nolimp  4,  5,  6
```

RAPL-3 Language Same as nolimp().

RAPL-II Same as NOLIMP.

See Also **limp** limps joint(s)

Category Motion

online

Description Sets the **online** mode to the specified value.

Syntax `online mode`

Parameter One required parameter.

mode the online mode to be set, one of:

0	off	space in queue for 1 motion command
1	on	space in queue for 8 motion command
2	wait	fill queue with motion commands

	3	proceed	begin execution of commands
	4	track	enable sensor tracking inputs
	5	notrack	disable sensor tracking inputs

Examples
```
online 1
```

RAPL-3 Language Same as online().

RAPL-II Same as ONLINE.

See Also **finish** finishes one motion

Category Motion

output

Description Sets the state of a parallel I/O (input/**output**) channel.

Syntax
```
output channel , state
```

Parameters Two required parameters.

channel	the number of the parallel I/O line to set, 1 . . . 16
state	the state of the output, one of:
	0 off
	1 on

Examples
```
output 2, 0
output 4, 1
```

RAPL-3 Language Same as output().

RAPL-II Same as OUTPUT with different parameters.

See Also **input** examines the state of an input channel

Category Input/Output

pendant

Description Starts and transfers control to the teach **pendant**. If the pendant software is already running, then only point of control is transferred.

Control is transferred back from the pendant by pressing SHIFT and ESC together on the pendant keyboard.

The application shell will not allow you to exit if the pendant has point of control. At the teach pendant keypad, press Shift + ESC to transfer control to ASH. The transfer function can also be reached by repeatedly pressing ESC to move up the hierarchy of screens to the Terminate Pendant screen.

Syntax
```
pendant
```

Parameters There are no parameters.

Example
```
pendant
```

Result
```
Starting pendant...
Transferring robot control to the pendant
```

See Also **exit**

Category Pendant

pitch, pitchs

Description **Pitch**es the tool centre point by a specified angle about the tool Y axis.

The motion from the current position is joint interpolated with pitch and cartesian interpolated, **s**traight line, with pitch**s**. With the pitchs command, the tool tip stays on the tool y axis, in the same place, while the tool rotates around the axis.

Caution The **pitchs** command should only be used with **online** mode on.

Syntax
```
pitch  angle
pitchs angle
```

Parameters One required parameter.

angle	the amount of rotation in degrees

Examples
```
pitch  22.5
pitchs 10
```

RAPL-3 Language Same as pitch() and pitchs().

RAPL-II No equivalent. In RAPL-3, pitch is a rotation in the tool frame of reference. In RAPL-II, PITCH was a rotation in the world frame of reference.

See Also

yaw, yaws	yaw the tool by a specified angle
roll, rolls	roll the tool by a specified angle
xrot, xrots	rotate the tool about the world X axis
yrot, yrots	rotate the tool about the world Y axis
zrot, zrots	rotate the tool about the world Z axis

Category Motion

print

Alias **?**

Description Displays (**print**s to screen) the value of a variable in the current database. For an array, displays the entire contents of the array.

Syntax
```
print  variable_name
?      variable_name
```

Parameters One required parameter:

variable_name	the name of the variable whose value is to be printed

Example
```
print  number_of_cycles
```
Result
```
= 10
```

See Also

erase	erases a variable and value
eraseall	erases all variables and values

list	lists variables and values
new	creates a new variable
set	sets a value to a variable

Category: Variables, Values, and Locations

quit

Quits the current application shell.

See: **exit**

ready

Description: Moves the arm to the **ready** position.

Syntax: `ready`

Parameter: There are no parameters.

Example: `ready`

RAPL-3 Language: Same as ready().

RAPL-II: Same as READY.

See Also:

calrdy	moves to the calrdy position

Category: Motion

refresh

Description: **Refresh**es the database from the program file. Used after a newer program file has been transferred to the controller.

The refresh command compares the age of the program file and the variable file. If the program file is newer (implying that the program has just been transferred to the controller), then ASH refreshes the variable file and adds any new teachable variables to the database.

This command is designed to work with a single program file in an application.

Syntax: `refresh`

Parameter: There are no parameters.

Example: `refresh`

See Also:

merge	merge in the contents of another v3 file
save	explicitly save the current database to another file

Category: Variables, Values, and Locations

robotver

Description — Displays the **robot ver**sion, the strings embedded in the robot kinematics code. This is useful in helping CRS to remotely diagnose problems.

Syntax
```
robotver
```

Parameters — There are no parameters.

Examples
```
robotver
```

Result
```
Version string: 'Rapl-3 Kin Core build 109 - A255 Kinematics
Model v2.0D, Jun 18 1998 11:02:43'
```

RAPL-3 Language — Same as verstring_get().

RAPL-II — Same as verstring_get().

See Also — **ver** displays version information about ASH

Category — Calibration, Homing, and Status

roll, rolls

Description — **Roll**s the tool by a specified angle about the tool "approach/depart" axis.

The motion from the current position is joint interpolated with roll and cartesian interpolated, **s**traight line, with roll**s**. With the rolls command, the tool tip stays on the axis, in the same place, while the tool rotates around the axis.

Caution — The **rolls** command should only be used with online mode on.

Syntax
```
roll  angle
rolls angle
```

Parameters — One required parameter.

angle	the amount of rotation in degrees

Examples
```
roll  45
rolls 22.5
```

RAPL-3 Language — Same as roll() and rolls().

RAPL-II — No equivalent. In RAPL-3, roll is a rotation in the tool frame of reference. In RAPL-II, ROLL was a rotation in the world frame of reference.

See Also

pitch, pitchs	pitchs the tool by a specified angle
yaw, yaws	yaws the tool by a specified angle
xrot, xrots	rotates the tool about the world X axis
yrot, yrots	rotates the tool about the world Y axis
zrot, zrots	rotates the tool about the world Z axis

Category — Motion

rotacc

Sets or displays the maximum **rot**ational **acc**eleration.

See **rotacc_set**

rotacc_set

Alias **rotacc**

Description This parameter is used to **set** the maximum **rot**ational **acc**eleration the robot can achieve. The value is used when performing straight line motions in online mode and when using the teach pendant. It is not possible to set the value of this parameter to be greater than the default value which is robot dependent. Units are given in degrees/sec/sec.

Syntax `rotacc_set [value]`

Parameters If the *value* parameter is omitted, then **rotacc_set** simply displays the current value of the rotational acceleration.

Example1 `rotacc_set`

Sample Result `Current Rotational Acceleration is 25.0`

Example2 `rotacc_set 30`

Result `The rotational acceleration is set to 30 degrees/sec/sec`

RAPL-3 Language Same as **rotacc_set()** and **rotacc_get()**

RAPL-II Same as @CROTACC

See Also

accel	displays/sets acceleration
linacc_set	displays/sets linear acceleration
linspd_set	displays/sets linear speed
rotspd_set	displays/sets rotational speed
speed	displays/sets speed

Category Motion

rotspd

Sets or displays the maximum **rot**ational **sp**ee**d**.

See **rotspd_set**

rotspd_set

Alias **rotspd**

Description This parameter is used to **set** the maximum **rot**ational **sp**ee**d** the robot can achieve. The value is used when performing straight line motions in online mode and when using the teach pendant. It is not possible to set the value

of this parameter to be greater than the default value which is robot dependent. Units are given in degrees/sec.

Syntax `rotspd_set [value]`

Parameters If the *value* parameter is omitted, then **rotspd_set** simply displays the current value of the rotational acceleration.

Example1 `rotspd_set`

Sample Result `Current Rotational Speed is 180.0`

Example2 `rotspd_set 120`

Result `The rotational speed is set to 120 degrees/sec/sec`

RAPL-3 Language Same as **rotspd_set()** and **rotspd_get()**

RAPL-II Same as @CROTSPD

See Also

accel	displays/sets acceleration
linacc_set	displays/sets linear acceleration
linspd_set	displays/sets linear speed
rotacc_set	displays/sets rotational acceleration
speed	displays/sets speed

Category Motion

run

Description **Run**s (executes) the application's program.

Runs the program file with the same name as the current application and uses the variable file with the same name. For example, in the application named test, runs "test" with **"**test.v3".

Syntax `run`

Parameters There are no parameters.

Examples `run`

See Also

file_name	runs a specified program and specified variable file

Category Execution

save

Description **Save**s variables and values from the current database to a variable file.

If no file name is specified, the system saves to a file with the same name as the application. For example, in the application "test", it saves to "test.v3".

Syntax `save [file_name]`

Parameter One optional parameter.

file_name the name of the variable file

The file name can include a path to another directory.

The .v3 extension is optional. If not specified, it is added by the system.

Examples

```
save
save  test
save  test.v3
save  \app\final\final.v3
```

See Also **merge** merges variables from a file to the database

Category Variables, Values, and Locations

servoerr

Description Display the servo error detection parameters for each axis.

Syntax

```
servoerr
```

Parameters There are no parameters.

Examples

```
servoerr
```

Result

```
Servo Error Detection Parameters are as follows:

        Overspeed       Collision     Runaway      Timeout
        Threshold       Error Max     Error Max    Threshold
      (pulses/cycle)    (pulses)      (pulses)     (cycles)
Axis 1:        230,       500,      1000,       100
Axis 2:        230,       500,      1000,       100
Axis 3:        230,       500,      1000,       100
Axis 4:        300,       500,      1000,       100
Axis 5:        230,       500,      1000,       100
Axis 6:        300,       500,      1000,       100
```

RAPL-3 Language Similar to servoerr_params().

RAPL-II Similar to @SERVERR.

Category Arm Configuration

set

Alias **!**

Description **Set**s (assigns) a value to a variable in the current database.

If a variable does not exist, you can create a variable and set a value at the same time. Use a prefix to specify the data type, as detailed with the new command.

Syntax

```
set  variable_name  =  value
!    variable_name  =  value
set  type_prefix-variable_name  =  value
!    type_prefix-variable_name  =  value
```

Parameters Two required parameters:

variable_name the variable in the database

value the value being assigned to the variable, either:

a constant any one of:

a signed or unsigned integer

		signed or unsigned floating point number
		a simple string in double quotes
		a cartesian location in the form: **{***x, y, z, yaw, pitch, roll, e1, e2}*
	a variable	*variable_name_2*, any variable in the database
		a scalar variable: *variable_name_2*
		a one dimensional array: *variable_name_2*[*index*]
		a two dimensional array: *variable_name_2*[*index*][*index*]

If you set the value with a second variable, any subsequent value settings of that second variable do not affect the first variable.

To set the value using a second variable, that second variable must already be in the database.

Examples

```
set  number_of_loops  =  100
!    number_of_loops  =  100
set  place[2][3]  = {20,15,5,0,-90,0,0,0}
set  safe_place_5  =  safe_place_1
set  inspect[8]  =  dispense[8]
```

See Also

erase	erases a variable and value
eraseall	erases all variables and values
list	lists variables and values
new	creates a new variable
print	displays the value of a variable

Category Variables, Values, and Locations

speed

Description Displays the current **speed** setting, or sets the speed for all subsequent motions.

Display

Syntax `speed`

Parameters none

Example `speed`

Result `Current speed is 25`

Set

Syntax `speed` *percent*

Parameter Takes one parameter.

Percent	the percentage of full speed: an int

Examples

```
speed  50
speed  25
```

RAPL-3 Language — Same as speed_get(), speed_set(), and their alias, speed().

RAPL-II — Same as SPEED.

See Also

accel	displays/sets acceleration
linacc_set	displays/sets linear acceleration
linspd_set	displays/sets linear speed
rotacc_set	displays/sets rotational acceleration
rotspd_set	displays/sets rotational speed

Category — Motion

stance

Alias — **stance_set**

Description — Displays the current stance setting, or places the arm in a specified stance.

Stance is a specific configuration of a joint or joints.

Display

Syntax

```
stance
```

Parameters — There are no parameters.

Example

```
stance
```

Result

```
Requested stance:  forward up free
Physical  stance:  forward up noflip
```

Set

Syntax

```
stance  shoulder|elbow|wrist
```

Parameters — Three required parameters as one string.

shoulder — the stance of the shoulder, one of:

f	forward	toward front of arm
b	back	toward back of arm
x	free	controller chooses closest to current position
p	previous	last commanded stance position
c	current	current arm configuration

elbow — the stance of the elbow, one of:

u	up	away from base
d	down	towards base
x	free	controller chooses closest to current position
p	previous	last commanded stance position
c	current	current arm configuration

wrist the stance of the wrist, one of:

f	flip	4 and 5 rotated 180° and 5 reversed
n	noflip	no rotation or reversal
x	free	controller chooses closest to current position
p	previous	last commanded stance position
c	current	current arm configuration

Examples
```
stance  fun
stance  xxx
```

RAPL-3 Language Same as stance_get(), stance_set().

RAPL-II Same as POSE.

Category Motion

stance_set

Displays the current stance setting, or places the arm in a specified stance.

See **stance**

tool

Alias

tool_set

Description Displays the current **tool** transform or sets a tool transform, a re-definition of the origin point and orientation of the tool coordinate system.

If a tool transform is set, then the cfg_save command must be used in order to save it as part of the robot power on configuration. Do not run cfg_save if the tool transform being set is not the one that you want the robot to power on with.

Display

Syntax `tool`

Parameters There are no parameters.

Example
```
tool
```

Result
```
Tool Transform is:
tx=0.00000, ty=0.00000, tz=150.00000, yaw=0.00000,
pitch=0.00000, roll=0.00000
```

Set

Syntax `tool` *x*, *y*, *z*, *yaw*, *pitch*, *roll*

Parameters Six required parameters.

x the distance along the X axis, in current units: a float

y the distance along the Y axis, in current units: a float

z	the distance along the Z axis, in current units: a float
yaw	the rotation around the "normal" axis, in degrees: a float on an F3, rotation around the tool X axis, on an A465 or A255, rotation around the tool Z axis
pitch	the rotation around the "orientation" axis, in degrees: a float on an F3, A465, or A255, rotation around the tool Y axis
roll	the rotation around the "approach/depart" axis, in degrees: a float on an F3, rotation around the tool Z axis, on an A465 or A255, rotation around the tool X axis

Examples
```
tool  2.0, 0.0, 3.0, 0.0, 90.0, 0.0
```

RAPL-3 Language Same as tool_get(), tool_set().

RAPL-II Same as TOOL.

See Also **cfg_save** save robot configuration
base displays/sets a base offset

Category Coordinate Systems

tool_set

Displays the current **tool** transform or sets a tool transform.

See **tool**

tshift

Description Modifies a location by a specified distance and rotation in the tool coordinated system, a **t**ool system **shift**.

Syntax `tshift  location, toolX, toolY, toolZ, yaw, pitch, roll`

Parameters Seven required parameters

location	the cartesian location variable to modify
toolX	the offset in the tool X direction.
toolY	the offset in the tool Y direction.
toolZ	the offset in the tool Z direction.
yaw	the rotation around the "normal" axis, in degrees on an F3, rotation around the tool X axis, on an A465 or A255, rotation around the tool Z axis
pitch	the rotation around the "orientation" axis, in degrees on an F3, A465, or A255, rotation around the tool Y axis
roll	the rotation around the "approach/depart" axis, in degrees on an F3, rotation around the tool Z axis, on an A465 or A255, rotation around the tool X axis

Example
```
tshift  myloc, 1.5, 0, 0, 0, 0, 22.5
```

RAPL-3 Language Same as shift_t().

RAPL-II No equivalent.

See Also **wshift** shift a location in the world coordinate system

Category Coordinate Systems

tx, txs

Description In the **t**ool frame of reference, jogs the tool centre point along the **X** axis by a specified amount.

The motion is joint interpolated with tx and cartesian interpolated, **s**traight line, with txs.

Syntax
```
tx  distance
txs distance
```

Parameters There is one required parameter.

distance the distance to move the tool centre point, in the current units

Examples
```
tx  -100
txs  4.5
```

RAPL-3 Language Same as jog_t(TOOL_X), tx, and jog_ts(TOOL_X), txs.

RAPL-II No equivalent. Similar to JOG, but in the tool frame of reference.

See Also

ty, tys	jog the tool centre point along the tool Y axis
tz, tzs	jog the tool centre point along the tool Z axis
yaw, yaws	jog the tool centre point around the tool “normal” axis
pitch, pitchs	jog the tool centre point around the tool “orientation” axis
roll, rolls	jog the tool centre point around the tool “approach” axis

Category Motion

ty, tys

Description In the **t**ool frame of reference, jogs the tool centre point along the **Y** axis by a specified amount.

The motion is joint interpolated with ty and cartesian interpolated, **s**traight line, with tys.

Syntax
```
ty  distance
tys distance
```

Parameters There is one required parameter.

distance the distance to move the tool centre point, in the current units

Examples
```
ty   20
tys –4.5
```

RAPL-3 Language Same as jog_t(TOOL_Y), ty, and jog_ts(TOOL_Y), tys.

RAPL-II No equivalent. Similar to JOG, but in the tool frame of reference.

See Also

tx, txs	jog the tool centre point along the tool X axis
tz, tzs	jog the tool centre point along the tool Z axis
yaw, yaws	jog the tool centre point around the tool “normal” axis

pitch, pitchs	jog the tool centre point around the tool "orientation" axis
roll, rolls	jog the tool centre point around the tool "approach" axis

Category: Motion

tz, tzs

Description: In the **t**ool frame of reference, jogs the tool centre point along the **Z** axis by a specified amount.

The motion is joint interpolated with tz and cartesian interpolated, **s**traight line, with tzs.

Syntax:
```
tz  distance
tzs distance
```

Parameters: There is one required parameter.

distance the distance to move the tool centre point, in the current units

Examples:
```
tz   300
tzs -4.5
```

RAPL-3 Language: Same as jog_t(TOOL_Z), tz, and jog_ts(TOOL_Z), tzs.

RAPL-II: No equivalent. Similar to JOG, but in the tool frame of reference.

See Also:

tx, txs	jog the tool centre point along the tool X axis
ty, tys	jog the tool centre point along the tool Y axis
yaw, yaws	jog the tool centre point around the tool "normal" axis
pitch, pitchs	jog the tool centre point around the tool "orientation" axis
roll, rolls	jog the tool centre point around the tool "approach" axis

Category: Motion

unlock

Description: **Unlock**s one, more than one, or all joints.

Syntax:
```
unlock  [axis], [axis] ...
```

Parameters: Zero or more optional parameters. If no parameter is given then all axes are unlocked.

axis the axis to be unlocked

Examples:
```
unlock  7
unlock  2,  3
```

RAPL-3 Language: Same as unlock().

RAPL-II: Same as UNLOCK.

See Also: lock locks joint(s)

Category: Motion

use

Description For systems with more than one robot.

Displays or selects the robot for communication with ASH. More specifically, displays or selects the socket of interprocess communication.

Display

Syntax `use`

Parameters none

Example `use`

Result `using 'DEFAULT'`

Select

Syntax `use` *`socket`*

Parameters One parameter, a string between double quotes.

socket	the path to the socket in the filesystem: a string
DEFAULT	the string to reset to the default robot

Examples
```
use "/dev/robot"
use "DEFAULT"
```

RAPL-3 Language Same as server_set / server_get

RAPL-II No equivalent.

Category Machine

w0

Displays the commanded position.

See **wcmd**

w1

Description Continually displays the actual position (where the arm has actually gone), in motor pulses. Is terminated by typing Ctrl-E.

Syntax `w1`

Parameter There are no parameters.

Example `w1`

Result
```
Actual Position (motor pulses):
-1      -2      -1      -1      -5
```

RAPL-3 language Similar to pos_get().

RAPL-II Same as W1.

See Also **wact** displays the actual position

Category Calibration, Homing, and Status

w2

Displays the actual position.

See **wact**

w3

Description Continually displays the commanded position (where the controller has commanded the arm to go), in motor pulses. Is terminated by typing Ctrl-E.

Syntax `w3`

Parameters There are no parameters.

Examples w3

Result
```
Commanded Position (motor pulses):
-1      -2      -1      -1      -5
```

RAPL-3 language Similar to pos_get().

RAPL-II Same as W3.

See Also **wcmd** displays commanded position

Category Calibration, Homing, and Status

w4

Displays the endpoint position.

See **wend**

w5

Description Continually displays the position error, that is, the difference between where the controller has commanded the arm to go and where it actually is, in motor pulses. Is terminated by typing Ctrl-E.

Syntax `w5`

Parameters There are no parameters.

Examples w5

Result
```
Position Error (motor pulses):
        +0      +0      +0      +0      +0      +0      +0
```

RAPL-3 language Similar to pos_get().

RAPL-II Same as W5.

See Also **wact** displays actual position
wcmd displays commanded position

Category Calibration, Homing, and Status

wact

Alias **w2**

Description Displays the actual robot position in motor counts, joint angles and world coordinates. Displays **w**here **act**ual.

Syntax `wact`

Parameters There are no parameters.

Example `wact`

Result

```
Actual Position :

       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
PULSES       -1        -34         -1         -1         -5         +0
             +0         +0

       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
JOINTS   -0.005     +0.170     -0.005     +0.022     -0.270     +0.000
         +0.000     +0.000

       X/TrackX   Y/TrackY          Z      Z-Rot      Y-Rot      X-Rot
WORLD      (in)       (in)       (in)      (deg)      (deg)      (deg)
        +22.000     -0.002    +10.030     -0.005     -0.023     -0.270
         +0.000     +0.000
```

RAPL-3 Language Similar to pos_get().

RAPL-II Same as W2.

See Also **wcmd** where commanded
wend where endpoint
here stores or displays current robot position

Category Calibration, Homing, and Status

wcmd

Alias **w0**

Description Displays the commanded position (where the controller has commanded the arm to go), in motor pulses, joint angles, and world coordinates. Displays **w**here **co**m**m**an**d**ed.

Syntax `wcmd`

Parameters There are no parameters.

Example `wcmd`

Result

```
Commanded Position :
```

```
       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
PULSES       -1        -34         -1         -1         -5         +0
             +0         +0

       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
JOINTS   -0.005     +0.170     -0.005     +0.022     -0.270     +0.000
         +0.000     +0.000

       X/TrackX   Y/TrackY          Z      Z-Rot      Y-Rot      X-Rot
WORLD       (in)       (in)       (in)      (deg)      (deg)      (deg)
        +22.000     -0.002    +10.030     -0.005     -0.023     -0.270
         +0.000     +0.000
```

RAPL-3 Language Similar to pos_get().

RAPL-II Same as W0.

See Also
wact where actual
wend where endpoint
here stores or displays current robot position

Category Calibration, Homing, and Status

---

wend

Alias **w4**

Description Displays the robot endpoint position in motor pulses, joint angles and world coordinates. Displays **w**here **end**point.

Syntax `wend`

Parameters There are no parameters.

Example `wend`

Result `Endpoint Position:`

```
       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
PULSES       +0         +0     -51200         +0         +0         +0
        +204032         +0

       Axis 1/7   Axis 2/8     Axis 3     Axis 4     Axis 5     Axis 6
JOINTS   +0.000     +0.000    -90.000     +0.000     +0.000     +0.000
        +59.023     +0.000

       X/TrackX   Y/TrackY          Z      Z-Rot      Y-Rot      X-Rot
WORLD       (in)       (in)       (in)      (deg)      (deg)      (deg)
        +17.520     +0.000    +24.213     +0.000    +90.000     +0.000
        +59.023     +0.000
```

RAPL-3 Language Similar to pos_get().

RAPL-II Same as W4.

See Also
wact where actual
wcmd where commanded
here stores or displays current robot position

Category Calibration, Homing, and Status

wgrip

Description — Displays the current distance between fingers of a servo gripper. Displays **w**hat **grip**per distance.

Gripper type must be set to 2 (servo) for this wgrip command to work.

Syntax — `wgrip`

Parameters — There are no parameters.

Example — `wgrip`

Result — `Gripper distance = 2.80473`

RAPL-3 Language — Same as gripdist_get().

RAPL-II — Same as !X=WGRIP().

See Also — **grip** moves fingers to specified distance
gtype displays/sets the type of gripper used (air, servo, none)

Category — Gripper

wshift

Description — Modifies a location by a specified distance and rotation in the world coordinate system, a **w**orld system **shift**.

Syntax — `wshift location, worldX, worldY, worldZ, zrot, yrot, xrot`

Parameters — Seven required parameters:

location	the cartesian location variable to modify
worldX	the offset in the world X direction
worldY	the offset in the world Y direction
worldZ	the offset in the world Z direction
zrot	the rotation about the world Z axis, in degrees
yrot	the rotation about the world Y axis, in degrees
xrot	the rotation about the world X axis, in degrees

Examples — `wshift myloc, 62.5, 0, 0, 0, 0, 45`

RAPL-3 Language — Same as shift_w().

RAPL-II — Same as SHIFTA.

See Also — **tshift** shift a location in the tool coordinate system

Category — Coordinate Systems

wx, wxs

Description — In the **w**orld frame of reference, jogs the tool centre point along the **X** axis by a specified amount.

The motion is joint interpolated with wx and cartesian interpolated, **s**traight line, with wxs.

Syntax
```
wx  distance
wxs distance
```

Parameters — One required parameter.

distance — the distance to move the tool centre point, in the current units

Examples
```
wx  200
wxs -4.5
```

RAPL-3 Language — Same as jog_w(WORLD_X), wx() and jog_ws(WORLD_X), wxs().

RAPL-II — Same as X. Similar to JOG.

See Also

wy, wys	jog the tool centre point along the world Y axis
wz, wzs	jog the tool centre point along the world Z axis
xrot, xrots	jog the tool centre point around the world X axis
yrot, yrots	jog the tool centre point around the world Y axis
zrot, zrots	jog the tool centre point around the world Z axis

Category — Motion

wy, wys

Description — In the **w**orld frame of reference, jogs the tool centre point along the **Y** axis by a specified amount.

The motion is joint interpolated with wy and cartesian interpolated, **s**traight line, with wys.

Syntax
```
wy  distance
wys distance
```

Parameters — One required parameter.

distance — the distance to move the tool centre point, in the current units

Examples
```
wy  300
wys -4.5
```

RAPL-3 Language — Same as jog_w(WORLD_Y), wy() and jog_ws(WORLD_Y), wys().

RAPL-II — Same as Y. Similar to JOG.

See Also

wx, wxs	jog the tool centre point along the world X axis
wz, wzs	jog the tool centre point along the world Z axis
xrot, xrots	jog the tool centre point around the world X axis
yrot, yrots	jog the tool centre point around the world Y axis
zrot, zrots	jog the tool centre point around the world Z axis

Category — Motion

wz, wzs

Description — In the **w**orld frame of reference, jogs the tool centre point along the **Z** axis by a specified amount.

The motion is joint interpolated with wz and cartesian interpolated, **s**traight line, with wzs.

Syntax
```
wz  distance
wzs distance
```

Parameters — One required parameter.

distance — the distance to move the tool centre point, in the current units

Examples
```
wz  400
wzs -4.5
```

RAPL-3 Language — Same as jog_w(WORLD_Z), wz() and jog_ws(WORLD_Z), wzs().

RAPL-II — Same as Z. Similar to JOG.

See Also

wx, wxs	jog the tool centre point along the world X axis
wy, wys	jog the tool centre point along the world Y axis
xrot, xrots	jog the tool centre point around the world X axis
yrot, yrots	jog the tool centre point around the world Y axis
zrot, zrots	jog the tool centre point around the world Z axis

Category — Motion

ver

Description — Displays the **ver**sion of the application shell being used.

Syntax
```
ver
```

Parameter — There are no parameters.

Example
```
ver
```

Result
```
ash (application shell) Revision: 1.62
```

RAPL-3 Language — No equivalent.

RAPL-II — No equivalent.

See Also

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

Category — Start Up and Exit

xrot, xrots

Description — In the world frame of reference, jogs the tool centre point around the X axis by a specified amount. Performs an **X rot**ation.

The motion is joint interpolated with xrot and cartesian interpolated, **s**traight line, with xrots.

Syntax
```
xrot   angle
xrots  angle
```

Parameters
One required parameter.

angle the distance to move the tool centre point, in degrees

Examples
```
xrot   45
xrots -22.5
```

RAPL-3 Language
Same as jog_w(WORLD_XROT), xrot() and jog_ws(WORLD_XROT), xrots().

RAPL-II
Same as ROLL. In RAPL-II, ROLL was the rotation in the world frame of reference around the X axis. In RAPL-3, this is called xrot.

See Also

yrot	rotation around the world Y axis
zrot	rotation around the world Z axis
wx	jog along the world X axis
wy	jog along the world Y axis
wz	jog along the world Z axis

Category
Motion

yaw, yaws

Description
Yaws, rotates, the tool by a specified angle about the tool "normal" axis.

The motion from the current position is joint interpolated with yaw and cartesian interpolated, **s**traight line, with yaw**s**. With the yaws command, the tool centre point stays on the axis, in the same place, while the tool rotates around the axis.

Caution
The yaws command should only be used with online mode on.

Syntax
```
yaw angle
yaws angle
```

Parameters
There is one required parameter.

angle the number of degrees to rotate the tool

Examples
```
yaw  2.5
yaws 10
```

RAPL-3 Language
Same as yaw() and yaws()

RAPL-II
No equivalent. In RAPL-3, yaw is a rotation in the tool frame of reference. In RAPL-II, YAW was a rotation in the world frame of reference.

See Also

pitch, pitchs	pitch the tool by a specified angle
roll, rolls	roll the tool by a specified angle
xrot, xrots	rotate the tool about the world X axis
yrot, yrots	rotate the tool about the world Y axis
zrot, zrots	rotate the tool about the world Z axis

Category
Motion

yrot, yrots

Description
In the world frame of reference, jogs the tool centre point around the Y axis by a specified amount. Performs a **Y rot**ation.

The motion is joint interpolated with yrot and cartesian interpolated, **s**traight line, with yrots.

Syntax
```
yrot   angle
yrots  angle
```

Parameters
One required parameter.

angle the distance to move the tool centre point, in degrees

Examples
```
yrot   45
yrots -22.5
```

RAPL-3 Language
Same as jog_w(WORLD_YROT), yrot() and jog_ws(WORLD_YROT), yrots().

RAPL-II
Same as PITCH. In RAPL-II, PITCH was the rotation in the world frame of reference around the Y axis. In RAPL-3, this is called yrot.

See Also
xrot rotation around the world X axis
zrot rotation around the world Z axis
wx jog along the world X axis
wy jog along the world Y axis
wz jog along the world Z axis

Category
Motion

zrot, zrots

Description
In the world frame of reference, jogs the tool centre point around the Z axis by a specified amount. Performs a **Z rot**ation.

The motion is joint interpolated with zrot and cartesian interpolated, **s**traight line, with zrots.

Syntax
```
zrot   angle
zrots  angle
```

Parameters
One required parameter.

angle the distance to move the tool centre point, in degrees

Examples
```
zrot   45
zrots -22.5
```

RAPL-3 Language
Same as jog_w(WORLD_ZROT), zrot() and jog_ws(WORLD_ZROT), zrots().

RAPL-II
Same as YAW. In RAPL-II, YAW was the rotation in the world frame of reference around the Z axis. In RAPL-3, this is called zrot.

See Also
xrot rotation around the world X axis
yrot rotation around the world Y axis
wx jog along the world X axis
wy jog along the world Y axis
wz jog along the world Z axis

Category	Motion

Features

One system shell feature is accessible through the application shell.

&

Description — Executes a program in the background. Allows you to get back the ASH prompt to execute other commands while the program is running.

Syntax — *program_name* &

Examples —
```
test.r:test.v3 &
test &
```

See Also — ***program_name*** runs the program
run runs the default program

System Shell Commands

Most system shell commands are accessible through the application shell. They are listed here. For details about system shell commands, see the command descriptions in the system shell part of this *Guide*.

Accessible from ASH

These system shell commands are accessible through the application shell. These are listed alphabetically.

System shell commands accessible from the application shell.	
/diag/cal	calibrate robot axes
/diag/calgrip	calibrate the servo gripper
/diag/configur	master configuration program for setting up the robot
/diag/encres	reset the joint position encoders (F series only)
/diag/pack	move an F3 robot into its packing position
/diag/xzero	zero a particular motor position register
/diag/zero	all motor position registers zero
axst	display the status of the robot axes
cd	change current working directory
chmod	change protection mode
cksum	calculate checksum of file
cp (or copy)	copy file
date	display or set date and time
df	display space on file system
kill	terminate a process
ln	make link to file
ls (or dir)	list directory contents
mem	display space in memory
mkdev	make device
mkdir (or md)	make directory
mkfifo	make fifo
mksock	make socket
more	display contents of file
mount	mount a file system on a directory
mv (or move)	move or rename file
pause	wait for user to type enter
ps	display status of processes

pwd	displays current working directory
rm (or del)	remove/delete or unlink file
rmdir	remove/delete directory
shell	start new system shell
siocfg	reconfigure serial port
sync	defragment memory
unmount	unmount a file system from a directory

The cd, kill, and mem commands are actually built in ASH.

Not Accessible from ASH

These system shell commands are not accessible from the application shell.

System shell commands NOT accessible from the application shell.	
crosver	display version of operating system (CROS)
do	execute a shell script
echo	echo a message to the console
exit	exit from system shell (the exit command in ASH exits from ASH)
help	help on system shell commands (the help command in ASH gives help on ASH commands)
msleep	put system shell to sleep
shutdown	shut down the system
type	display contents of file
ver	display version of system shell (the ver command in ASH displays the version of ASH)