
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 bypasses 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 RAPT-3 program.

```
new counter
```

Identifiers

The variable name follows the rules for RAPT-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
<code>new count er [2, 2]</code>	a two-dimensional array of integers count er [0, 0], count er [0, 1], count er [1, 0], count er [1, 1]
<code>new %di ff [2, 5]</code>	a two-dimensional array of floats di ff [0, 0], di ff [0, 1], di ff [0, 2], di ff [0, 3], di ff [0, 4], di ff [1, 0], di ff [1, 1], di ff [1, 2], di ff [1, 3], di ff [1, 4]
<code>new _saf e[5, 16]</code>	a two-dimensional array of cartesian locations saf e[0, 0] ... saf e[0, 15] ... saf e[4, 0] ... saf e[4, 15] ...
<code>new #di spense[10, 24]</code>	a two-dimensional array of precision locations di spense[0, 0] ... di spense[0, 23] ... di spense[9, 0] ... di spense[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
<code>new count er [2] [2]</code>	a two-dimensional array of integers count er [0, 0], count er [0, 1], count er [1, 0], count er [1, 1]
<code>new %di ff [5] [2]</code>	a two-dimensional array of floats di ff [0, 0], di ff [0, 1], di ff [0, 2], di ff [0, 3], di ff [0, 4], di ff [1, 0], di ff [1, 1], di ff [1, 2], di ff [1, 3], di ff [1, 4]
<code>new _saf e[16] [5]</code>	a two-dimensional array of cartesian locations saf e[0, 0] ... saf e[0, 15] ... saf e[4, 0] ... saf e[4, 15] ...
<code>new #di spense[24] [10]</code>	a two-dimensional array of precision locations di spense[0, 0] ... di spense[0, 23] ... di spense[9, 0] ... di spense[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.

```
er ase   number_of_loops
er ase   pick_1
er ase   safe
er ase   dispense
```

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

```
er aseal l
```

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