

NAME

syntax – interpreter syntax

DESCRIPTION

This syntax summary attempts to present all the built-in keywords, operators and commands of `spec`. The grammar rules listed below are very similar to those given in the standard manuals describing the C language. Operators are listed in order of precedence, with the highest precedence operators listed first.

The following terms are used:

lvalue – “left value”, something that can go on the left side of the equals sign.

binop – binary operator (+, -, etc.)

asgnop – assignment operator (=, +=, etc.)

identifier – a variable

identifier-list – a space- or comma-separated list of identifiers

pattern – An alphanumeric string possibly containing the meta characters ? or *.

pattern-list-opt – An optional space-separated list of patterns.

expr-opt – an optional expression

expr-list-opt – an optional expression list

[;] – A semicolon or a newline. (A semicolon after a statement is optional if the statement is followed by a newline.)

EXPRESSIONS

expression:

lvalue

numeric-constant

string-constant

(*expression*)

function (*expression-list*)

- *expression*

! *expression*

~ *expression*

++ *lvalue*

-- *lvalue*

lvalue ++

lvalue --

expression ? *expression* : *expression*

expression binop expression

lvalue asgnop expression

expression-list

expression , *expression*

lvalue:

identifier

identifier [*expression*]

binop:

* / %

+ -

>> <<

> < <= >=

```

==  !=
&
^
|
&&
||

```

asgnop:

```

=  +=  -=  *=  /=  %=  >>=  <<=  &=  ^=  |=

```

array_type:

```

byte
ubyte
short
ushort
long
ulong
float
double
string

```

array_decl

```

[ array_type ] array identifier [ expr ]
[ array_type ] array identifier [ expr ] [ expr ]

```

array_decl-list

```

array_decl [ , ] array_decl

```

Statements**compound-statement:**

```

{ statement-list }

```

statement-list:

```

statement
statement statement-list

```

statement:

```

compound-statement
expression [ ; ]
if ( expression ) statement
if ( expression ) statement else statement
while ( expression ) statement
for ( expr-list-opt; expr-opt; expr-list-opt ) statement
for ( identifier in identifier ) statement

break [ ; ]
continue [ ; ]
exit [ ; ]
return expr-opt [ ; ]

history expr-opt [ ; ]
lscmd pattern-list-op [ ; ]
syms pattern-list-op [ ; ]

print expression-list [ ; ]
eprint expression-list [ ; ]

global identifier-list [ ; ]
constant identifier expression [ ; ]

```

```

unglobal identifier-list [ ; ]
local identifier-list [ ; ]
delete identifier [ expression ] [ ; ]

array_decl-list [ ; ]
local array_decl-list [ ; ]
global array_decl-list [ ; ]
shared array_decl-list [ ; ]
extern shared array_decl-list [ ; ]

def identifier string-constant [ ; ]
rdef identifier expression [ ; ]
undef identifier-list [ ; ]
prdef pattern-list-op [ ; ]
lsdef pattern-list-op [ ; ]

memstat [ ; ]
savstate [ ; ]
reconfig [ ; ]
getcounts [ ; ]
move_all [ ; ]
move_cnt [ ; ]
sync [ ; ]
quit [ ; ]

```

FUNCTIONS**0 or 1 arguments:**

chdir()	input()	on()
open()	rand()	stop()
time()	wait()	

0 to 2 arguments:

date()

0 to 3 arguments:

unix()

1 argument:

acos()	asc()	asin()
atan()	bcd()	close()
cnt_mne()	cnt_name()	cnt_num()
cos()	dcb()	deg()
exp10()	exp()	fabs()
getenv()	gethelp()	gpib_poll()
int()	length()	log10()
log()	mcount()	motor_mne()
motor_name()	motor_num()	off()
plot_cntl()	port_get()	port_getw()
rad()	set_sim()	sin()
sleep()	sqrt()	srand()
tan()	tcount()	tty_cntl()

1 or more arguments:

array_dump()	array_op()	array_plot()
data_dump()	data_plot()	eprintf()
printf()	sock_io()	sock_par()
sprintf()	vme_get()	vme_get32()

1 or 2 arguments:

anka_par()	ca_cntl()	calc()
dofile()	em_io()	eval()
fbus_get()	file_info()	getline()
getsval()	getval()	gpib_get()
mca_sel()	qdofile()	read_motors()
remote_poll()	remote_stat()	ser_get()
sock_get()	spec_par()	whatis()
yesno()		

1 to 3 arguments:

epics_get()	epics_put()	mca_par()
-------------	-------------	-----------

1 to 4 arguments:

array_pipe()	cdef()	data_pipe()
esrf_dc()	mca_get()	mca_put()

1 to 5 arguments:

esrf_io()

1 to 17 arguments:

fbus_put()

2 arguments:

array_read()	atan2()	ca_get()
chg_offset()	clone()	data_info()
dial()	fmt_close()	get_lim()
gpib_cntl()	image_get()	image_put()
index()	port_put()	port_putw()
pow()	prop_get()	prop_watch()
remote_cmd()	remote_eval()	user()

2 or more arguments:

array_fit()	data_fit()	fprintf()
prop_send()	sscanf()	vme_move()
vme_put()	vme_put32()	

2 or 3 arguments:

chg_dial()	epics_par()	esrf_db()
gpib_put()	remote_async()	ser_par()
ser_put()	sock_put()	split()
substr()	tty_move()	

2 to 4 arguments:

counter_par()	data_read()	mca_spar()
motor_par()	plot_move()	

2 to 5 arguments:

mca_sget()	mca_sput()
------------	------------

2 to 6 arguments:

image_par()

3 arguments:

ca_put()	data_get()	data_grp()
prop_put()	set_lim()	

3 or more arguments:

data_nput()

3 or 4 arguments:

anka_get()

ca_fna()

3 to 5 arguments:

fmt_read()

fmt_write()

4 arguments:

data_put()

plot_range()

tty_fmt()

4 or 5 arguments:

anka_put()

5 or 6 arguments:

data_uop()

6 or 7 arguments:

data_anal()

7 arguments:

data_bop()

SEE ALSO

funcs flow