## 6.2. A Quick Tour of SKILL® Programming

SKILL is the extension language for Cadence™ tools. With the extension capability, designers can readily add new capabilities with complex built-in functions to Cadence design tool suite. The power of SKILL is derived from these large libraries of subroutine calls to manipulate design data structures like nets, instances, cells, etc... Without these built-in routines, SKILL probably would not achieve its dominant role as it plays today in the EDA (Electronic Design Automation) for custom circuit area. The "Cadence SKILL Functions Quick Reference" is a very comprehensive manual with about 500 pages full built-in SKILL commands, and it is still growing (i.e. **extendibility** capability) every day. Luckily, most of us need to know only a handful of SKILL commands to be able to start exploring around.

If you are proficient in other scripting languages and you plan for a "quick" pickup and run with SKILL language just like picking up one more programming language to deal with, you will get a little jolt unless you have prior experience with **LISP** programming language or one of its derivative dialects like **Scheme**. Even though the author was proficient other scripting languages before plunging into SKILL, it took the author a while longer to learn how to "tame" the basics of SKILL syntax, compared to absorbing other languages.

Using SKILL in practical application would involve handling the Cadence database (CDB) or OpenAccess (OA) design structure. This would require an extra step of learning process, especially understanding the Cadence® Design Framework II Tech DB CellView Information Model. This section will discuss the basics of SKILL programming only and does not involve in any database. Its goal is to get the novice reader familiar with the foundations of SKILL syntax before plunging into the real applications in the regular SKILL chapters.

As a scritping language, SKILL can also be used as a Shell language like the Unix Bash or C shells. As a shell language, you can run SKILL shell in batch or interactive mode. To run SKILL in interactive mode from a Unix/Linux terminal:

Unix> .....installed cadence path...../**tools/dfII/bin/skill**

You can also launch and execute SKILL commands from Bash scripts or other shell scripts. To run SKILL shell in batch mode, the first line of the SKILL script must start with a #! directive:

#!.....installed cadence path...../**tools/dfII/bin/skill**

First, you need to know the base directory where the Cadence software is installed. The SKILL executable binary should be in (Cadence installed directory)/tools/dfII/bin/skill. Ask your system administrator about the Cadence installed directory. Or if you happen to be able to logon to Cadence, you can execute command "**getInstallPath**" at the CIW (Command Interpreter Window or Cadence Interface Window as some would call) to tell you

where the base Cadence install full pathname is located.

- ciw> **getInstallPath()**

This chapter is to introduce to the novice readers the "pure" SKILL programming. Pure SKILL is the virgin SKILL code with no (DB) database access or other enhancement goodies that turn raw SKILL into a powerful in IC design tool environment that most IC designers are familiar with.

We will go over the basic command constructs of SKILL scripting language. All examples in this chapter do not involve any database access or library setup. A good way to get a feel of the power of any programming language is to dive in and explore the data structure of that programming language. Since SKILL is originated from **LISP** (i.e. **LIS**t **P**rocessing or LISt Programming), we will have many problems dealing with the SKILL list data structure and its manipulation.

### 6.2.1 SKILL: Interactive shell commands at a glance

This session is to provide sample SKILL commands so the readers can get the feel of the SKILL programming environment. The task is to let the readers get familiar with the SKILL code syntax. If you get problem understanding the code snippets, don't worry. Keep on reading and come back later. We also attach some one-liner code snippets of other scripting languages (Perl, Ruby, Python) at the end of some examples. This is for command-to-command reference purpose if you know one the common scripting languages. If you have previously exposed to Scheme or LISP, you will see that many SKILL command constructs are similar to Scheme or LISP.

Once you try and execute the commands below successfully, you can grab and save the commands into a file for later execution. You execute these below commands in CIW window or at SKILL shell window. To run SKILL on shell window, read section 6.2.7 for more information.

Depending on your style, you have the choice to put parentheses "inside" or "outside" the command construct. For example, the three below commands are equivalent with various placements of parentheses.

- **for( i 1 3 print( x ) )**
- **for( i 1 3 ( print x ) )**
- **( for i 1 3 ( print x ) )**

If after you enter the command and see no response from the system, the system may be still waiting for you to enter more commands. Try to hit symbol "]" (i.e. super right parenthesis) to close all parentheses to see if that fixes the problem.

```
unix> skill                  #invoke SKILL interactive mode

>                            #SKILL prompt in interactive mode

> 8 * 11.1                   #arithmetic: (8 * 11.1)

88.8                         #x=let((a b) a=4 b=6 a-b)
```

---

```
> a=5 b=6                    #assign variable: x={a=5 b=6 a+b}

> print a+b                  #add variable: (print a+b)

11

* Perl: $a=5; $b=6; print $a+$b
* Ruby: a=5; b=6; print a+b
* Python: a=5; b=6; print a+b
* Tcl: set a 5 ; set b 6 ; puts [expr $a + $b]
```

---

```
--1-- for loop

> for( i 1 3 printf("%5.2f\n" i*1.1 ))          #  for( .. )
> ( for i 1 3 printf("%5.2f\n" i*1.1 ))         # (for  .. )
> ( for i 1 3 (printf "%5.2f\n" i*1.1 ))        # (for .. (..) )

  1.10
  2.20
  3.30

*Perl: for ($i=1; $i<=3; $i++) { printf("%5.2f\n", $i*1.1 )}
*ruby: 1.upto(3) { |i| printf("%5.2f\n", i*1.1 )}
*Python: for i in range(1,4): print "%5.2f\n" %(i*1.1),
*Tcl:for{set i 1}{$i<=3}{incr i}{puts[format"%5.2f"[expr $i*1.1]]}
```

---

```
--2-- foreach looping with immediate items with list(. . .)

> foreach(i list("the" "cad") print(upperCase(i)) )

"THE" "CAD"

*Perl: foreach $i ("the","cad") {print uc($i) }
*Ruby: ["the","cad"].each { |i| print i.upcase }
*Python: for i in ("the","cad"): print i.upper()
*Tcl: foreach i [list the cad] {puts [string toupper $i]}
```

---

```
--3-- foreach looping with immediate items with list `( . . .)

> n=0 foreach(i `(0 1 2 3)  n=n+i printf("%L %L\n" i n ) )
```

```
0 0
1 1
2 3
3 6

*Perl: foreach $i qw(0 1 2 3) {$n+=$i;printf("%s %s\n", $i,$n)}
*Ruby: n=0 ; [0,1,2,3].each { |i| n+=i ;printf("%d %d\n", i,n )}
*Python: n=0 ; for i in [0,1,2,3]: n+=i ;print "%d %d" %(i,n)
*Tcl: set n {0}; foreach i [list 0 1 2 3] {incr n $i ;puts "$i $n"}
```

---

```
--4-- Looping a list

> x=list("THE" "CAD")

> for(i 0 length(x)-1 printf("%L \n" lowerCase(nth(i x))  i ) )

"the" 0
"cad" 1

*Perl: for($i=0;$i<=$#x;$i++) {printf("%s\n",lc($x[$i]))}
*Ruby: x.each_with_index do |y,i| printf("%s\n",x[i].downcase) end
*Python: for i,y in enumerate(x): print "%s" %(x[i].lower())
*Tcl:for {set i 0}{$i < [llength $x]}{incr i} {
*      puts [format "%s" [string tolower [lindex $x $i]]]}
```

---

```
--5-- when

> x=5 when( 9 > x print("Larger than") print(x) )

"Larger than" 5

* Perl: $x=5; if (9 > $x) {print "larger than $x" }
* Ruby: x=5; if 9 > x then print "larger than #{x}" end
* Python: x=5; if 9 > x : print "larger than %s" %(x)
* Tcl: set x {5}; if { 9 > $x } { puts "larger than $x"}
```

---

```
--6-- if then else

> x=5 if( x > 9 then print("High") else print("Low") )
> x=5 (if  x > 9 print("High") print("Low") )

"Low"

* Perl: $x=5; if ($x > 9) {print "high"} else {print "low"}
* Ruby: x=5; if x > 5 then print "high" else print "low" end
* Python: x=5; if x>9: print "high";else: print "low"
* Tcl: set x {5}; if {$x>9} {puts "high"} else {puts "low"}
```

---

```
--7-- Set default if variable not declared

> if( !boundp('x) x=5 )
```

```
> when( !boundp('x) x=5 )

> boundp('x) || (x=5)

> unless( boundp('x)  x=5 )

5

* Perl: $y=7 if !defined $y
* Ruby: y=7 if y==nil
* Python: try: y; except NameError: y=7
```

---

```
--8-- Substring & strcat

ciw> strcat( substring( "connection" 1 5) "CTION" )

conneCTION

* Perl: print substr("connection",0,5) . "CTION"
* Ruby: x="connection"; print x[0:5] + "CTION"
* Python: x="connection"; print x[0:5] + "CTION"
* Tcl: set y [string range "connection" 0 4]CTION; puts $y
```

---

```
--9-- foreach( mapcar ...) to return the corresponding results

x=foreach( mapcar i '(1 2 3 4) i*i ) )     print x

(1 4 9 16)

* Perl: @x=map { $_*$_ } qw(1 2 3 4);
* Ruby: x=[1,2,3,4].map do |y| y*y end
* Python: x=[i*i for i in [1,2,3,4]]
```

---

```
--10-- no mapcar; no change; return original elements

y=foreach(i '(1 2 3 4) i*i )     print y

(1 2 3 4)

* Perl: @x=grep { $_*$_ } qw(1 2 3 4);
```

---

```
--11-- setof to filter out elements

setof( i '( 1 2 3 4 5 6 7 8 9) ( i > 3 && i <= 7 )

(4 5 6 7)

* Perl: grep { $_ > 3 && $_ <= 7 } 1..9
```

```
* Ruby: print (1..9).to_a.find_all { |x| x > 3 && x <=7 }
* Python: print [x for x in range(1,10,1) if x > 3 and x <= 7 ]
```

---

```
--12-- substitute "BAD" to "good" in a list

> print subst( "good" "BAD" list( "A9" "78" "BAD" "34" ))

( "A9" "78" "good" "34" )

*Perl: @x=qw(A9 78 BAD 34);print map { s/BAD/good/;$_ } @x;
*Ruby:x=%w[A9 78 BAD 34];print x.map{|x| x.to_s.gsub("BAD","good")}
*Python: print [y.replace("BAD","good") for y in x]
```

---

```
--13-- Change din<2> to din_2

> buildString( parseString("din<2>" "<>") "_")

din_2

*Perl: join( "_", split("[<>]","din<2>") )
*Ruby: print "din<2>".split(/[<>]/).join("_")
*Python: print "_".join(re.compile(r"[<>]+").split("din<2>"))
```

---

```
--14-- Change in<2> to in_2_ with rexCompile/rexReplace

> rexCompile("[<>]") print rexReplace("in<2> "_" 0)

in_2_

* Perl: s/[<>]/_/g
* Ruby: print "din<2>".gsub!(/[<>]/,"_")
* Python: print re.sub(r"[<>]","_","din<2>")
```

---

```
--15-- rexMatchList to filter out elements from a list

> print rexMatchList("^[A-D][0-9]$"  list( "A9" "B32" "D2"))

("A9" "d2")

*Perl: print grep { /^[A-D][0-9]$/ } qw( A9 B32 D2) ;
*Ruby: print ["A9","B32","D2"].grep(/^[A-Z][0-9]$/)
*Python:[i for i in["A9","B32","D2"]if re.search(r"^[A-Z][0-9]$",i)]
```

---

```
--16-- System call to execute Perl script to convert <..> to [..]

unix> cat f1
din<2>

ciw> system( strcat( "perl -pe 's/</[/;s/>/]/' f1 > f2" ) )
```

```
unix> cat f2
din[2]

* Perl -pe 's/</[/;s/>/]/' f1 > f2
* Ruby -pe 'gsub!("<","[").gsub!(">","]")' f1 > f2
* Python: fh=open("f2","w"); for i in open("f1"):
*   fh.write(i.replace("<","[").replace(">","]") )
```

---

```
--17-- Emulate Unix: " grep hello file_in > file_out "

> system( strcat(" grep hello file_in > file_out"))
```

---

```
> fhr = infile( "/usr/quan/file_in")

> fhw = outfile( "/usr/quan/file_out")

> when( fhr
      while( gets( line fhr)
       if( rexMatchp("hello" line) fprintf( fhw "%s" line )) ) )

> close(fhr) close(fhw)

*Perl -ne 'print if /hello/' file_in > file_out
*Ruby -ne 'print if /hello/' file_in > file_out
*Python: fh=open("f2","w") ; for i in open("f1"):
*  if "hello" in i:  fh.write(i.replace("<","[").replace(">","]") )
```

---

```
--18-- Expand bus "din<3:5>" to ( "din[3]" "din[4]" "din[5]")

> str="din<3:5>"

> f=parseString( str "<:>")

> start=evalstring(nth(1 f)))
> end  =evalstring(nth(2 f)))

> for(i 0 abs(start-end)
      sprintf(newPin "%s[%d]" nth(0 f) start+i )
      printf("%L \n" newPin)
     )

"din[3]"
"din[4]"
"din[5]"

*Perl: $str="din<3:5>"; @f=split("[<:>]",$str);
*  for ($i=$f[1];$i<=$f[2];$i++) {printf("%s[%d]\n", $f[0],$i)}
*Ruby: str="din<3:5>";f=str.split(/[<:>]/)
*  f[1].upto(f[2]) { |i| print f[0]+"[#{i}]\n" }
```

```
*Python: f=re.compile(r"[<:>]+").split("din<3:5>")
*  for i in range(int(f[1]),int(f[2])+1): print "%s[%d]" %(f[0],i)
```

---

```
--19-- Sort keys in association list (sort options: 'lessp nil)

>print sortcar(list( '("A9" 45) '("C43" 56) '("B5" 23)) 'alphalessp)

(("A9" 45) ("B5" 23) ("C43" 56))

* Perl: %h=(A9=>45, C43=>56, B5=>23);
*  map { print "\($_=>$h{$_}\) " } sort{$a<=>$b} keys %h;
```

---

```
--20-- Sort value in association list

> procedure( sortcadr( a b) cadr(a) < cadr(b) )

> print sort( list( '("A9" 45) '("C43" 56) '("B5" 23) ) 'sortcadr )

(("B5" 23) ("A9" 45) ("C43" 56))

* Perl: %h=(A9=>45, C43=>56, B5=>23);
*  map { print "\($_=>$h{$_}\) " } sort{$h{$a}<=>$h{$b}} keys %h;
```

---

```
--21-- cond

> x=-5
     cond( ( (x<=0) x=abs(x) )
           (  t     x=x+2    )
           )

5

* Perl: switch: {$x<0 && do {$x=abs($x);last switch};do {$x+=2}}
* Ruby: x=-5; case x ; when -10..2,1,0: x=x.abs; else x=x-2;end
* Python: no case statement. use "if-then-else" or "try".
```

---

```
> x=5 cond( ((x<=0) x=abs(x)) (t x=x+2) )

7

* Perl: switch: {if ($x < 0) {$x=abs($x);last}; $x+=2 }
```

---

```
--22-- rplaca to replace the first element; replace "a" with "xx"

rplaca( '( a b c d) "xx") => '( "xx" b c d)

* Perl: @a=qw(a b c d); splice(@a,0,1,"xx");
```

---

```
--23-- rplacd to replace the rest of the list, except first element

rplacd( '( a b c d) list("xx") ) => '( a "xx")
```

```
* Perl: @a=qw(a b c d); splice(@a,1,$#a,"xx");
```

---

```
--24-- Get Cadence install directory name
> getInstallpath()
("/project/vendors/tools/cadence/IC5141ISR0106/tools.sun4v/dfII")
```

---

```
--25-- setInstallPath
> setInstallPath(append( list("/usr/quan") getInstallPath() )
```

---

```
--26-- Get executable SKILL path
> getSkillPath()
("." "~" "/local/skill/codes")
```

---

```
--27-- setSkillPath to set executable PATH for SKILL programs
> setSkillPath( cons( "/usr/quan" getSkillPath() )
```

---

```
--28-- Get Cadence install directory name with getShellEnvVar
> getShellEnvVar("CDS_INST_DIR")
("/project/vendors/tools/cadence/IC5141ISR0106")
```

---

```
--29-- Get full-path filename of where icfb is invoked
> simplifyFilename(getWorkingDir())
"/usr/quan"
```

---

```
--30-- alias
> alias(lf listFunctions)
> alias(h help)
> lf("ToString")                    # list all cmds with ToString
intToString timeToString
```

---

```
--31-- help
```

```
> help("stringToSymbol")

 stringToSymbol(
 t_string
 )
 => s_symbolName

 Converts a string to a symbol of the same name.
```

---

```
--32-- procedure

> procedure( add( x y )
x+y
)

> add( 5 6)
11
```

---

```
--33-- define

> ( define mult( x y )
( x*y )
)

> ( mult 5 6 )
30
```

---

```
--34-- load

unix> cat mySKill.il
print "Hello, the World"
alias(lf listFunctions)

ciw> load("mySkill.il")
"Hello, the World"
```

---

```
--35-- List Addressing

> car( '(a b c d e ))      ; return 1st item => a
> cdr( '(a b c d e ))      ; return rest of list => '(b c d e)
> car(cdr('(a b c d e))    ; return 2nd item => b
> nth( 2 '(a b c d e))     ; return 2nd item => b
> cons( 9 '(b c d ))       ; insert into a list => '(9 b c d)
> cons( 'a '(b c d ))      ; insert into a list => '(a b c d)
> cons( '(b c d) '(a) )    ; insert into a list => '((b c d) a)
> append( list(a b) '(c d) )      ; merge lists => '( a b c d)
> listA= cons( car(listA) cdr(listA)
```