VCL 1.1.4 Syntax

1 Preliminaries:

1.1 VAR-NAME:

(any letter | ‘_’) (any letter | number | ‘_’)*

VCL processor is case-sensitive.

1.2 Expression:

Name-Expression | String-Expression | Arithmetical-Expression

1.3 Arithmetical-Expression:

The syntax of arithmetic expressions in VCL is be the same as in C preprocessor except bitwise operations and shifts. (Addition, subtraction, multiplication, division, comparisons, ||, &&)

1.4 Name-Expression:

‘?’ ’@’ (VAR-NAME | ‘@’)* VAR-NAME ‘?’

*Here the first VAR-NAME don’t have to be an existing variable name, the variable will be retrieved after concatenations, the first VAR-NAME means a mixture of characters which has the same syntax rule as VAR-NAME.

A simple **Name Expression** may contain just a variable reference, such as: ?@C? or ?@@C?. (A Name Expression must be enclosed between question mark symbols ‘?’.) More complex (but more useful) Name Expressions can be written as: ?@A@B@C?. In that case, the value of such a Name Expression is computed from right to left as follows:

value-of (A <> value-of (B <> value-of (C)))

where symbol ‘<>’ means string concatenation.

Referenced variable names created at each intermediate evaluation step must represent variables that exist in processing flow, otherwise the VCL processor reports an error.

Further examples: ?@@A?, ?@A@@B@C?

1.5 String-Expression:

(STRING* Name-Expression+ STRING*) +

String Expressions may contain any number of Name Expressions intermixed with character strings. To evaluate a String Expression, we evaluate the Name Expressions from the left to the right of the String Expression, replace Name Expressions with their respective values and concatenate with character strings at the point of replacement.

Examples: ?@A@B@C?”P”?@X?, ?@@A?”B”?@C?

1.6 STRING:

“ (mixture of any characters) “

1.7 COMMENT:

‘%’ (till end of line) | ‘%>’ (mixture of any characters, multi-line) ‘<%’

2 Commands:

Examples at: http://vcl.comp.nus.edu.sg/?q=commands

2.1 Set Command:

set := #set[-defer] VAR-NAME = value (, value)*

value := Expression | VAR-NAME | STRING | INTEGER

2.2 Adapt Command:

adapt: as-is-adapt | true-adapt

as-is-adapt: #adapt[-samelevel] path

true-adapt: #adapt[-samelevel]: path

adapt-body

#endadapt

path := Expression | VAR-NAME | STRING

adapt-body := (command)*

Example: #adapt ?@nextFile?

#adapt “folder1/file.vcl”

2.3 Adapt-copy

adapt-copy: copy content | copy file

copy content: #adapt-copy path

copy file: #adapt-copy srcFile destFile

path, srcFile, destFile := Expression | VAR-NAME | STRING

2.4 Output Command:

output := #output path

path := Expression | VAR-NAME | STRING

2.5 Outdir Command:

outdir := #outdir path

path := Expression | VAR-NAME | STRING

2.6 Outfile Command:

outfile := #outfile path

path := Expression | VAR-NAME | STRING

2.7 Remove Command:

remove: #remove Expression | VAR-NAME

2.8 If Command:

If := #vcl.if condition

if-body

[#vcl.elif condition

if-body]*

[#vcl.else

if-body]

vcl.endif

condition := Expression

if-body := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.9 Select Command:

select := #select control-var

[#option-undefined

option-body

#endoption-undefined]

(#option value (| value)*

option-body

#endoption)*

[#otherwise

option-body

#endotherwise]

#endselect

control-var := Expression | VAR-NAME

value := Expression

option-body := (textual-content | String-Expression | command)*

2.10 While Command:

While := #while control-var (, control-var)*

while-body

#endwhile

control-var := Expression | VAR-NAME

while-body := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.11 Message Command:

message := short-message | extended-message

short-meesage := #message message-text

extended-message = #message:

(message-text)*

#endmessage

message-text := textual-content | Expression

2.12 Ifdef Command:

If := #vcl.ifdef VAR-NAME

  if-body

vcl.endif

if-body := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.13 Ifndef Command:

If := #vcl. ifndef VAR-NAME

  if-body

vcl.endif

if-body := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.14 Message-if Command:

Message-if := short-message-if | extended-message-if

short-meesage-if := #message-if Expression message-text

extended-message-if = #message-if: Expression

  (message-text)*

#endmessage

message-text := textual-content | Expression

2.15 Message-debug Command:

Message-debug := short-message-debug | extended-message-debug

short-meesage-debug := #message-debug message-text

extended-message-debug = #message-debug:

(message-text)*

#endmessage

message-text := textual-content | Expression

2.16 Text command (unprocessed text on output)

Text-command := #text

(text-body)

#endtext

text-body := textual-content

2.17 Insert Command:

insert := #insert[-before|-after] break-name

insert-content

#endinsert

break-name: Expression

insert-content := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.18 Break Command:

break := #break break-name

break-content

#endbreak

break-content := (textual-content | adapt | set | setloop | insert | break | output | outdir | outfile | select | if | ifdef | ifndef | message | while | remove)*

2.19 Setloop Command:

setloop: #setloop setloop-name

[setloop-vars]

(#iter iter-desc)+

[break]

#endsetloop

setloop-name := VAR-NAME

setloop-vars := #vars

(#var VAR-NAME [= value])+

[break]

#endvars

iter-desc := VAR-NAME = value (, VAR-NAME = value)*

value := Expression | VAR-NAME | STRING | INTEGER

3 VCL Processor Variables and Functions

3.1 Processor variables:

$currentfile - The currently processed file

$outfile - The currently used output file

$outdir - The currently used output directory

$defaultoutput - The default output given by command line argument or by running the processor with default values.

$debugmode - true if the VCL Processor is in debug mode. Can be set by command line argument or in VCL code

3.2 Processor funcitons:

$isFirst(<Multi-var>) - true if <Multi-var> is currently in the first iteration

$isLast(<Multi-var>) - true if <Multi-var> is currently in the last iteration

$size(<Var>) - number of elements in <Var>, if is not Multi-var the result is 1

$isNumeric(<Var>) - true if <Var>'s current value is numeric

$ulcap(<Var>) – Upper-Leading cap: changes <Var's> first character to upper case

$llcap(<Var>) – Lower-Leading cap: changes <Var's> first character to lower case

Example for the last two:

input:

```
#set task = "task"

?$ulcap(task)? ?$llcap(task)? = new ?$ulcap(task)?();
```

output:

```
Task task = new Task();
```

Remark: From VCL 1.1.4 string-expressions are not going to be evaluated in the text content.

i.e.

input:

```
#set A = 5
"t1"?@A?"t2"
```

output:

```
"t1"5"t2" // before VCL 1.1.4 -> t15t2
```

We found that, the string-expressions in the textual content are never or very rarely used. Having string-expressions allowed in textual content however introduced the need of escaping every " double quote character in the text which in programming languages is very frequently present. For these reasons, by default in textual content the processor won't evaluate string expressions in textual content, just at definitions and commands which take expressions as input. **So we don't need to escape double quote characters in the instrumented code** String expressions can be still used in textual content easily. i.e.:

input:

```
#set A = 5
#set tAt = "t1"?@A?"t2"
?@tAt?
```

output:

```
t15t2
```

For compatibility reasons, \" (escaped double quote character) **will still be accepted** and the output will be " (unescaped double quote character).

More information and examples: *http://vcl.comp.nus.edu.sg/?q=commands*

http://sourceforge.net/p/vclang/discussion/features/