Snaml for HTML5

Version 2.0

Neatware

Introduction

Snaml for HTML5 is a HTML5 generator for **Web** applications.

Hyper**T**ext **M**arkup **L**anguage (**HTML**) is a description language for web document. Berent. Lee innovated the HTML from SGML. Netscape first implemented a widely used HTML browser. Today HTML is a defacto standard for web document. The new Extensible Markup Language (**XML**) of W3 standard extended HTML to allow authors define new tags and attributes. HTML 5.0 is the latest HTML standard.

HTML5 is designed to be the presentation standard for both web and mobile document. As description languages, HTML5 is suitable to represent embedded documents. However, since HTML5 language is lack of variables, it is difficult to use HTML5 as a programming language. Therefore, HTML5 document may be not well on modularity and reusability. To do programming, HTML5 must work with JavaScript language.

Tcl is a Tool Command Language innovated by Dr. Jhon Ousterhout in the late 1980s. It is a typeless scripting language with simple and elegant syntax. It originated from unix shell command language. Because of its command, substitution and grouping syntax, Tcl is very powerful and flexible for string processing and glue components. However, because of the lack of block commands, Tcl is difficult to express the embedded documents.

Snaml for HTML5 is similar to Tcl. It added block command and inline command to integrate the Tcl with HTML5 for Web programming. Snaml code can generate an HTML5 web page and document. Snaml programs make a web site more maintainable, modifiable, and reusable espcially for mobile web applications. Another benefit of Snaml for HTML5 application is its higher level consistence. In addition, Snaml for HTML5 is much easier to connect to a database and present database content. Snaml for HTML5 is well done on the Presentation Logic on Web Applications.

Following simple **Snaml for HTML5** program shows a picture with the title '*Hello*' and output to hello.html.

# use html5 package

```
package require HTML5
```

# output to a file. If there is no output or its first parameter is null, default output is to stdout.

```
output hello.html
```

# html header. quote command render the string.

```
_html "lang='en'"
_head
  _title
    quote Hello
  title_
head_
```

# html body

```
_body
  __img "src='hello.gif' alt='hello'"
  quote "Hello World!"
body_
html_
```

Syntax

Variable, Command, Substitution, Grouping

- **Variable**

  A **variable** is a string with letters, digits, and underscores. It's case sensitive. A variable can be used anywhere without declaring its type. In addition, you may assign a value to a variable and access its value.

- **Command**

  The command-line syntax of Snaml has the form:

  command argument$_1$ argument$_2$ ...

  where command is a buildin or a defined procedure in the Snaml. Spaces separate a command name and its arguments in a line. Newline and semicolons are line or command terminators. To continue arguments on the next line, a command must have a backslash in the end of a line. Each command has a return value. To evaluate a command, interpreter substitutes command-line arguments from left to right. It makes a single passing substitution. Snaml commands are cataloged as regular, block, empty, and inline.

  - **Regular Command**

    Regular Command has a typical command-line syntax. Its number of argument is unlimited. There is no direct relation for two commands. A new command can be defined by proc command. For example, set command assigns a value to a variable.

    ```
    set var value
    ```

  - **Block Command**

    Block Command may specify a scope of a set of commands. **start command** is the beginning of a block command. Snaml specifies the satrt command name with the underscore prefix such as**'_command'**. A start command may have zero to two arguments called *attributes*. **end command** is the end of a block command. The command name has the underscore postfix such as**'command_'**. An end command has no argument. The commands between the start and end commands are called **content**.

Attribute is a list of key/value pairs separated by whitespace. A pair has the format key='value' where key must be a character string (include letters, digits, hypen, period, and underscore). The value may be any of strings or variables. The single quote ' ' groups the value. A key that is not defined as an attribute of a start command will be ignored. A non-declared key will be assigned to its default value when a start command is invocated.

Block commands may be nested. A difference between the block command and regular command is that the attributes of block commands may be inherited. The inner commands in the content of a block command may inherite the attributes of the outer block command. A command in the content is called a child of the outer block command. In contrast, the outer block command is a parent of the inner commands. Finally, the block commmands should be properly paired.

```
_font "face='verdana' size='2' color='green'"
  _b
    quote "a block command example."
  b_
font_
```

- **Empty Command**

  Empty command is a special case of the block command where the content is empty. It has a special syntax with double underscore prefix such as **'__command'**. It is convenient for a parser to check a program for empty command since there is no end command is expected. Empty command is consistent to the XML syntax such as <element/>.

  ```
  __img "src='try.gif' width='320' height='240'"
  ```

- **Inline Command**

  Inline command is a shorthand of block command. But it is limited to be a command in the square brackets and acts as an element of an argument. An inline command is represented as **'_command_'**where two underscores are prefix and postfix of a block command name. Usually it has two arguments: the first one is the attribute of the block command; the second one is the content of the block command. The content of an inline command must be a group of string, a regular or inline command. Inline command must return a value. If inline command has only one argument, this argument is considered as content rather than an attribute. To make one argument consider as attribute you must use the **inline empty**command. It has the double underscore prefix and one underscore postfix as **'__command_'**. Typically, an inline command is used inside a short text description.

  ```
  quote "a inline command [_b_ bold]"
  ```

- ## Substitution

Dollar sign $ enables variable substitution. It replaces the variable with its value. **Variable** are useful to mix with other special characters.

```
set x variable
```

Square brackets enable command substitution. A command must be inside the brackets. After evaluation of the command, the return value will replace the square bracket string. The brackets maybe nested.

```
[clock seconds]
```

Backslash enables next character substitution. The next character or group after the backslash will be replaced by a new representation of the character. Usually backslash \ is widely used to quote a specific character

```
"inside double quote \" "
"newline \n and tab \t"
```

- **Grouping**

Double quote " " enable inside string substitution. The " character inside the double quote must be disabled with the backslash \" quoting. The grouping value of double quotes is the string inside after substitution.

```
dobule quote enable variable substitution {$variable}
and command substitution [clock seconds]
```

Curly braces {} disable substitution inside. All the characters include whitespace, double quotes, even nested curly braces (exclude outmost curly braces) are value of the group. When curly braces are inside the double quote, they will not work as grouping.

```
curly braces disable variable substitution {$variable}
and command substitution [clock seconds]
```

Control

Sequence, Condition, Loop, Return, Exception

1. Sequence

Sequence control organizes the command flow in sequence. Each commands is executed one after another. In Snaml all the block commands for HTML and XML are sequence. Following paragraphs introduce primitive sequence commands. They are also the Tcl commands.

- ### set

set command assigns value to a variable. Its first argument is a variable and its second argument is an expression. You can use a variable anywhere without declare its type. A variable with dollar sign $ returns its value

```
set v "a string value"
set u $v
```

- ### incr

The first argument of **incr** command is a variable name. The second optional argument is increment integer. incr command increases (decreases) variable value by increment. The default value of increment is 1.

```
incr x                # x increase 1
incr x -1             # x decrease 1
incr x [expr 2*3]     # x increment
```

- ### expr

expr command evaluates a math expression with C expression syntax. It concatenates all the arguments to form a input string. The expression may be integer, floating point, and boolean. The return value is a numeric string. Many basic math functions in the standard C math library have been built in.

```
expr 2/3                     # return 0 (why?)
expr 2.0/3.0                 # 0.666666666667
expr asin(1.0)*round(sqrt(2)) # 1.579079632679
```

- ### # comment

comment in the Snaml starts with **#** in the start of a line. You'd better use comment # as a command. There are some quirks prohibited you from writing comment anywhere. (i.e. no inside the switch command.

```
set x 6; # comment here. ; new command
# comment start
```

- **quote**

quote command outputs its argument string to a IO channel. The IO channel is defined by __output command. It maybe a file or stdout. It is a special start command with only one argument and no tags.

```
quote "output a $string and a [compute $value]"
```

2. Condition

Condition control may select a command to execute according to the variable value. It branches the command flow.

- **if**

if command will execute truebody when the expression is true, otherwise it will execute elsebody. Its body is a group of commands. The else and elseif keyword are optional for if command. Following codes are several forms of if command

```
if {expression} {truebody}
if {expression} {truebody} else {elsebody}
if {expression1} {
  truebody1
} elseif {expression2} {
  truebody2
} else {
  elsebody
}
```

- **switch**

The syntax of **switch** command is:

```
switch option value pattern body ...
switch option value {pattern body ...}
```

switch command compares a value with patterns. If one of them is matched then program executes the related body. The first argument of the switch command is an option. The **'-exact'**attribute will match the value to the pattern exactly; **'-glob'**attribute will use glob pattern matching; and **'-regexp'** will match with regular expression pattern. **'--'** represents the end of the option. The last pattern **'default'** will execute its body if no patterns are matched before.

In the following example, since pattern and body pairs are grouped into an argument, there is no substitution inside the pattern/body pairs.

```
switch -exact -- $val {
  first {doFirst}
  second {doSecond}
  third {doThird}
  default {doDefault}
}
```

following example can substitute its patterns

```
switch -glob -- $val  $v1 do_v1  $v2 do_v2  $v3 do_v3
```

3. Loop

Loop commands execute a group of commands in iteration. The iteration may terminate after all the elements are traversed or a condition expression becomes true.

- ### foreach

foreach command repeatly executes its body until all the elements in a list have been traversed. Its form is,

```
foreach var alist ... body
```

The var is the current loop variable that is assigned an element from the alist one after another. **foreach** will traverse all the elements in the alist. This command is a compact expression of iteration.

```
foreach v {a b c d e} {
  quote $v
}
```

You can declare two or more loop variables. The variables will orderly sample the elements in the list until all of them are traversed. Following example shows that varaiable (v1 v2) pair is assigned the value (a b) respectively, and then the value (c d), and so on.

```
foreach {v1 v2} {a b c d e f} {
  quote "($v1 $v2)"
}
```

To loop over multiple lists, you may organize arguments in var/list pair order. The variable var may also be multiple variables. A loop variable will be set to empty {} when its list has finished traverse but the entire loop did not terminated.

```
foreach v {a b c d} {v1 v2} {1 2 3 4 5 6} {
  quote "($v) ($v1 $v2)"
}
```

- **while**

  while command evaluates the *expression*, if it is true then executes the *body*, and then evaluates the *expression* again until the*expression* is not true. Its syntax is like while statement of C language.

  ```
  while expression body
  ```

  An example of while command is,

  ```
  set count 7
  while {$count > 0} {
    quote "2*$count"
    incr count -1
  }
  ```

- **for**

  The **for** command syntax is,

  ```
  for initial expression increment body
  ```

  At first it evaluates the *initail* argument and then evaluates the*expression*. If the *expression* is true it executes the *body* and*increment*. Repeatlly, it evaluates the *expression* again and continues the loop until *expression* returns a false value.

  ```
  set len 7
  for {set count 0} {$count < $len} {incr count 1} {
    quote "2*$count"
  }
  ```

4. Return

return command comes back from a procedure with a value; **break**command exits from a loop; and **continue** command will goto the start of a loop to execute the next iteration.

```
set b 6
set c 5
set len 7
for {set count 0} {$count < $len} {incr count 1} {
  if {$count == $b} {
    break
  } elseif {$count == $c} {
    continue
  }
}
return $c
```

5. Exception

Exceptions raise abnormal conditions during the execution of commands.

- ### Catch

catch command caught the exceptions of a command during its execution. Its syntax is,

```
catch {command args ... } result
```

catch command sets trap to the command in the curly braces. When there is an exception during the command execution, the exception message is assigned to the variable 'result', otherwise the 'result' gets the return value of the command. catch command returns zero when no exception is raised, otherwise returns non-zero.

```
if [catch {test $exception} result] {
  quote "Exception: $result"
} else {
  quote "OK: $result"
}
```

- ### Error

error command generates an error code. Its first argument is a string that indicates the reason of an error

```
catch {...} errmsg
error $errmsg
```

Types

1. String

String is a primitive object of MCL. It consists of any characters. Some characters may have special meaning for a string in the context. String commands consists of a group of subcommands.

String Construction

(i) **string append VAR STRING1 STRING2 ...** command concatenates STRING1, STRING2, ... onto the variable VAR and returns the new value of the variable VAR; (ii)**join LIST STRING** command joins the elements of LIST together and distinguishes them with a STRING. The default STRING is a space; (iii) **split STRING CHAR** command splits STRING with the CHAR.

```
 set v "one "                 # $v is "one "
string append v "two"        # new $v is "one two"

set l {a {b c} d}            # $l is {a {b c} d}
set r [join l "::"]          # $l is "a::b c::d"
set s [split r "::"]         # split r with '::'
```

String Access

(i) **'string length STRING'** command returns the number of STRING characters; (ii) **'string range STRING i j'** command returns the substring of the STRING from i to j; (iii) **'string index STRING i'** returns the character in the position i. A string has a zero based position. (iv) to find the occurrence of a string, **'string first STRING ELEMENT'** command returns the first occurrence of ELEMENT in the STRING, no finding return -1; (v) **'string last STRING ELEMENT'** command finds the last ELEMENT occurrence in the STRING, no finding return -1.

```
 set s "abc defe"
set n [string length $s]      # n is 8
set r [string range $s 1 5]   # r is "bc de"
set i [string index $s 0]     # i is "a"
set f [string first "ef" $s]  # f is 5
set t [string last "e" $s]    # t is 7
```

String Operation

(i) **'string compare STRING1 STRING2'** compares two strings. It returens 0 if they are equal, or -1 if STRING1 is less than the STRING2, otherwise +1; (ii) to find a string in a pattern, **'string match STRING PATTERN'** command completes pattern matching, that is the STRING matches PATTERN. PATTERN may be the combination of characters and the special matching characters where * for any characters, ? for any single character, and [xyz] for one of a character in the [ ]. If STRING matches the PATTERN then it returns 1, otherwise it returns 0. (iii) **'string tolower STRING'** and **'string toupper STRING'** will convert the STRING to lower and upper case respectively. Examples of string compare are:

```
 set s "abc "
set r [string compare $s "abc"]
if {$r == 0} {
  puts "s == 'abc'"
} elseif {$r == -1} {
  puts "s < 'abc'"
} else {
  puts "s > 'abc'"
}
```

string match example:

```
 if {[string match $s {a?[xyz]}] == 0} {
  puts "matched"
}
```

string conversion example:

```
set l [string tolower $s]
set u [string toupper $s]
```

String Format

'string format STRING VAR$_1$ VAR$_2$ ...' command is similar to printf() function of C. It returns a formatting string. STRING is the format specification. VAR$_1$, VAR$_2$ ... are corresponding values.)

```
 set s "32"
set d [::string::format "%2d %lf" $s $s]
```

2. List

List is an order set of items.

List Construction

(i) **'list a1 a2 ...'** command constructs a list from its arguments a1, a2, ... The curly brace {} represents empty list; (ii) **'lappend LIST a1 a2 ...'**command appends arguments a1, a2, ... to the the end of the LIST as elements; (iii) to merger lists, **'lconcat LIST1 LIST2 ...'** joins the elements in LIST1, LIST2, ... together to form a new list.

```
# lx is a list {a b c {d e}}
set lx [list a b c {d e}]

# ly is a new list {a b c {d e} {g h}}
set ly [lappend $l {g h}]

# lxy is the join of lx and ly
set lxy [lconcat $lx $ly]
```

List Access

'llength LIST' returns the number of elements in the LIST; (ii) to get a sub-list, **'lrange LIST i j'** command returns elements of LIST from i to j; finally, (iii) **'lindex LIST i'** returns the ith element of the LIST.

```
set l {a b {c d}}
set n [llength $l];           # n is 3
set e [lrange $l 0 1]; # e is {a b}
set i [lindex $l 2];  # i is {c d}
```

List Operation

'lsearch LIST VAR [OPTION]' returns the index of LIST that matches the VAR value in one of an OPTION or return -1 if no value is found. -glob, -exact, and -regexp are possible OPTION values. The default OPTION value is -exact.

(i) to add a new element into a list, **'linsert LIST i a1 a2 ...'** command inserts elements a1, a2, ... before the index i of the list LIST. It returns the new list; (ii) to modify elements, **'lreplace LIST i j a1 a2 ...'** command replaces elements from i to j in LIST by elements a1, a2, ... and returns the new list; (iii) **'lsort LIST [OPTION]'** sorts elements in LIST according to one or more OPTION values (-ascii, -integer, -real, -dictionary, -increasing, -decreasing, -command, -index i). The default options are '-ascii -increasing'. It returns the new list.

```
 set l {a b {c d}}

# n is 1
set n [lsearch $l "b" -exact]

# v is {a f b {c d}}
set v [linsert $l 1 "f"]

# u is {g f b {c d}}
set u [lreplace $l 0 0 "g"]

# s is {b {c d} f g}
set s [lsort $l {-increasing -ascii}]
```

3. Array

In Snaml for HTML5 an array is acturally an associate array rather than a traditional array. It is a collection of key/value pairs. The key is a index and the value is an element of an array. An element of array 'a' with index 'key' is represented as a(key). Its value is $a(key). An array is implemented as a hash table.

Array Construction

'array names ARRAY [PATTERN]' command returns the list of ARRAY keys that match the PATTERN. If no PATTERN item it returns the list of all the keys of the ARRAY.

```
set a(x) "abc"
set a(y) "def"
set l [array names a]   # $l is {x y}
```

Array Access

(i) **'array exists ARRAY'** returns 1 if ARRAY is an array variable, otherwise it returns 0; (ii) **'array size ARRAY'** returns the number of elements of ARRAY.

```
if {[array exists a] == 1} {
    puts "a is an array"
    set n [array size a]
  } else {
    puts "a is not an array"
    set n 0
  }
```

Array Operation

(i) **'array get ARRAY [PATTERN]'** returns a key/value pair list. PATTERN is used for matching keys. Without PATTERN 'array get' command will return all the pairs; (ii) **'array set ARRAY LIST'** command sets ARRAY with the LIST in the key/value form.

```
set a(x) "abc"
set a(y) "def"
set l [array get a]; # l is a list {x abc y def}
array set m $l; # m is an array same as a
```

4. File

File commands are divided inot directory and file operations.

Directory Status

(i) **'file dirname name'** returns a directory name in a path. If name is a relative file name and only contains one path element, then returns ``.''. If name refers to a root directory, then the root directory is returned. (ii) **'file tail name'** returns the name after the last directory separator. If name contains no separators then returns name itself (iii) **'file isdirectory name'**returns 1 if file name is a directory, otherwise returns 0. (iv) **'file mkdir dir1 dir2 ...'** creates one or more directories. For each pathname dir specified, this command will create all non-existing parent directories as well as dir itself. If a directory exists, then no action is taken and no error is returned. Trying to overwrite an existing file with a directory will result in an error. dir arguments are processed in the order specified, halting at the first error, if any.

```
file dirname ~/src/foo.c   # returns ~/src
file tail ~/src/foo.c      # returns foo.c
file mkdir src             # create src directory
```

File Status

(i) **'file size name'** returns the file size;

(ii) **'file atime name'** returns a decimal string giving the time at which file name was last accessed. The time is measured in the standard POSIX fashion as seconds from a fixed starting time. If the file doesn't exist or its access time cannot be queried then an error is generated;

(iii) **'file stat name varname'** invokes the stat kernel call on name, and uses the variable given by varname to hold information returned from the kernel call. varname is treated as an array variable, and the following elements of that variable are set: atime, ctime, dev, gid, ino, mode, mtime, nlink, size, type, uid. Each element except type is a decimal string with the value of the corresponding field from the stat return structure; see the manual entry for stat for details on the meanings of the values. The type element gives the type of the file in the same form returned by the command file type. This command returns an empty string;

(iv) **'file attributes name [option]'** this subcommand returns a list of the platform specific flags and their values. The 'file attributes name [option value ...] sets one or more of the values. The values are as follows:

On Windows, -archive gives the value or sets or clears the archive attribute of the file. -hidden gives the value or sets or clears the hidden attribute of the file. -longname will expand each path element to its long version. This attribute cannot be set. -readonly gives the value or sets or clears the readonly attribute of the file. -shortname gives a string where every path element is replaced with its short (8.3) version of the name. This attribute cannot be set. -system gives or sets or clears the value of the system attribute of the file.

```
  set s [file atime filename]
  set len [file size filename]
```

File Operation

(i) **'file copy source target'** copies source file to the target file or directory; (ii) **'file delete pathname [-force]'** remove files and directories. -force option will delete the pathname in force.; (iii) **'file rename source target'**rename source file name to the target; (iv) **'file join name [name ...]'** takes one or more file names and combines them, using the correct path separator for the current platform; (v) **'file split name'** returns a list whose elements are the path components in name. The first element of the list will have the same path type as name. All other elements will be relative.

```
file copy src.sml dest.sml
```

Module

Procedure, Package, Namespace

1. Procedure

Snaml for Tcl constructs modules with procedure, namespace, and package. A procedure defines a new command with the combination of existed commands; a namespace distinguishes a command name; and package provides a method to use source or binary codes without specifying their locations.

- ### proc

The syntax of proc command is

proc name argument body

The first argument **'name'** of proc is the procedure name. **'argument'** is a list of arguments of the defined command. An element of the**'argument'** can be a string or a list of pairs. A string is the argument name without default value. A pair consists of an arugment name and its default value. The **'body'** specifies the command sequence that implements the function of a procedure. All the variables except of the global variables in the body have a local scope. The return value of the last command is the return value of the proc. Typically, a **'return'**command in a proc can return a value directly.

```
proc distance {x, y, {a 0} {b 0}} {
  set xa [expr $x-$a]
  set yb [expr $y-$b]
  return [expr sqrt($xa*$xa + $yb*$yb)]
}

set d [distance $a1 $b1 $a2 $b2]
```

- ### global

Global scope is the top level scope. In the global scope a variable is a global variable as default. A **'global v1 v2 ...'** command declares variables as global variables in a proc. Namespace prefix :: of a variable also makes a variable access as a global variable.

```
global v1 v2
set v ::v1
```

- **upvar**

upvar command passes the name of a variable into a proc. It refers a local variable to a variable outer one level of the scope. Its syntas is

upvar var1 localvar1 [var2 localvar2] ... .

To pass an array name into a proc, upvar command may refer to the array with a local variable. For example.

```
proc iteration {aName} {
  upvar $aName a

  foreach index [array get a] {
    _quote "a[$index] = $a[$index]"
  }
}
```

2. Namespace

Namespace specifies a new scope for global variables and procedures. It minimizes the naming conflict. Namespace is a mechanism to organize large Snaml programs. Its declaration is represented as

```
# namespace declaration
namespace eval name {
  variable var value ...
  namespace export proc1 proc2 ...
}

# procedure declaration
proc name::proc1 {args} {
  variable var
  commands ...
}
...

# procedure declaration
proc name::procn {args} {
  commands ...
}
```

'namespace eval name' specifies the name of a namespace. In the namespace specification, keyword **'variable'** declares the variable 'var' and its initial value 'value'.

The **'namespace export'** command declares the procedure names that will be available for invocation. Outside the **'namespace eval'**declaration the proc specifies a procedure of the namespace. The procedure name consists a namespace prefix name linked by **::** with a procedure name and its arguments. Local namespace variable must be declared by variable command.

It is possible to define a namespace with the full qualified name rather than relative qualified name. Global prefix :: must be added to the namespace name. We suggest namespace name and proc name starts with the capital letter, variable name starts in the lowercase letter. These naming convenition will make code readable.

```
Network::Protocol "TCP/IP"
::Network::Protocol "TCP/IP"
```

3. Package

Package organizes a libray of programs. Snaml uses the facility of Tcl to extend its functions for component programming.

- **source**

'source filename' command will evaluate the commands in the filname. Its return value is the value of the last command in the filename.*filename* is the location of the file. It may be a relative or absolute path. Source command is limited to load a text file and the file location is dependent on the platforms.

```
source "html.sal"
```

- **load**

'load filename' command will load an binary code into the program and calls an initialization procedure in the extension. The filename varies from platforms such as .so in Sun and .dll in Windows. The extension must obey Tcl extension rules in order to add them as new commands.

```
load xml.dll
```

- **package**

Package command provides a facility to group a set of commands. To setup a package each library must declare a **'package providepkgname pkgver'** in its file. The 'pkgname' is the package name. 'pkgver' is the version number of the package with the format 'major.minor'. Same major number expresses the interfaces of the packages are compatible. Different minor number may have different implementation. Usually a package should keep backward compatibility. That is the package with bigger major number will work for the package with smaller major number. A package may be distributed on several files by specifing the identical **'package provide'** command.

```
# in the library file
package provide html 4.0
```

To use a package a **'package require'** command must be declared in a program. The syntax is **'package require pkgname [pkgver]'**. Without the pkgver argument the hightest version of the package is loaded. If there is no suitable version of package available, 'package require' command will raise an error. To create a package, you need to do manually package installation.

1) You need to create a package file with namespace or procedures and add**'package provide'** command in the file.

2) You may put the package file to a subdirectory. Then you need add a command 'lappend ::auto_path subdirectory' in the beginning of your code. With this command, the package will automatically search the files in the auto_path and its subdirectories.

3) In addition, you must execute a command 'pkg_makIndex sudirectory name1.tcl name2.dll' to generate pkgIndex.tcl file.

```
# package require command
package require html5
```

Status

1. Info

- **info exist varname**

if variable varname exists in the current context then returns 1, otherwise returns 0.

- **info path**

return current processing file name.

- **info globals**

return a list of global variables.

- **info locals**

return a list of all the names of currently-defined local variables, including arguments to the current procedure. Variables defined with the global and upvar commands will not be returned.

- **info library**

return the path name of the library.

2. Clock

commands in the clock manipulate time and date.

- **clock clicks**

return a high-resolution time value as a system-dependent integer value. The unit of the value is system-dependent but should be the highest resolution clock available on the system such as a CPU cycle counter. This value should only be used for the relative measurement of elapsed time.

- **clock seconds**

return the current date and time as a system-dependent integer value. The unit of the value is seconds.

- **clock format clockvalue [options]**

converts an integer time value, typically returned by clock seconds, clock scan options, to human-readable form. The options maybe format=string that describes how the date and time are to be formatted. Field descriptors consist of a % followed by a field descriptor character. All other characters are copied into the result. Valid field descriptors are: converts an integer time value, typically returned by clock seconds, clock scan options, to human-readable form.

Without the format=string option, the format string "%a %b %d %H:%M: %S %Z %Y" is used. The gmt=true specifies that the time will be formatted as Greenwich Mean Time. The false value is the local timezone defined by the operating environment and it is a default value.

Valid field descriptors are:

```
%%  Insert a %.

%a  Abbreviated weekday name (Mon, Tue, etc.).
%A  Full weekday name (Monday, Tuesday, etc.).
%b  Abbreviated month name (Jan, Feb, etc.).
%B  Full month name.

%c  Locale specific date and time.

%d  Day of month (01 - 31).
%H  Hour in 24-hour format (00 - 23).
%I  Hour in 12-hour format (00 - 12).
%j  Day of year (001 - 366).
%m  Month number (01 - 12).
%M  Minute (00 - 59).
%p  AM/PM indicator.
%S  Seconds (00 - 59).
%U  Week of year (01 - 52), Sunday is
   the first day of the week.
%w  Weekday number (Sunday = 0).
%W  Week of year (01 - 52), Monday is
   the first day of the week.
```

```
%x  Locale specific date format.
%X  Locale specific time format.
%y  Year without century (00 - 99).
%Y  Year with century (e.g. 1990)

%Z  Time zone name.
```

HTML5

1. **Structure**

HTML5 is HyperText Markup Language 5, a standard for Web document. HTML5 modularization decomposes HTML5 into a collection of abstract modules. These modules may be combined to create an HTML5 subset or extension. Content developers can use the subset of HTML5 to fit different platforms such as mobile devices, game consoles, and appliances.

Element and its attributes are basic components of HTML5. There are three types of element: **start** tag, **end** tag, and **empty** tag. The start and end tag has the format <element attribute> and </element> respectively. The text between the start tag and the end tag is called**content**. An element may have no end tag. An **empty element** has neither end tag nor content. An element name is always case-insensitive.

Attributes are properties of an element. The attribute='value' pairs are attached behind the start element name and inside the < >. Any number of attribute value pairs may be attached in arbitrary order. The attributes that are not used in the start tag are set to be default values. Boolean attribute may only have the value. Usually, double quotation mark " " and single quotation mark ' ' group the value. We prefer to use single quote ' ' as value grouping. To specify attribute name it is better to use letters, digits, hyphens and periods. Both the attribute name and value are case insensitive.

In Snaml for Tcl, **_element** and **element_** command with underscore in the head and end represent the start tag and the end tag. One argument of the start element command is its attribute. The empty element is represented as double underline element (**__element**). It specifies that no content and end tag are expected. For example,

```
package require HTML5
output hello.html

_html
  _head;
    _title; quote "Title"; title_
  head_
  _body
    quote "Document Content"
  body_
```

```
    html_
```

where "package require HTML5" use the HTML5 package. "output" command writes the generated HTML5 to .html file. **_html** specifies the xml namespace and language. html_ specifies the end of html. **_head**command specifies the human readable and machine readable head information of an HTML5 document. The **_title** command must be included inside the _head content.

Core

Core module includes Structure, Meta, Text, Hypertext, and List modules.

1. Structure Module

Structure Module includes **html**, **head**, **title** and **body** elements. _html specifies the xml namespace and language. html_ specifies the end of xhtml. For example,

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
html_
```

_head command specifies the human readable and machine readable head information of an XHTML document. The **_title** command must appear inside the **_head** content. Other commands are optional. For example,

```
_head
  _title "title='Document'"
    quote "Title"
  title_
head_
```

A browser will display the value of the title attribute such as title='Document' on the title bar. Typically, 'title' attribute is used as a tooltip of a command.

_body command specifies the content of an XHTML document. It has the COMMON attribute set. Style sheets are now the preferred way to describe the presentation.

```
_body
  quote "XHTML content"
body_
```

2. Meta Module

__meta command sets machine readable information. Its **'name'** attribute defines the property name; **'content'** attribute defines the property value;**'scheme'** attribute interprets the property value; **'lang'** attribute specifies the language; and **'http-equiv'** attribute provides HTTP server information for response header.

```
__meta "name='author' content='David Robert' scheme='ISBN Author'"
__meta "http-equiv='Expires' content='Fri, 25 Dec 1998'"
```

http-equiv may be used to wait seconds and refresh to another page. Following example waiting 2 seconds and switch to new URL in content.

```
__meta "http-equiv='refresh' content='2, http://www.neatware.com'"
```

When the name's value is a keyword, it is helpful to list a series of keywords in the content's value for search engine readable.

```
__meta "name='keywords' lang='en-us' content='Snaml Generic'"
```

3. Text Module

Text Module consists of h1-h6 for heading, div, p, and pre for block, span, br, em, and strong for inline. Other deprciated commands are abbr, acronym, address, cite, code, dfn, kbd, samp, var, q, and blockquote. Developers should avoid to use the deprciated commands. Refer to Text Module for more details.

4. Hypertext Module

Hypertext Module includes the **_a** command. Refer to Link Module for more details.

5. List Module

List Module includes ul, ol, li, dl, dt, and dd elements. Refer to **List Module** for more details.

Text

Text Module consists of h1-h6 for heading, div, p, and pre for block, span, br, em, and strong for inline. Other deprciated commands are abbr, acronym, address, cite, code, dfn, kbd, samp, var, q, and blockquote. Developers should avoid to use the deprciated commands.

- **Whitespace**

Whitespaces are *space*, *tab*, *form feed* (), and *zero-width space* (). The carriage return (CR) and line feed (LF) are also whitespaces in the XHTML document. Whitespace characters will have no visual formatting effects. A number of whitespace will only compact to one space.

- **Heading**

_h1 to **_h6** are heading commands to describe the topic of a section. There are six levels. _h1 is the highest level (usually larger font) and _h6 is the lowest level. You need to set their attributes in the style sheet.

```
_body
  _h1; quote "H1 Title"; h1_
  _h2; quote "H2 Title"; h2_
body_
```

- **Block**

The **_div** command offers a generic mechanism to construct structure documents. _div defines block content. By using **'class'** and **'id'** attributes, authors may easily control the content in the block for presentation. _div along with inline block_span commands are prefered commands to separate structure and presentation. For example,

```
_div "id='David' class='client'"
_span "class='title'"
  quote "Client information:"
span_
_table "class='data'"
  _tr
    _th; quote "Last name:"; th_
    _td; quote "David"; td_
  tr_
  _tr
    _th; quote "First name:"; th_
    _td; quote "Robert"; td_
  tr_
  _tr
    _th; quote "Tel:"; th_
    _td; quote "(905) 534-2812"; td_
  tr_
  _tr
    _th; quote "Email:"; th_
    _td; quote "robertd@neatware.com"; td_
  tr_
table_
div_
```

- **Paragraph**

_p command defines a paragraph. It is also used as a newline in the Snaml. Its**'clear'** attribute specifies the property that floats around another object. **__br**command forces a line break. The character **' '** is a real space.

Plain hyphen is just a regular character '-'. Usually a browser did not display soft hyphen if a line is not broken at a soft hyphen, otherwise it displays the '-' in the end of a line.

_pre command defines preformatted text. It will render the text content of the _pre command "as is". All the newline and space characters will be kept.

```
_p "clear=right"
  quote "float paragraph"
  __br
  quote "in the document."
p_

_pre
  quote "preformatted text with space and newline."
pre_
```

- **Inline Block**

The **_span** command separates the small text inline block. **_br** is a new line command. Usually it is used with an empty command **__br**. **em** and **strong**are emphasis and strong commonds.

The **_em** and **_strong** are two most common commands to make structural text. _em is emphasis (default italic) and _strong is strong emphasis (default bold).

```
_em; quote "The emphasis is italic."; em_
_strong; quote "The strong is blod."; strong_
```

- **Others**

Other commands work for special purposes. The **_cite** command is a citation or reference to other source; **_dfn** command is the defining instance of the enclosed term; **_code** command is used to represent computer code; and**_samp** command is for the sample output. In addition, **_var** command marks the instance of a variable; **_abbr** command tags abbreviated form;**_acronym**_kbd command represents the text to be entered by user. Finally,**_sup** and **_sub** mark the text as superscript and subscript.

```
_cite; quote "citation"; cite_
_dfn; quote "defining instance"; dfn_

_code; quote "computer code"; code_
_samp; quote "sample output"; samp_
_var; quote "variable"; var_
_abbr; quote "abbreviated form"; abbr_

quote [_acronym_ "acronym"]
quote [_kbd_ "entered text"]
quote [_sub_ "subscript"]
quote [_sup_ "superscript"]
```

_blockquote and **_q** command are for long and short quotation respectively. Browsers generally render _blockquote as an indented block. _q command generally shows text with delimiting quotation marks.

```
_blockquote "id='group'"
  quote "block words."
blockquote_
_q; quote "quote words"; q_
```

Finally, **_ins** and **_del** commands mark the sections of a document that has been inserted or deleted. They are rarely used.

Font

In the HTML5 you'd better use CSS instead of _*font* command.

1. **Formatting**

Typically, **'bgcolor'** attribute sets the background color with a color value.**'align'** attribute aligns block elements such as tables, objects, and paragraphs on the canavas. The possible values are the left, center, right, and justify. For left-to-right text and right-to-left text, the default values are left and right respectively. An object will float on the left or right margin because of its **'align'** attribute. The **'clear'** attribute of _br command controls text flow around floating objects. Its value 'none' (default) starts next line normally. The value 'left' and 'right' will make next line begin at nearest line below any floating objects on the left and right margin. The value **'all'** is for either margin.

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
_head
  _title; quote "Formatting"; title_
  _style "type='text/css'"
    _comment
      rule "br#test" {clear: left}
    comment_
  style_
head_
_body "bgcolor='white'"
  quote "long text paragraph"
  _br "id='test'"
  quote "is separated."
body_
html_
```

2. **Font**

Although font command is depreciated in favor of style sheets, they have been widely used in the current web pages. Here, **_tt** command renders text as teletype or monospaced; **_i** command renders text as italic; and **_b**command renders text as bold. I addition, **_small** and **_big** commands render text in a small and big font respectively. Finally, **_strike** and **_u**commands render text as strike-through and underline.

Rather than uses font command, style sheet is clarity to achieve visual effects. Following example shows green and italic text in a paragraph with style sheet.

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
_head
  _title; quote "Font"; title_
  _style
    _comment
      rule "p.igreen" {font-style: italic; color: green}
    comment_
  style_
head_
_body
  _p "id='igreen'"
  quote "Green italic text by style sheet."
  p_

  # the font equivalent
  _font "color='green'"
    _i; quote "Green italic text by font."; i_
  font_
body_
html_
```

_font command changes the font size and color of the text in its content. The **_basefont** command sets the base font size with **'size'** attribute. Typical default base font size is 3. The **'size'** attribute must be an integer from 1 to 7. The '+' and '-' sign of an integer means relative increasement and decreasement in font size. To declare font names, **'face'** attribute must specify a comma-separated list. In addition, **'color'** attribute specifies the text color.

```
_font "face='arial, helverta' size='+2' color='red'"
  quote "This is red arial text with size 2."
font_
```

_hr command shows a horizontal rule. Boolean **'noshade'** attribute will render rule in a solid color rather than a groove. Furthermore, **'size'**attribute specifies the height of the rule; **'width'** attribute specifies its width (default 100%); and **'align'** attribute specifies its horizonal alignment.

```
_hr "align='center' width='50%' size='3' noshade"
```

Link

Anchor, Link, Base

1. Anchor

A **link** is a connection from one web resource to another. **Anchor** is the end of a link. A link pointers from source anchor to destination anchor. Users may visit the destination resource by activating a link (e.g. clicking mouse button). The destination anchor may be a resource on the Internet or an element with the id on the XHTML document.

_a command declares a link in the **Snaml**. It can only appear as the content of the _body. The content of **_a** command is a source anchor. A browser will display the content as an underline text or an image in normal. By clicking the text user can switch to the destination anchor that is specified by **'href'** attribute with a URL. The **'id'** attribute of **_a**command declares its anchor. Other links may refer to it with **#idvalue**. Any block commands may specify **'id'** attribute for its anchor. The **'title'**attribute is used to display an anchor's tooltip. The attribute of _a command can belong to the COMMON attribute set and be other attributes: accesskey, charset, href, hreflang, rel, rev, tabindex, and type. You can use accesskey attribute to define a key shortcut.

```
_a "id='token' href='http://www.neatware.com/'"
  quote "Anchor to href URL with Name token"
a_

_a "href='#token' title='here is the token'"
  quote "Anchor to token in the document"
a_

_h1 "id='token2'"
  quote "Destination anchor with id"
h1_
```

2. Link

__link command defines a relationship between current document and other resources. It may appear in the _head and _body of a Snaml program. __link command may describe the position of a document within a series of documents. The **'rel'** (relationship) and **'rev'** (reverse) attributes are used for this purpose. __link command may also refer to the external style sheets.

```
_head
  _title; quote "Chapter 2"; title_; # current
  __link "rel='prev' href='chapter1.html'"; # forward
  __link "rev='next' href='chapter3.html'"; # backward
head_
```

3. Base

URL (**U**niversal **R**esource **L**ocator) may be absolute, relative, and internal. Absolute URL has the full path. Relative URL has only relative path that will be resolved according to the base address. The internal URL in the document starts with the '#' character followed by a string that is an id value of a command or a name value of _a command.

_base command specifies a document's URL explicitly. Its default value is the current document.

```
absolute URL:   http://www.neatware.com/dest.html#one
relative URL:   dest.html#one
internal URL:   #one
```

List

Firstly, **_ol** (order list) command creates a numbered list. Secondly, **_ul**(unorder list) creates a bulleted list. In their contents, **_li** represents the marker of an order and unorder list. Finally, **_dl** creates a definition list where**_dt** (definition title) and **_dd** (definition description) are markers for the list elements.

Lists may be nested and combined with different types. The _ol only **'start'**attribute specifies the starting marker of the first item in an ordered list (default 1). The **'type'** attribute of _ol may have the value '1', 'a', 'A', 'i', 'I' for arabic numbers, lower alpha, upper alpha, lower roman, and upper roman markers respectively. However, the **'type'** attribute in _ul sets the shape of markers with value disc, square, and circle. In addition, the _li only **'value'**attribute sets the number of the current list item.

unorder list

```
_ul "type='circle'"; # shape of marker
  _li; quote "First Item" li_
  _li; quote "Second Item" li_
  _li; quote "Third Item" li_
ul_
```

order list

```
_ol "start=2 type='i'"
  # starting marker 2 with lower roman
  _li "value='1'"; _quote "1st Item"
  # current item
  _li; quote "2nd Item" li_
  _li; quote "3rd Item" li_
ol_
```

definition list

```
_dl
  _dt; quote "1st Title" dt_
  _dd; quote "1st Content" dd_
  _dt; quote "2nd Title" dt_
  _dd; quote "2nd Content" dd_
  _dt; quote "3rd Title" dt_
  _dd; quote "3rd Content" dd_
dl_
```

Objects

Objects, Applets, Images

1.**Objects**

Objects are things in an XHTML document. Images, applets, and plugins are instances of objects. **_object** command describes the object in general. The attributes 'classid', 'codebase', and 'codetype' specify the object code. To specify the location of an object, **'classid'** attribute has a URL value. In addition, **'codebase'** attribute defines the base path to resolve the relative URL in the classid. To specify the content type of a media, **'codetype'** has the MIME format such as 'application/mpeg'.

The 'data' and 'type' attributes also work for object data. The **'data'**attribute defines the location of an object's data. The **'type'** attribute specifies the data type. To describe the relevant resources of an object,**'archive'** is a urllist. Finally, boolean **'declare'** attribute specifies that the object is only a declaration and does not execute during loading.

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
  _head
    _title; quote "Object"; title_
  head_
  _object "title='space' classid='space.sml'"
    quote "Space"
  object_
html_
```

- **Render Object**

_object command specifies how to render an object data. The object's code, data, and parameters may be required to specify an objcet. The _object's content is an alternation when the object is not available. Following example demonstrates the rendering of a Snaml animation. If this Tcl animation object is not available it renders a mpeg video, otherwise it shows a gif picture.

```
_object "title='space' classid='space.sml'
  _object "data='space.mpg'
        type='application/mpeg'"
    _object "data='space.gif' type='image/gif'"
      quote "Space Animation"
    objcet_
  object_
object_
```

- **Initialize Object**

_param command specifies a set of runtime values of an object. Any number of the _param commands in arbitrary order may be the content of an object. Its **'name'** attribute specifies a runtime parameter that is known by object's implementation. Its **'value'**attribute is the value of the runtime parameter. In addition, its**'valuetype'** may be data (default), ref, and object. Where data is a string, ref refers to a location of parameters, and object refers to another object in the same document. **'type'** is the 'content-type'.

```
_object "classid='space.tcl'"
  _param "name='width' value='320' valuetype='data'
  _param "name='height' value='200' valuetype='data'
object_
```

- **Naming Schemes**

_object command may insert a java applet with a prefix **'java:'**. To insert an ActiveX object it must have a prefix **'clsid:'**. Following paragraphs are two examples:

Java Applet

```
_object "codetype='application/java-archive'
       classid='java:program.start'"
  quote "Java Applet Example."
object_
```

ActiveX

```
_object "classid='clsid:663A8FEE-1EF7-12CF-B3DB070036F12432'
        data='program.stm'"
  quote "ActiveX Example."
object_
```

- **Declare and Instantiate Object**

Boolean **'declare'**attribute in the _object will not execute the object after loading. The **'id'** attribute sets a unique id for later using. In the following example, an anchor declares an object and instaniates it. You can click the linking text and active the object.

```
_object "declare id='SpaceID'
```

```
         data='space.mpg' type='application/mpeg'"
    quote "Load MPEG Object."
  object_
     ...
  _a "href='#SpaceID'"
    quote "MPEG Animation"
  a_
```

2. Applets

_applet command enables a java applet in an XHTML document. This command is depreciated in favor of the _object command. Its **'code'**attribute specifies the applet location.

```
_applet "code='audio' width='30' height='20'"
  _param "name='wave' value='sound.wav'"
applet_
```

the equivalent _object code is

```
_object "codetype='application/java'
       classid='audio' width='30' height='20'"
  _param "name='wave' value='sound.wav'"
object_
```

3. Images

__img command embeds an image in the current document. It is a special case of the _object. Its **'src'** attribute specifies the location of the image resource. The URL value of the **'longdesc'** attribute specifies a location for a long description of the image. When an image is loading, the alternate text of the **'alt'**attribute will display at first. This attribute is useful to display a web page in the low speed Internet. __img command may have width adn height attributes to specify the rect size of a picture.

```
__img "src='http://www.neatware.com/default.gif'
    alt='Snaml GIF Demo'"
```

the equivalent _object code is

```
_object "data='http://www.neatware.com/default.gif'
       type='image/gif'"
  quote "Snaml GIF Demo"
object_
```

_map is an image map command. It specifies regions of an image or an object and assign a specific action to each region. When a user actives a region, browser will invoke the corresponding action. A browser or a server may interpret the coordinates of a region. _map command's **'name'**'shape' attribute specifies the region shape with the value default, rect, circle, and poly. In addition, **'coords'** attribute specifies the position of a shape on the screen. The rect shape may have the coord list left, top, right, bottom; the coord list of circle is center-x, center-y, radius; and the coord list of poly is the pair of x_0, y_0, x_1, y_1, etal. To disable a region that associates a link, you must set **'nohref'** boolean attribute. Furthermore, **'usemap'** attribute associates an image map with an element. Finally, the **_area** command acts like _a command in the _map content.

```
__object "data='toolbar.gif' type='image/gif' usemap='#map'"
_map "name='map'"
  _area "href='beginner.html' shape='rect' coords='0,0,31,31'"
  area_
  _area "href='professional.html' shape='rect' coords='31,0,63,31'"
  area_
  _area "href='executive.html' shape='rect' coords='63,0,95,31'"
  area_
map_
```

Here is a source code of an image map procedure. Comparing to other web page editing method, Snaml code automatically compute the coordinates and flexible to change. Resuable Snaml code is another benefit.

```
#*******************************************
# rect image map
#  name image name
#  w image width
#  h image height
#  n number of sub-area
#  refer url list for each area
# *******************************************

proc ImageMap {name w h n {refer {}}} {
  for {set i 0} {$i < $n} {incr i} {
    set a [expr $w*$i/$n]
    set b [expr $w*($i+1)/$n]
    set rect($i) "$a,0,$b,$h"
  }

mapping

  __img "src='$name' width='$w'
      height='$h' usemap='#map'"
  _map "name='map'"
```

```
    for {set i 0} {$i < $n} {incr i} {
      _area "href='[list index $refer $i]'
        shape='rect' coords='$rect($i)'"
      area_
    }
  map_
}
```

# usage

```
ImageMap "toolbar.gif" 480 32 3 {One Two Three}
```

Table

Table Modules are divided into Basic and Advanced Table Modules. The Basic Table Module is the set of Advanced Table Module and is primiarily used for wireless Internet.

_table is a command to construct a table. The **_tr** command specifies table row and the **_td** command specifies table data. _table content may include a**_caption** command.

The attributes of basic _table include COMMON, summary, and width attributes. The attributes of advanced _table are: **'border'** attribute sets the table border with the integer from 0 to n. To declare the position of cells, the**'cellspacing'** attribute specifies the space between two cells; **'cellpadding'**attribute sets the space between the border and the content of a cell; and the**'height'** and **'width'** attributes specify the height and width of a table respectively. Finally **'bgcolor'** attribute specifies the background color of a table.

Table Demo

Title 1	Title 2
Area A	Area B
Area C	

To specify visible sides, **'frame'** attribute may be one of the value: *void, above, below, hsides, vsides, lhs, rhs, box,* and *border*. They represent the none side(default), top side, bottom side, top&bottom side, left&right side, left-hand side, right-hand side, and all four sides respectively. To describe which rules will appear between cells, the **'rules'** attribute may have one of a value *none, groups, rows, cols, or all*. In default when there are no frame and rules attributes, border='0' implies frame='void' and rules='none'; border='2' implies frame='border', rules='all', and border width is 2.

The **_tr** command defines a row. Its attributes may be COMMON, align (left | center | right), or valign (top | middle | bottom). The content of _th in Advanced Table Module may be divied into head (*_thead*), foot (*_tfoot*), and body (*_tbody*) sections. To group columns, **_colgroup** and **_col** commands define the property of group.

_th command specifies a cell head in the **_tr**. Usually it can be replaced by the**_td** command that defines a cell (data) in the content.

_td command may have the attributes 'align', 'valign', 'height', 'width', 'bgcolor', and other COMMON attributes. **'align'** attribute controls the horizontal alignment of the cell with the value left, right, or center; **'valign'**attribute controls the verticle alignment with the value top, bottom, or middle. In addition, The **'colspan'** attribute allows a cell span its width to one or more cells (default is 1). The 'rowspan' attribute spans its height to more cells. There are abbr, headers, axis, scope(row | col) attributes for _td command.

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
_head
  _title; quote "Table"; title_
head_
_body
  _table "border='1' cellspacing='4' cellpadding='8'
    bgcolor='#FFFFFF' nowrap='nowrap';
    align='center' valign='middle'
    width='320' height='200'
    frame='hsides' rules='groups'"

  _caption
    quote "Table Demo"
  caption_
  _thead
    _tr
      _td; quote "Title 1'; td_
      _td; quote "Title 2"; td_
    tr_
  thead_

  _tbody
  _tr "width='30%'"
    _td "colspan='1' bgcolor='#FF0000'"
      quote "Area A"
    td_
    _td "bgcolor='#0000FF'"
      quote "Area B"
    td_
  tr_
  _tr
    _td "colspan='2' bgcolor=#00FF00"
      quote "Area C"
    td_
  tr_
  tbody_
  table_
body_
html_
```

Frame

Frames allow authors to present document in multiple views. **_frameset**command replaces the _body command to divide web page into several frames. **'cols'** attribute specifies frames that are divided along the column. The 'cols' value specifies the width of each columns. You may specifies any number of columns with length such as '33%, 67%'. **'rows'** attribute specifies frames that are divided along the row. When both rows and cols are set simultaneously _frameset command creates a grid. Frames are created from left to right for the columns and from top to bottom for the rows. _frameset may be nested to any level. **'frameborder'** attribute with value '0' will hide the frame border and value '1' will show a 3D border. Finally, **'framespacing'**attribute specifies the number of whitespaces between frames.

The **_frame** command in the content of _frameset loads the html file from the**'src'** attribute value. Its **'name'** attribute is an id that allows other commands target this frame. The value '1' of the **'frameborder'** attribute draws a border around the frame and the value '0' makes no border. **'marginheight'** and**'marginwidth'** specify the margin height and width of a frame. The value 'yes' of the **'scrolling'** attribute makes browser display a scrolling bar on the frame. The value 'no' will hide the scrolling bar.

The content of the **_noframes** command is compabitiable to the browser that does not support the frame.

```
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
  _head
    _title; quote "Frame"; title_
  head_
  _frameset "cols='33%, 67%' frameborder='0' framespacing='1'"
    __frame "name='index' src='index.html'"
    __frame "name='content' src='content.html'"
  frameset_
  _noframes
    quote "older browser without frame."
  noframes_
html_
```

Form

1. **Form**

Form is a window that connects client to a program on the server. Several control components (widgets) such as checkbox help to complete this task.**'action'** attribute of the **_form** command has a URL value that refers to a CGI program. **'method'** attribute has one of the value GET, POST, and ENCTYPE. **GET** method sends the information through **env** variables; **POST**method sends the information through stdout; and **ENCTYPE** method specifies the MIME type for POST data(default 'application/x-www-form-data').

```
_html "xmlns='http://www.w3.org/1999/xhtml'
  xml:lang='en' lang='en'"
_head
  _title; quote "Form"; title_
head_
_body
  _form "action='/cgi-bin/form.sml' method='POST'"
  form_
body_
html_
```

2. **Button**

- **Submit**

'submit' button will send the name=value string to a CGI program. The **'type'** attribute of the **__input** command must be the value 'submit'. The **'value'** attribute sets the label of a button (default submit). Finally, **'name'** attribute sets the submit name.

```
_body
  _form "action='/cgi-bin/form.tcl' method='POST'"
    __input "type='submit' name='submit_name'
      value='button_label'"
  form_
body_
```

- **Reset**

Reset button clears all the input data. The **'type'** attribute of a**__input** command must be the value 'reset'.

```
_body
  _form "action='/cgi-bin/form.sml' method='POST'"
    __input "type='reset'"
  form_
body_
```

- **Image**

When user clicks on an **image** button, browser will send the coordinates of the mouse clicking point to a CGI program. The **'type'**attribute of the **__input** command has the value 'image'; **'src'**attribute specifies the file name of the image; and the **'name'** attribute is the button id.

```
_body
  _form "action='/cgi-bin/form.sml' method='POST'"
    __input "type='image' src='icon.gif' name='icon'"
  form_
body_
```

3. **EditBox**

Editbox will open a box with width specified in the **'size'** attribute. When**'type'** attribute has the value 'text', **__input** command allows you input text in an editbox. If **'type'** has the value **'password'**, the input characters will be displayed as asterisks. The **'hidden'** value of 'type' will hide editbox from display.

'name' attribute of the **__input** command specifies the name of an editbox. The **'value'** attribute specifies the initial string in the editbox (default 'string'). **'maxlength'** attribute specifies the max length of a string.

```
_body
  _form "action='/cgi-bin/form.tcl' method='POST'"
    __input "type='text' name='user' size='20' maxlength='50'"
  form_
body_
```

4. **RadioBox**

Radiobox is a group of circle buttons that only one can be checked. The**'type'** attribute in an **__input** command has the value 'radio'. In a group of radiobox all of them must be the same **'name'** attribute. However, they must be different in the **'value'** attribute. A CGI program will receive the selected radiobox. **'CHECKED'** attribute sets a radiobox as default.

```
_body
  _form "action='/cgi-bin/form.tcl' method='GET'"
    quote "First"
    __input "type='radio' name='order' value='first'"
    quote "Second"
    __input "type='radio' name='order' value='second'"
  form_
body_
```

5. CheckBox

Checkbox is a group of square buttons that can be checked on or off. The**'type'** attribute of a **__input** command has the value 'checkbox'. Each checkbox must have a different **'name'** attribute. The checked box will send a value 'on' to a CGI program in default. When the **'value'** attribut is defined the name=value will be send. The **'CHECKED'** attribute sets a checkbox on as default. You may check more than one box at the same time.

```
_body
  _form "action='/cgi-bin/form.tcl' method='GET'"
    __input "type='checkbox' name='red' CHECKED"
    __input "type='checkbox' name='green'"
    __input "type='checkbox' name='blue' CHECKED"
  form_
body_
```

6. MenuBox

MenuBox shows a long selectable list in a small space. The **'size'** attribute in **_select** command sets to 1 will make the form work as a menubox. An**_option** command lists an item. The **'SELECTED'** attribute in the _option command selects this item as default.

```
_body
  _form "action='/cgi-bin/form.tcl' method='GET'"
    _select "name='color' size='1'"
      _option; quote "red"
      _option; quote "green"
      _option SELECTED; quote "blue"
    select_
  form_
body_
```

7. ListBox

Similar to the MenuBox, however, **ListBox** must set **'size'** a greater than 1 value in the _select command. It is displayed as scrolled list. **'MULTIPLE'**attribute in the **_select** command will allow you select multiple items.

```
_body
  _form "action='/cgi-bin/form.tcl' method='GET'"
    _select "name='color' size='2' MULTIPLE"
    _option; quote red
    _option; quote green
    _option SELECTED; quote blue
    select_
  form_
body_
```

8. EditText

_textarea command defines a scrolled text editbox with attribute **'rows'**and **'cols'**. The **'name'** attribute specifies the name of a text area. Unlike the HTML text, the text in an EditText area will not ignore newlines.

```
_body
  _form "action='/cgi-bin/form.tcl' method='GET'"
    _textarea "rows='20' cols='40' name='words'"
    quote "Demo of Text Editor."
    textarea_
  form_
html_
```

9. CGI and Form

Client **Form** is a basic interactive interface connected to a CGI program. Client browser encodes form data by using a MIME type. The default MIME type in the ENCTYPES attribute is *application/x-www-form-urlencoded*. The value of **'name'** attribute attached with '=' and the entered string construct a key/value pair. All the pairs are linked by the **'&'** character to form a string. CGI encoder will translate control characters into hexadecimal codes with the prefix **'%'**. For example, the space character is translated to %20.

For **EditBox** and **EditText** in a form if user did not input anything, the value will be left to empty. For **RadioBox** and **CheckBox** in a form if a box is checked, the value will be the string in **'value'** attribute. Its default value is**'on'**. A client will not send a key/value pair if a box is not checked. For**MenuBox** and **ListBox** a client will send a key/value pair if user select the item of a **_option** command.

Following procdure describes how a CGI program decode the data. First, it checks **REQUEST_METHOD** environment variable. If the value is **GET** then read **QUERY_STRING** and **PATH_INFO**, otherwise, get the**CONTENT_LENGTH** and read data from the stdin. In GET method, the**QUERY_STRING** is the string that has been appended to the URL after the**'?'** character. It is possible to access a CGI program without using a form. Furthermore, CGI program splits the string to get the key/value pairs. Finally, CGI program converts the hexadecimal and **'+'** characters for all the key/value pairs. It also sets an array where key is index and value is an elemnet. The ParseForm program has an error checking as well.

```
#
# Report report the exception
# status server status
# keyword short description
# message detail description
#
proc Report {status keyword message} {
  quote "Status: $status; Keyword: $keyword; Message: $message"
}

#
# ParseForm parse CGI string
#   aFormInfo return form data array with key/value.
# Return RESULT_OK or RESULT_ERROR
#
proc ParseForm {aFormInfo} {
  upvar $aFormInfo sFormData

  set sMethod $::env(REQUEST_METHOD)
  switch -exact -- $sMethod {
    POST {set sQuery [file::read stdin $::env(CONTENT_LENGTH)]}
    GET  {set sQuery $::env(QUERY_STRING)}
    default {Report  500 "server error" "unsupport method"
             return RESULT_ERROR}
  }

  set pairs [string::split $sQuery &]
  foreach item $pairs {
    set pair [string::split $item "="]
    set key [list::index $pair 0]
    set value [list::index $pair 1]

    regsub -all {\+} $value { } value
    set range {[0-9a-fA-F]}
    while {[regexp "%$range$range" $value match]} {
      scan $match "%%%x" hex
      set symbol [string::format %c $hex]
      regsub -all $match $value $symbol value
    }

    if {[info exists sFormData($key)]} {
      string::append sFormData($key) "\0" $value
    } else {
      set sFormData($key) $value
    }
  }
  return RESULT_OK
}

#
# cgi main program
#
#!/usr/local/bin/tclsh

__cgi {Content-type: text/html}
_html "xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'"
  _head
    _title
      quote "CGI Form Program Sample"
    title_
  head_
 _body

  if {[string::compare [ParseForm sForm] RESULT_OK] == 0} {
    foreach item $sForm {
      quote "$item=$sForm($item)"
    }
  }

  body_
html_
```

Cascading Style Sheets (CSS)

1.CSS Basic

HTML was originally designed as a document description language that was device independent. Then it added more tags such as font for typography. HTML 4.0 is back to its origin and uses **C**ascading **S**tyle**S**heets (**CSS**) for its presentation. New CSS web pages clarify the document structure and presentation. Snaml prompts to use CSS to control web content as new browsers have implemented all CSS functions.

- **Cascading Style Sheets (CSS)**

CSS is defined by rules. In Snaml the **rule** command has the format:

```
rule selector {property: value; ...}
```

The first argument is called **selector**. Usually it is an element name of HTML. There are four kinds of selectors: type, attribute, contextual, and pseudo. The second argument in a curly brace is called **declaration**. It is a list of property and value pairs separated by ';'. Each pair is represented as 'property: value'. Note **rule** is a special block command with two arguments. It has no end command and content. You can think of it as an empty command with the start command representation.

```
rule p {color: blue; font-size: 12pt;
  font-family: sans-serif; background: white}
```

where **'p'** is an XHTML element; **'color'** is a property of a pargraph; and blue is the value of the **'color'**.

One way to add rules into an XHTML document is to use _style command in a document.

```
_head
  _title; _quote {CSS Demo}; title_
  _style {type='text/css'}
  _comment
    rule body {color: black; background: white;}
    rule h1 {color: green}
  comment_
  style_
head_
```

The type value 'text/css' of the **_style** command is also a default value. A browser that is unknown the CSS rules will ignore all the**_rule** commands as the content of the _comment command.

Another method uses **_link** command.

```
    __link {rel='stylesheet' type='text/css'
      href='http://neatware.com/file.css'}
```

A browser will get a CSS specification from file.css. The **'rel'**attribute value 'stylesheet' specifies that the URL is a CSS file. Simply refering to another CSS file, you can change the presentation of a HTML document dramatically.

- **Hierarchy and Inheritance**

A **HTML** document has a hierarchy. The XHTML element is the root of an XHTML document. Both HEAD and BODY are elements inside the XHTML and they are children of XHTML. Therefore XHTML is the parent of them. Other elements may be embedded into an XHTML document in the same way. There are no overlap for two block elements. Explicitly, for Snaml, if one start command is inside the content of a block command but its end command is outside the content, this document is illegal.

The hierarchy of XHTML document makes sense to the inheritance of **CSS**. Child elements will inherit the CSS properties from their parents. If a child and a parent share the same property but with different values, the child's value will override the parent's value in the scope of the child. It is called attribute inheritance.

```
rule body {font-family: arial, sans-serif;}
rule h1 {font-family: times, serif;}
```

The h1 will use times font in its context. Note the background property does not inherit. It is a global presentation property for a document.

Generally a browser provides a default CSS setting for XHTML document. It is the environment for document presentation. The CSS declaration of an XHTML document will override the same properties in the default setting.

- **Grouping**

Selectors can be grouped with comma-separated lists.

```
    rule {h1, h2, h3} { font-family: arial }
```

Declarations can also be grouped by ';' as well.

```
rule  h1 {
  font-weight: bold;
  font-size: 12pt;
  line-height: 14pt;
  font-family: arial;
  font-style: normal;}
```

In addition, font, color, and margin properties have their own grouping syntax. Following example is equivalent to the previous one

```
rule h1 { font: bold 12pt/14pt arial }
```

2. **Selectors**

- **Type Selectors**

Type selector may be any one of XHTML element such as h1.

```
rule h1 {color: blue}
```

- **Attribute Selectors**

There are **CLASS**, **ID**, and **Style** attributes.

  - **CLASS Attribute**

You may add class attribute into an XHTML command. It classifies a group of elements and makes them easy to change. Class name must be a character string without special symbols except hyphens and underscores. The selector starts with a dot '.' prefix in a rule command. A selector can only declare one class attribute.

```
_ol
  _li {class='deep'};  quote "red";   li_
  _li {class='light'}; quote "green"; li_
  _li {class='deep'};  quote "blue";  li_
ol_
```

.deep is a class selector. A rule is

```
rule .deep {font-weight: bold}
```

- **ID Attribute**

ID attribute specifies a unique value over an XHTML document. The selector starts with the #.

```
rule #ProductID {font-family: times}

  _h1 {id='ProductID'}
    quote {Product Information}
  h1_
  _em {id='ProductID'}
    quote {Web Builder}
  em_
```

- **Style Attribute**

Style attribute may embed into an XHTML element. In the following example, the selector is the element command name that the style attribute is embedded. Since this method mixed the document structure and its presentation, it is discouraged to use CSS in this way

```
_h1 {style='color: black; font-weight: bold'}
  quote {Style Attribute}
h1_
```

- **Contextual Selectors**

This selector allows you to apply rules on the context of an XHTML document. The selector elements are separated by whitespace. The preceding element is the parent of the later elements. In the following example, the last rule expresses that if em is inside the h1 then it has the weight italic, otherwise it is bold. This makes em distinguish even if it is in the context of h1.

```
rule h1 {font-weight: bold}
rule em {font-weight: bold}
rule {h1 em} {font-weight: italic}
```

- **Pseudo Selectors**
  - **Anchor pseudo-classes**

Pseudo selector has the format element:attribute . For example, psedo-classes selector displays newly visited anchors differently from older ones for an _a command is, All the **'_a'** command with an **'href'** attribute will belong to one of the above group.

unvisited link	rule {a:link} {color: blue}
visited links	rule {a:visited} {color: yellow}
active links	rule {a:active} {color: lime}

- **The 'first-line' pseudo-element**

  The **'first-line'** pseudo-element applies special styles to the first line of a paragraph.

```
_head
  _style type='text/css'
    rule {p:first-line} {font-style: small-caps}
  style_
head_
_body
  _p
    quote {News report from New York.}
  p_
body_
```

- **The 'first-letter' pseudo-element**

  To generate typographical effects on the first letter with drop caps, the **'first-letter'** pseudo-element is used in the CSS.

```
_html "xmlns='http://www.w3.org/1999/xhtml'
  xml:lang='en' lang='en'"
_head
  _title; quote {First Letter}; title_
  _style {type='text/css'}
    rule p {font-size: 12pt; line-height: 12pt}
    rule p:first-letter {font-size: 200%;
                         float: left}
    rule span {text-transform: uppercase}
  style_
head_
_body
  _p
```

```
        _span
          quote {This is}
        span_
        quote {a word of ant research.}
      p_
    body_
    html_
```

3. Formatting

CSS1 assumes each element is bounded in one or more bounding box. Content is the core. A padding area surrounds the core; a border area surrounds the padding area; and a margin area surrounds the border. From inner to outer order they are core, padding, border, and margin area. The box size is the sum of content, padding, border, and margin size. However, the padding and margin properties are not inherited.

- **Block Elements**

  Block elements have vertical and horizontal formatting.

  - **Vertical Formatting**

    The margin width specifies the minimum distance to the edges of surrounding boxes. Two or more adjoining vertical margins are collapse to use the maximum of the margin values.

  - **Horizontal Formatting**

    Seven properties, 'margin-left', 'border-left', 'padding-left', 'width', 'padding-right', 'border-right' and 'margin-right', determines the horizontal position and the size of a block element. Their sum is equal to the 'width' of their parent element.

- **List Item Elements**

  It presents list item. The first rule shows the list marker is outside the list box. The second rule shows the list marker is inside the list box.

```
rule ul         { list-style: outside }
rule ul.compact { list-style: inside }
```

- **Floating Elements**

By setting the **'float'** property to left, a box is moved to the left until it reaches to the margin, border, padding, or border of another block-level element. The normal text flow will wrap around on right side of the box.

```
_html "xmlns='http://www.w3.org/1999/xhtml'
  xml:lang='en' lang='en'"
_head
  _style {type='text/css'}
    _comment
      rule img { float: left }
      rule {body, p, img} { margin: 2em }
    comment_
  style_
head_
_body
  _p
    __img {src=image.gif}
    quote {Explain the image.}
  p_
body_
html_
```

- **Inline Elements**

'em' and **'strong'** are typical inline elements. They share space with other box elements.

- **Replaced Elements**

'img' is a replaced element. Its position and size will be replaced by the src attribute.

4. **Property**

- **Font Property**

Font is a specific size and variation of a typeface. Usually, it is measured in point. One point (pt) is equal to 1/72 inch in the CSS; an em unit of width is equal to 12pt; and an ex unit is equal to the x-height of a font.

A font belongs to one of five generic family: serif, sans-serif, monospaced, cursive, and fantasy. Firstly, **Serif** means decorative cross stroke. Time New Roman is a default serif font on Windows. Secondly, **Sans-serif** is a font without serifs. Arial and Helvetica are sans-serif font on Windows and Mac respectively. Thirdly,**Monospaced** font occupies the fixed width space. Courier is this kind of font. Other fonts have the variation width space. Furthermore, **Cursive** is the handwritten font. It is useful for headlines and decorative letters. The examples include Brush Script, Vivaldi, and Comic Sans. Finally, **Fantasy** is decorative font for text effection. There are Engraver, Impact, and Revue.

There are a group of font properties. **'font-family'** specifies font name and generic family; **'font-weight'** may be normal (default), bold, and bolder, it may also be one of a number value of 100, 200, 300, 400, 500, 600, 700, 800, and 900 where smaller number is lighter; and **'font-style'** may have the value normal (default), italic, and oblique. To render lowercase letters as a small version of the uppercase letters, you can declare the **'font-variant'** property with the value small-caps. Its default value is normal.

Finally, **'font-size'** specifies the size of a font, its value may be absolute, relative, lenght, and percentage. The absolute size may be one of a value xx-small, x-small, small, medium, large, x-large, and xx-large. The relative size may be the value smaller or larger. They are relative to the parent size. The length size is the exact value with unit point (pt), inch (in), millimeter (mm), or centimeter (cm). Percentage size is represented as the n% where n is a number.

Sometimes it is convenient to select all the font property in a rule. For example:

```
{font-family: arial, helvetica, sans-serif}
{font-weight: bold}
{font-style: oblique}
{font-variant: small-caps}
{font-size: 12pt}

{font: arial, helvetica, sans-serif,
      bold, oblique, small-caps, 12pt}
```

- **Color Property**

You can set color to be name such as red or RGB value. To background, **'background-color'** may be a color value or 'transparent' (default); **'background-image'** may be a url pointer to an image file or a value 'none'; **'background-repeat'** specifies how to repeat the background image, it may be a value 'repeat' (default), 'repeat-x', 'repeat-y', and 'no-repeat'. In addition,**'background-attachment'** determines whether a background image is 'fixed' or 'scroll' along with the content. Finally,**'background-position'** specifies the initial position of a background image where '0% 0%' is the default value. It maybe a percentage or length pair. The combination from top, center, bottom and left, center, right may also construct a pair. The**'background'** property is a shorthand that specifies a group of background properties.

```
{color: red}
{color: rgb(255,128,64)}
{background-color: #FF00FF}
{background-image: url(mosaic.gif)}
{background-repeat: repeat-y}
{background-attachment: fixed}
{background-position: 0% 0%}

{background: url(abc.png) gray 50% repeat fixed}
```

- **Text Property**

Text property describes a text. **'text-decoration'** property specifies a text line with value 'underline', 'overline', 'line-through', and 'blink'. To convert letters to uppercase or lowercase, you can set the **'text-transform'** to be 'uppercase' or 'lowercase'. The value 'capitalize' will set first letter to be uppercase and the value 'none' (default) does nothing. To control text alignment, **'text-align'** specifies alignment of text with one of a value 'left', 'right', 'center', and 'justify'. **'vertical-align'** specifies text position with the value 'baseline' (defult), 'sub', 'super', 'top', 'text-top', 'middle', 'bottom', 'text-bottom', and 'percentage'. In addition, **'text-indent'** is the indent of text with the value of length or percentage. Finally, **'letter-spacing'**, also known as **kerning**, is the width between two letters with value 'normal' (default) or a value 'length';**'letter-height'**, also called **leading**, is the spacing between lines and may be the value normal (default), number, length, and percentage.

```
{text-decoration: line-through}
{text-transform: capitalize}
{text-align: justify}
{text-indent: 0.2em}
{vertical-align: middle}
{letter-spacing: normal}
{letter-height: 20%}
```

- **Box Property**

Any element is in a bounding box. An element's content area is surrounded by padding, border, and margin areas in order. The box width is the sum of the content width and two times padding, border, and margin widths.

- **Margin**

Margin maybe the value of length, percentage, and auto (default). The properties of **'margin-top'**, **'margin-right'**,**'margin-bottom'**, and **'margin-left'** specify the top, righ, bottom, and left margin respectively. The **'margin'** property is a shorthand for all of them.

```
{margin-top: 2em}
{margin-right: 33.3%}
{margin-bottom: 2px}
{margin-left: 1in}
{margin: 2em}
{margin: 1em 2em}
{margin: 1em 2em 3em 4em}
```

- **Border**

Border may have the value 'thin', 'medium', and 'thick'. The properties of **'border-top-width'**, **'border-right-width'**,**'border-bottom-width'**, and **'border-left-width'** specify the corresponding border with a value 'thin', 'medium', 'thick', and 'length'. The **'border-width'** property is a shorthand of above declaration.

'border-color' property specifies the color of a border.**'border-style'** specifies border style. It may be one of the following values 'solid', 'dashed', 'dotted', 'ridge', 'double', 'outset', 'inset', 'groove', and 'none' (default). The properties of **'border-top'**, **'border-right'**, **'border-bottom'**, and**'border-left'** are shorthand of 'border-top-width', 'border-right-width', 'border-bottom-width', and 'border-left-width' with the 'border-style' and 'border-color'. The **'border'**property is the shorthand of 'border-width', 'border-style', and 'border-color'.

```
{border-top-width: 0.1em}
{border-right-width: medium}
{border-bottom-width: thin}
{border-left-width: thick}
{border-width: thin}
```

```
{border-style: dotted}
{border-color: black}

{border-top: thick solid red}
{border-right: thin dotted blue}
{border-bottom: medium double green}
{border-left: thin, ridge, yellow}
{border: solid red}
```

- **Padding**

Padding may have the value of length or percentage. The properties of **'padding-top'**, **'padding-right'**, **'padding-bottom'**, and **'padding-left'** specify the top, right, bottom, and left padding respectively. The 'padding' property is a shorthand of above declaration. It is similar to the margin declaration.

```
{padding-top: 0.3em}
{padding-right: 10px}
{padding-bottom: 2em}
{padding-left: 25%}

{padding: 1em}
{padding: 1em 2em}
{padding: 1em 2em 3em 4em}
```

- **Float Property**

It is a layout property. **'float'** maybe one of a value 'none' (default), 'left', and 'right'. 'left' sets an element to the leftmost of its parent. 'right' sets an element to the rightmost of its parent.

```
{float: left}
```

'clear' specefies if a floating element is allowed on its side. It may be one of a value 'left', 'right', 'both', and 'none' (default). The value 'none' allows floating element on its sides.

```
{clear: right}
```

- **Position Property**

The properties **'width'** and **'height'** are usually applied to **img**element to set the display width and height of an image. The**'position'** property maybe absolute or relative position.

```
{width: 50%}
{height: 10px}
{position: absolute; left: 20 top: 30}
```

- **Classification Property**

'display' property describes how to show an element on the canvas. It maybe one of the value 'block', 'inline', 'list-item', and 'none'. **'white-space'** property specifies how to handle whitespace. Its value **'pre'** will keep all the whitespace; **'normal'** value ignores extra whitespace and return character; **'nowrap'** value will not wrap on the end of a line.

'list-style-type' property describes the appearance of a list marker. It may be one of a value 'disc' (default), 'circle', 'square', 'decimal', 'lower-roman', 'upper-roman', 'lower-alpha', 'upper-alpha', and 'none'. **'list-style-image'** property sets the list marker as an image. To specify the position of the list marker, **'list-style-position'** maybe set the value inside or outside. Finally, **'list-style'**is a shorthand of **list-style-type**, **list-style-image**, and **list-style-position**.

```
{display: block}
{whitespace: pre}

{list-style-type: decimal}
{list-style-image: ellipse.gif}
{list-style-position: inside}
{list-style: lower-alpha ellipse.gif inside}
```

Script

1. **Script**

   Scripts are programs that embed in a HTML document. Scripts may modify the document dynamically, response events such as mouse moving, and generate GUI. One kind of script executes one time when a document is loaded. Another kind of script is triggered whenever a specific event occurs.

   _script command defines a script in its content. It maybe nested and appeared any times. **'type'** and **'src'** attribute specify the scripting language (e.g. text/javascript) and the location of an external script respectively.

```
_script "type='text/javascript'"
  # ... JavaScript
script_
```

2. **Event**

   Intrinsic events will active script to execute. There are many events for client interactive controls.

onload it occures after loading a frame or window. used in _frameset or _body.	**onunload** it occures when a browser removes a doc. used in _frameset or _body.
onclick it occures when mouse is clicked over an element.	**ondbclick** it occures when mouse is double clicked over an element.
onmousedown it occures when mouse button is pressed on an element.	**onmouseup** it occures when mouse button is released over an element.
onmouseover it occures when mouse moved onto an element.	**onmousemove** it occures when mouse moved while it is over an element.
onmouseout it occures when mouse moved away from an element.	**onkeypress** it occures when a key is pressed and released over an element.
onkeydown it occures when a key is pressed down over an element.	**onkeyup** it occures when a key is released over an element.
onfocus it occures when an element receives focus. used with _label, _input, _select, _textarea, and _button commands.	**onblur** it occures when an element loses focus. used like 'onfocus'.

onsubmit
it occures when a form is submitted. used with _form.

onreset
it occures when a form is reseted. used with _form.

onselect
it occures when user selects a text clip in a text field. used with _input and _textarea.

onchange
it occures when a control loses the input focus and its value has been modified _input, _select, or _textarea.

3. **Activation**

The intrinsic event has a value of script. An event actives the executation of a script. Script's syntax is dependent on the scripting language. Except the Javascript and VB Script, Snaml is also suitable to write a script.

The **_noscript** command provides alternate content when a script is not executed. A better practice is adding **_comment** command in the script content. A browser that does not recognize the scripting language will ignore them.

```
_input "name='edit1' size='32'"
_script "type='text/Snaml'"
  _comment
  proc edit1_changed {} {
    if {[edit value] == "Snaml"} {
      button1 enable 1
    } else {
      button1 enable 0
    }
  }
  edit1 onChange edit1_changed
  comment_
script_
```

Extended Backus-Naur Form (EBNF)

EBNF is used to describe the formal grammar of a language. A rule has the form '**symbol ::= expression**'. If a symbol is defined by a regular expression then it has an initial capital letter, otherwise it has an initial lower case letter. Following table lists the EBNF definition.

#xN	N is a hexadecimal integer. a character in ISO/IEC 10646
'string'	matches a literal string that inside
"string"	the single and double quotes
[a-z]	matches any character from a to z
[^a-z]	matches any character outside the ranage a to z
[^abc]	matches any character not among the given character
(expr)	is a unit
A?	is optional A; matches A or nothing
A*	matches zero or more occurences of A
A+	matches one or more occurences of A
A B	matches A followed by B
A|B	matches A or B but not both
A-B	matches A but does not match B