Template Language and Syntax Reference

This chapter describes the language and syntax conventions used in the VPN Solutions Center template implementation.

Grammar and Syntax

The Extensible Markup Language (XML) template definition section consists of a sequence of IOS command lines, a sequence of control statements and some template comments.

This section describes in detail the template language grammar and syntax. For specifying the syntax of the template language, Cisco used a widely-used notation, called context-free grammars (Bachus-Naur Format). For specifying the semantics of the template language, we use informal descriptions and examples.

Argument

```
argument →
    variable = variable
    | variable = constant
    | variable = array_variable
    | variable = IOS_command
```

Argument List

```
argument_list →
    argument
    | argument_list, argument
```

Array Variable

```
array_variable →
    variable[index]
    | array_variable[index]
```

Constant

```
constant →
    num
    | float
    | quoted_string
```

Expression

```
expression →
    term
    | (expression)
    | expression relational_opr expression
    | expression logical_opr expression
    | expression additive_opr expression
    | expression multiplicative_opr expression
```

Index

```
index →
    variable
    | num
```

IOS Command Expression

```
IOS_command_expression →
    IOS_command
    | expression
    | IOS_command_expression expression
    | IOS_command_expression IOS_command
```

Repeat Counter Variable

```
repeat_counter_variable →
    $(letter | digit)*
```

Repeat Variable

```
repeat_variable →
    variable
    | num
    | array_variable
```

Statement

```
statement →
    IOS_command_expression
    |IOS_comment
    |Template_comment
    |{statement_list }
    |#if ( expression ) statement
    | #if ( expression ) statement #else statement
    | #repeat ( repeat_variable, repeat_counter_variable ) statement
```

Statement List

```
statement_list →
    statement
    |statement_list statement
```

Template

```
template →
    statement_list
```

Template Variable

```
template_variable →
    variable ( argument_list )
    | variable ( )
```

Term

```
term →
    variable
    | constant
    | array_variable
    | template_variable
```

Variable

```
variable →
    $(letter | digit)*
```

Lexical Conventions

This section summarizes the lexical conventions and the notations used in the VPN Solutions Center template grammar:

Tokens

There are six classes of tokens: IOS commands, variables, keywords, constants, operators and other separators.

Blanks, tabs, new lines and comments (collectively referred to as “white space”) as described below are ignored except when they separate tokens. Some white space is required to separate otherwise adjacent keywords and constants.

Template Comments

Template comments are indicated by ‘//’; everything after the comment indicator is treated as a template comment. Template comments are not included in the template configuration file—they are discarded during the template generation process.

Token Variables

A token variable is a sequence of letters and digits. The first character must be a letter ‘**$**’; it is followed by letters or digits:

letter a–z, A–Z

digit 0–9

Upper and lowercase letters are treated as being different. Variables can be up to 32 characters in length.

Keywords

Keywords are reserved and appear in boldface. The following keywords are reserved and cannot be used in any other context:

Table B-1 Template Keywords

#and	**#else**	**#eq**	**#ge**
#gt	**#if**	**#le**	**#lt**
#ne	**#or**	**#repeat**	

Constants

There are several kinds of constants, as described below:

- A Token **num** constant consists of a sequence of digits:

  num digit digit*

- A Token **float** constant consists of three parts: integer, a decimal point, and a fraction. The integer and fraction parts both consist of a sequence of digits:

  float num.num

- A Token **quoted_string** is a sequence of characters surrounded by double quotes, as in "….".

Expressions

The expressions used in VPN Solutions Center templates are as follows:

- IOS commands, variables
- Array_variables
- Template_variables
- Constants
- Expressions in parentheses

The precedence of an expression operator is the same as the order that the expressions are documented in this section (that is, multiplicative operators first, followed by additive operators, and so on), with the highest precedence first.

1. **Multiplicative Operators**

   The operators include ***** and /, which group left-to-right. The operands of * and / must be of arithmetic type. The binary ***** operator denotes multiplication. The binary / operator yields the quotient of the division of the first operand by the second.

2. **Additive Operators**

   The operators include + and – , which group left-to-right. The result of the + operator is the sum of the operands. The result of the – operator is the difference of the operands.

3. **Relational Operators**

   The relational operators group left-to-right. The operators are:

   - **#lt** (less than)
   - **#gt** (greater than
   - **#le** (less than or equal to)
   - **#ge** (greater than or equal to)
   - **#eq** (equal to)
   - **#ne** (not equal to)

   All of the relational operators yield 0 if the specified relation is false, and yield 1 if it is true. The type of the result is an integer.

4. **Logical Operators**

   The logical operators group left-to-right. They are as follows:

   - **#or**

     The **#or** operator returns 1 if both its operands compare unequal to zero, and 0 otherwise.

   - **#and**

     The **#and** operator returns 1 if either of its operands compare equal to zero, and 0 otherwise. The operands need to be the same arithmetic type and the result is an integer.

5. **Array Operators**

   The array operators are as follows:

   - One-dimensional operator: [a]
   - Two-dimensional operator: [a] [b]

Statements

Except when described otherwise, statements are executed and output in sequence. Statements are executed for their effect and they do not have values. They fall into the following several groups:

- IOS Command Expression Statement

  Most statements in the template are **IOS command expression** statements, which are Cisco IOS router commands.

- Compound Statement

  A compound statement (also called a “block”) is used so that several statements can be used where one is expected.

- Selection Statements

  The selection statements have two forms. In both forms of the **#if** statement, the expression, which must be an arithmetic type, is evaluated.

  If it compares unequal to 0, the first substatement is executed.

  In the second form, if the expression is 0, the second (**#else**) substatement is executed.

- Iteration Statement

  The iteration (**repeat**) statement specifies a looping operation. Depending on how many values are specified for the **repeat** variable, the **#repeat** statement executes the substatement as many times as there are values defined for the variable.

  For example, the following **repeat** statement would cause the **access-list** substatement to repeat as many times as there are values specified for the **ACL** (one-dimensional) variable.

  The **$i** specified is the *repeat counter variable*. This repeat counter variable is set to 0 initially; the repeat counter is incremented by 1 for each iteration of the statement.

```
!
#repeat ($ACL, $i)
{
access-list $ACL[$i] permit ip $Source_IP_Address[$i] $Source_Mask[$i]
$Dest_IP_Address[$i] $Dest_Mask[$i] precedence 7
}
```

- The **$i** specified in the repeat statement is the counter for the repeat statement. This counter variable is set to 0 initially; the counter is incremented by 1 for each iteration of the statement.

  For example, the **repeat** statement in the example above would cause the **access-list** substatement to repeat as many times as there are values specified for the **ACL** (one-dimensional) variable. This produces in the template configuration file one line for each iteration.

- The **$i** specified in the variables declared in the substatements indicates that it is a one-dimensional variable. For example, for each value specified for the **Source_IP_Address** variable, the template substitutes one value for **Source_IP_Address**.