APPENDIX D

Template Usage

The following questions and answers can help you troubleshoot. The following topics are for Template Manager, which is explained in Chapter 6, "Service Design":

- How do I obtain address information from the given IP address?, page D-2
- How do I obtain the octets from the given IP address?, page D-2
- How do I obtain the octets from the given IP address?, page D-2
- How do I call a subtemplate in a template?, page D-2
- How do I concatenate two strings?, page D-3
- How can I convert a string to an integer and how can I increase the last octet of the IP address by one?, page D-3
- Can I use nested if statements?, page D-3
- How can I perform basic arithmetic operations?, page D-4
- How can I retrieve data from a two-dimensional array and what is the use of $velocityCount?, page D-4
- How can I print $a instead of its value?, page D-4
- What is the difference between #include() and #parse()?, page D-5
- What is a macro and how is it used?, page D-6
- What is a range operator and how can I use it?, page D-6
- How can I split strings containing special characters?, page D-7
- How can I use repository variables?, page D-7
- How can I use a variable as a dynamic URL?, page D-7
- Can I see more examples?, page D-7

How do I split a string?

ISC provides a function substring**ToDelim(),** which can split the given string and return the substring based on the given delimiter.

Syntax:

substringToDelim (srcString, delimChar, 0/1)
where:

0 returns the string before the delimiter.
1 returns the string after the delimiter.

Usage: **$b=$TMSystem.substringToDelim(“10.11.230.145”, ".230.145", "0")**

Result: The value of **$b** is **10.11**. If **1** is specified instead of **0**, the value of **$b** is **230.145**.

How do I obtain address information from the given IP address?

ISC provides the functions that can be used to get the address, mask, and reverse mask from the given IP address.

Usage:

$TMSystem.getAddr ("10.33.4.5/30") returns 10.33.4.5
$TMSystem.getMask ("10.33.4.5/30") returns 255.255.255.252
$TMSystem.getReverseMask ("10.33.4.5/30") returns 0.0.0.3
$TMSystem.getNetworkAddr ("10.33.4.5/30") returns 10.33.4.4
$TMSystem.GetClassfulNetworkAddr ("10.33.4.5/30") returns 10.0.0.0
$TMSystem.CurrentTimeInIOSFormat () returns hh:mm:ss day_of_month month_of_year year

How do I obtain the octets from the given IP address?

ISC provides the functions that can return the octets when called.

Usage:

$TMSystem.getOctet1($ipAddr) returns the first octet of **ipAddr**
$TMSystem.getOctet2($ipAddr) returns the second octet of **ipAddr**
$TMSystem.getOctet3($ipAddr) returns the third octet of **ipAddr**
$TMSystem.getOctet4($ipAddr) returns the fourth octet of **ipAddr**

How do I call a subtemplate in a template?

A subtemplate can be called in a main template. The subtemplate being called should be called with its datafile. The variable is declared as a subtemplate. The location of the subtemplate is specified in the datafile.

Usage: In the template body the subtemplate is declared as:

$a. callWithDatafile("data1")

where:

the variable **a** is declared as a subtemplate in the variables

data1 is the name of the datafile of the subtemplate, and

in the datafile the path of the subtemplate path is specified.

How do I concatenate two strings?

Concatenation of strings is simple.

For example:

where: **$a=vpnsc** and **$b=properties**
then: **${a}${b}** concatenates these two strings and gives the result as **vpnscproperties**.
or, **${a}_${b}** gives the result as **vpnsc_properties**.

How can I convert a string to an integer and how can I increase the last octet of the IP address by one?

The last octet of the IP address can be increased by using the following code:

```
#set($d=$TMSystem.getOctet1($c))
#set($e=$TMSystem.getOctet2($c))
#set($f=$TMSystem.getOctet3($c))
#set($g=$TMSystem.getOctet4($c))
#set($valueOfString = $g)
#set($valueOfCharsCount = $valueOfString.length() - 1)
#set($valueOfVector = "0123456789")
#set($valueOfBase = 1)
#set($valueOfInt = 0)
#foreach($valueOfCharIterator in $valueOfCharsCount..0)
#set($valueOfChar=$valueOfString.charAt($valueOfCharIterator).toString())
#set($valueOfInt = $valueOfInt +$valueOfVector.indexOf($valueOfChar) * $valueOfBase)
#set($valueOfBase = $valueOfBase * 10)
#end
#set($valueOfInt = $valueOfInt+1)
```

The incremental value is **$d.$e.$f.$valueOfInt**

Can I use nested if statements?

If statements can be nested. Proper care must be taken for indentation when nesting **if** statements. The following code shows the usage of nested **if** statements, **elseif** statements, and the comparisons made in the **if** clause.

```
#if($a==”a”) // here: string comparison is made
   --
      #if($b || $d) // here: $b and $d are the Boolean expressions. || equals OR and && equals AND
         --
         #if(!$c) // here: $c can be integer, string, or Boolean.
            ---
               #if($p<10)// here: $p is a integer.
               #elseif($p==10)
               #end
        #end
     #end
 #end
```

How can I perform basic arithmetic operations?

Velocity Template Language (VTL) supports built-in mathematical functions that can be used in the templates with the set directives.

Usage:

#set($a = $b + 3)
#set($a = $b - 6)
#set($a = $b * 6)
#set($a = $b / 5)
#set($a = $b % 2)

Note Only integers are valid for performing mathematical operations in the VTL.

How can I retrieve data from a two-dimensional array and what is the use of $velocityCount?

The default name for the loop counter variable reference, which is specified in the velocity.properties file, is **$velocityCount**. By default the counter starts at **1**, but this can be set to either **0** or **1** in the **velocity.properties** file at:
$ISC_HOME/resources/webserver/tomcat/shared/lib/velocity-dep-*VelocityVersion***.jar** (where the current *VelocityVersion* is 1.3.1-rc2). The associated settings are:

directive.foreach.counter.name=velocityCount
directive.foreach.counter.initial.value=1

Data from an array can be obtained by using **get($i)**

where: **$i** is the $velocityCount.

The following example illustrates the usage of the method **get()**:

Usage:
```
#foreach ($Acl in $ACL-List)
    #set ($i = $velocityCount)
    #foreach ($protocol in $Protocol-Lists.get($i))
    #set ($j = $velocityCount)
       access-list $Acl permit $protocol $Source-IP.get($i).get($j)
    #end
    #end
```

where:

$ACL-List is a one-dimensional array.
$Protocol-Lists and **$Source-IP** are two-dimensional arrays.

Here the $velocityCount is set to **1** by default. It can be changed in velocity.properties, if desired.

How can I print $a instead of its value?

Printing a value without processing is done by use of the character \ ,even if the value of the variable for **a** is defined.

Usage:

\$a gives output as **$a** if **$a** is defined. If **$a** is not defined, it is printed as **\$a**.

What is the difference between #include() and #parse()?

The **#include("velocity.txt")** directive allows you to import a file and then include the file in the location where it is defined. The content of the file is made available to the template engine. The *.vm files can also be called by using **#include**. The name of the file can also be passed by a variable. For security reasons, the file should be included under **TEMPLATE_ROOT** (/vob/ntg/dev/resources/templatesystem).

The **#parse("velocity.vm")** directive allows you to import a local file that contains VTL.Velocity will parse the VTL and render the template specified. The template that **#parse** references must be included under **TEMPLATE_ROOT**. The **#parse** directive only takes a single argument. VTL templates can have **#parse** statements referring to templates that in turn have **#parse** statements. The default value of the **directive.parse.max.depth** property is set to 10, in the **velocity.properties** file at: **$ISC_HOME/resources/webserver/tomcat/shared/lib/velocity-dep-***VelocityVersion***.jar** (where the current *VelocityVersion* is 1.3.1-rc2) and can be modified, if desired.

Note If the **directive.parse,max.depth** property is not present in the **velocity.properties** file, the default is set to 10.

Example:

In **TEMPLATE_ROOT**, the file **velocity.vm** has the following content:

```
welcome to the parse file
The count is $count
#set($count = $count - 1)
#set($cl-list="cl1","cl2","cl3")
#foreach($i in $cl-list)
ipcommunity-list permit $i 30:20
#end
The count is $count
returning from parse
```

The template body contains the following:

```
#set($count=8)
#include("velocity.vm")
-------------------------
#parse("velocity.vm")
-------------------------
welcome back to template
The value of count is $count
```

The following O/P is obtained:

```
welcome to the parse file
The count is $count
#set($count = $count - 1)
#set($cl-list="cl1","cl2","cl3")
#foreach($i in $cl-list)
```

```
ipcommunity-list permit $i 30:20
#end
The count is $count
returning from parse
-------------------------
welcome to the parse file
The count is 8
ipcommunity-list permit cl1 30:20
ipcommunity-list permit cl2 30:20
ipcommunity-list permit cl3 30:20
The count is 7
returning from parse
-------------------------
welcome back to template
The value of count is 7.
```

Note The previous examples clearly show that variables are parsed in the **#parse** directive and not in the **#include** directive.

What is a macro and how is it used?

The directive macro is almost similar to a function. This has a set of statements, which can be called repetitively.

Example:

```
#macro(community $CL $bgp-list)
   #foreach($bgp in $bgp-list)
      ip $CL standard permit $bgp
   #end
#end

#set($bgp_list ="20:10","30:10","40:10","50:10")
#set($CL = "community-list")

#community($CL $bgp_list)
```

Here, the macro name of **community** is defined. The macro takes two arguments **$CL** and **$bgp-list**. The macro is called at the end line.

The output of the previous template is:

```
ip community-list standard permit 20:10
ip community-list standard permit 30:10
ip community-list standard permit 40:10
ip community-list standard permit 50:10
```

What is a range operator and how can I use it?

The range operator can be used in conjunction with **#set** and **#foreach** statements. It is used to produce an object array containing integers. The range operator has the following construction **n..m**.

Example:

```
#set($a=0..2)
#foreach($b in $a)
    $b
#end
#foreach($c in -2..2)
$c
#end
```

How can I split strings containing special characters?

#foreach ($i in $PE_Intf_Name.split('\.')) $i #end

here: In the first iteration, **$i** contains the string before the period, and in the second iteration, **$i** contains the string after the period.

How can I use repository variables?

Repository variables can be selected in the datafile. When a template along with a datafile is associated with a Service Request and the Service Request is deployed, then the value of the repository variable gets substituted.

How can I use a variable as a dynamic URL?

A variable declared as a dynamic URL can call the URL, by the method:

callUrl(String S)

For example: **$a. callUrl("http://www.cisco.com")**

Can I see more examples?

Examples are given for:

- Usage of Strings, page D-8
- Usage of a Macro, page D-9
- Usage of Subtemplates, page D-10

Usage of Strings

The body of the template contains:

```
## This example illustrates the usage of strings

#set($a="Fast")
#set($b="ethernet")
interface ${a}_${b}

#foreach ($i in $PE_Intf_Name.split('\.'))
$i
#end

#set($c="10.11.230.145")
#set($b=$TMSystem.substringToDelim($c, ".230.145", "0"))
interface Loopback1
description By VPN-SC
ip vrf forwarding V31:eigrpfm
ip address ${b}.20.34 255.255.255.255
no ip directed-broadcast

#set($b=$TMSystem.substringToDelim($c, ".230.145", "1"))
interface Loopback1
description By VPN-SC
ip vrf forwarding V31:eigrpfm
ip address 20.45.${b} 255.255.255.255
no ip directed-broadcast

#set($c="10.33.4.5/30")
#set($d=$TMSystem.getAddr($c))
The Address of $c is $d
#set($d=$TMSystem.getMask($c))
The mask of $c is $d
#set($d=$TMSystem.getReverseMask($c))
The Reverse mask of $c is $d
#set($d=$TMSystem.getNetworkAddr($c))
The network address of $c is $d

#set($e=$TMSystem.currentTimeInIOSFormat())
The current time in IOS format is : $e

-------------------------------------------------------
getting the octets from the ipaddress
#set($c="10.33.4.5")
#set($e=$TMSystem.getOctet1($c))
The first Octet of $c is $e
#set($e=$TMSystem.getOctet2($c))
The second Octet of $c is $e
#set($e=$TMSystem.getOctet3($c))
The third Octet of $c is $e
#set($e=$TMSystem.getOctet4($c))
The fourth Octet of $c is $e
```

The variables are declared as strings, integers, or sub-templates accordingly.

The Output of the above template body is:

```
interface Fast_ethernet

10
11
12
13

interface Loopback1
description By VPN-SC
ip vrf forwarding V31:eigrpfm
ip address 10.11.20.34 255.255.255.255
no ip directed-broadcast

interface Loopback1
description By VPN-SC
ip vrf forwarding V31:eigrpfm
ip address 20.45.230.145 255.255.255.255
no ip directed-broadcast

The Address of 10.33.4.5/30 is 10.33.4.5
The mask of 10.33.4.5/30 is 255.255.255.252
The Reverse mask of 10.33.4.5/30 is 0.0.0.3
The network address of 10.33.4.5/30 is 10.33.4.4

The current time in IOS format is: 00:17:01 21 Aug 2006

-----------------------------------------------------
getting the octets from the ipaddress
The first Octet of 10.33.4.5 is 10
The second Octet of 10.33.4.5 is 33
The third Octet of 10.33.4.5 is 4
The fourth Octet of 10.33.4.5 is 5
```

Usage of a Macro

The body of the template contains:

```
## This example illustrates the usage of macro

#macro(community $CL $bgp-list)
#foreach($bgp in $bgp-list)
ip $CL standard permit $bgp
#end
#end

#set($bgp_list = "20:10","30:10","40:10","50:10")
#set($CL = "community-list")
```

#community($CL $bgp_list)

The Output is obtained as:

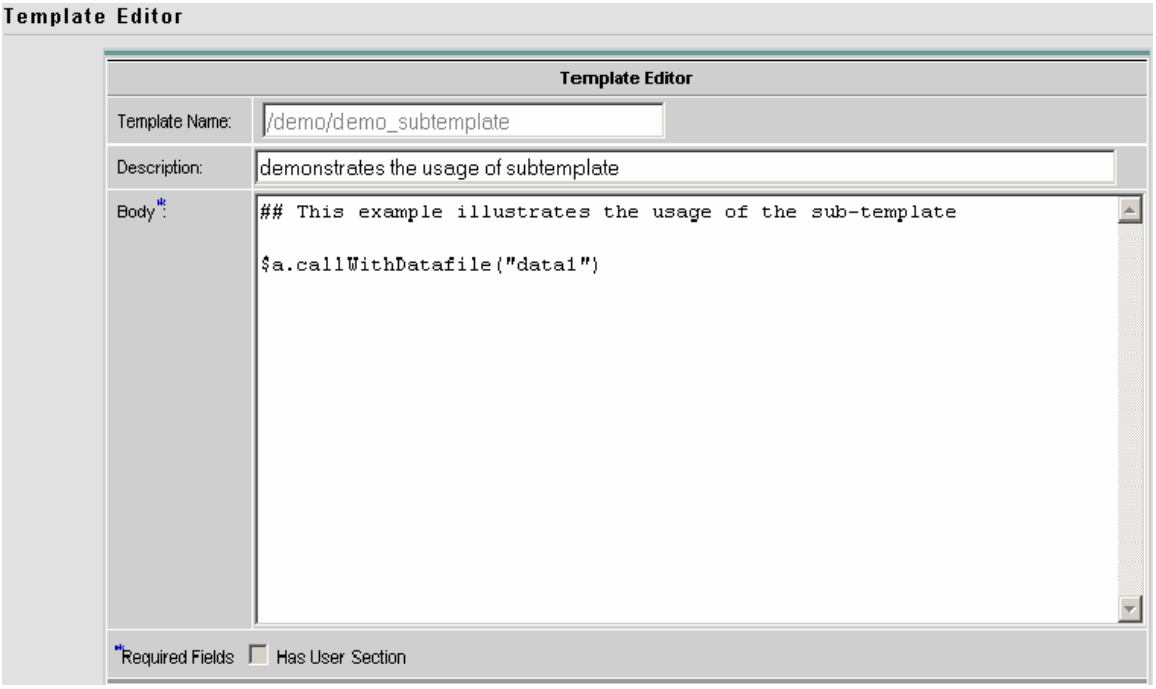
ip community-list standard permit 20:10
ip community-list standard permit 30:10
ip community-list standard permit 40:10
ip community-list standard permit 50:10

Usage of Subtemplates

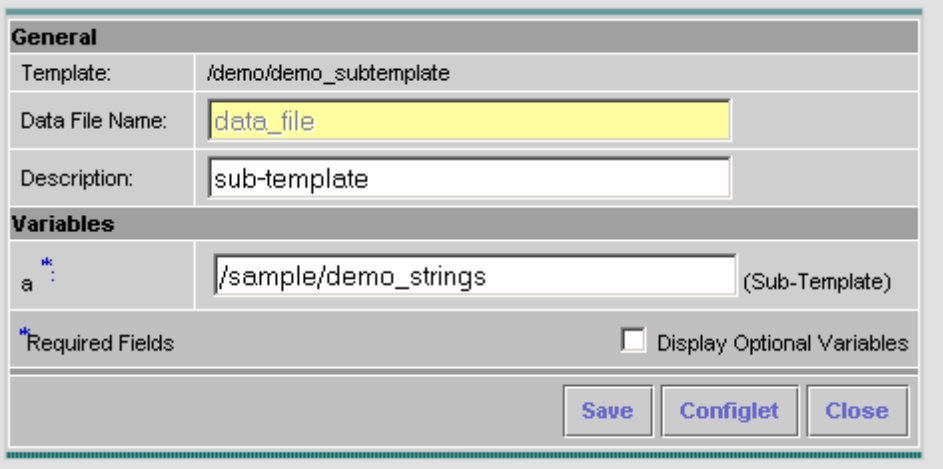
The body of the template is as follows:

This example illustrates the usage of the sub-template

$a.callWithDatafile("data1")



The variable **a** is declared as a subtemplate. The datafile provided here, **data**, must be a datafile for the template **a**, which must also exist. In the datafile of the main template, the path of the subtemplate is specified.



In the datafile of the main template, the specified path of the subtemplate might be the same directory or a different directory.