Using Descriptor Variables and Parameters

Variable descriptors allow you to define commonly-used information once and refer to them symbolically throughout your application descriptors.

On this page:

- Descriptor Substitution Syntax
  - Limitations
  - Escaping a Variable
- Special Descriptor Variables
- Descriptor Variable Scoping Rules
  - Resolving a Reference
  - Template Parameters
  - Modifying a Variable

Descriptor Substitution Syntax

Substitution for a variable or parameter *VP* is attempted whenever the symbol ${*VP*} is encountered, subject to the limitations and rules described below. Substitution is case-sensitive, and a fatal error occurs if *VP* is not defined.

Limitations

Substitution is only performed in string values, and excludes the following cases:

- Identifier of a template descriptor definition

```
<server-template id="${invalid}" ...>
```

- Name of a variable definition

```
<variable name="${invalid}" ...>
```

- Name of a template parameter definition

```
<parameter name="${invalid}" ...>
```

- Name of a template parameter assignment

```
<server-instance template="T" ${invalid}="val" ...>
```

- Name of a node definition

```
<node name="${invalid}" ...>
```

- Name of an application definition

```
<application name="${invalid}" ...>
```

Substitution is not supported for values of other types. The example below demonstrates an invalid use of substitution:

```
<variable name="register" value="true"/>
<node name="Node">
    <server id="Server1" ...>
        <adapter name="Adapter1" register-process=${register} .../>
```


Copyright © 2017, ZeroC, Inc.

In this case, a variable cannot supply the value of `register-process` because that attribute expects a boolean value, not a string.

Most values are strings, however, so this limitation is rarely a problem.

Escaping a Variable

You can prevent substitution by escaping a variable reference with an additional leading $ character. For example, in order to assign the literal string `${abc}` to a variable, you must escape it as shown below:

```
<variable name="x" value="$${abc}"/>
```

The extra $ symbol is only meaningful when immediately preceding a variable reference, therefore text such as `US$$55` is not modified. Each occurrence of the characters $$ preceding a variable reference is replaced with a single $ character, and that character does not initiate a variable reference. Consider these examples:

```
<variable name="a" value="hi"/>
<variable name="b" value="$${a}"/>
<variable name="c" value="$$${a}"/>
<variable name="d" value="$$$${a}"/>
```

After substitution, b has the value `${a}`, c has the value `$hi`, and d has the value `$${a}`.

Special Descriptor Variables

IceGrid defines a set of read-only variables to hold information that may be of use to descriptors. The names of these variables are reserved and cannot be used as variable or parameter names. The table describes the purpose of each variable and defines the context in which it is valid.

Reserved Name	Description
`application`	The name of the enclosing application.
`application.distrib`	The pathname of the enclosing application's distribution directory, and an alias for `${node.datadir}/distrib/${application}`.
`node`	The name of the enclosing node.
`node.os`	The name of the enclosing node's operating system. On Unix, this is value is provided by `uname`. On Windows, the value is `Windows`.
`node.hostname`	The host name of the enclosing node.
`node.release`	The operating system release of the enclosing node. On Unix, this value is provided by `uname`. On Windows, the value is obtained from the `OSVERSIONINFO` data structure.
`node.version`	The operating system version of the enclosing node. On Unix, this value is provided by `uname`. On Windows, the value represents the current service pack level.
`node.machine`	The machine hardware name of the enclosing node. On Unix, this value is provided by `uname`. On Windows, the value can be x86, x64, or IA64, depending on the machine architecture.
`node.datadir`	The absolute pathname of the enclosing node's data directory.
`server`	The ID of the enclosing server.
`server.distrib`	The pathname of the enclosing server's distribution directory, and an alias for `${node.datadir}/servers/${server}/distrib`.
`service`	The name of the enclosing service.
`session.id`	The client session identifier. For sessions created with a user name and password, the value is the user ID; for sessions created from a secure connection, the value is the distinguished name associated with the connection.

Copyright © 2017, ZeroC, Inc.

The availability of a variable is easily determined in some cases, but may not be readily apparent in others. For example, the following example represents a valid use of the `${node}` variable:

XML

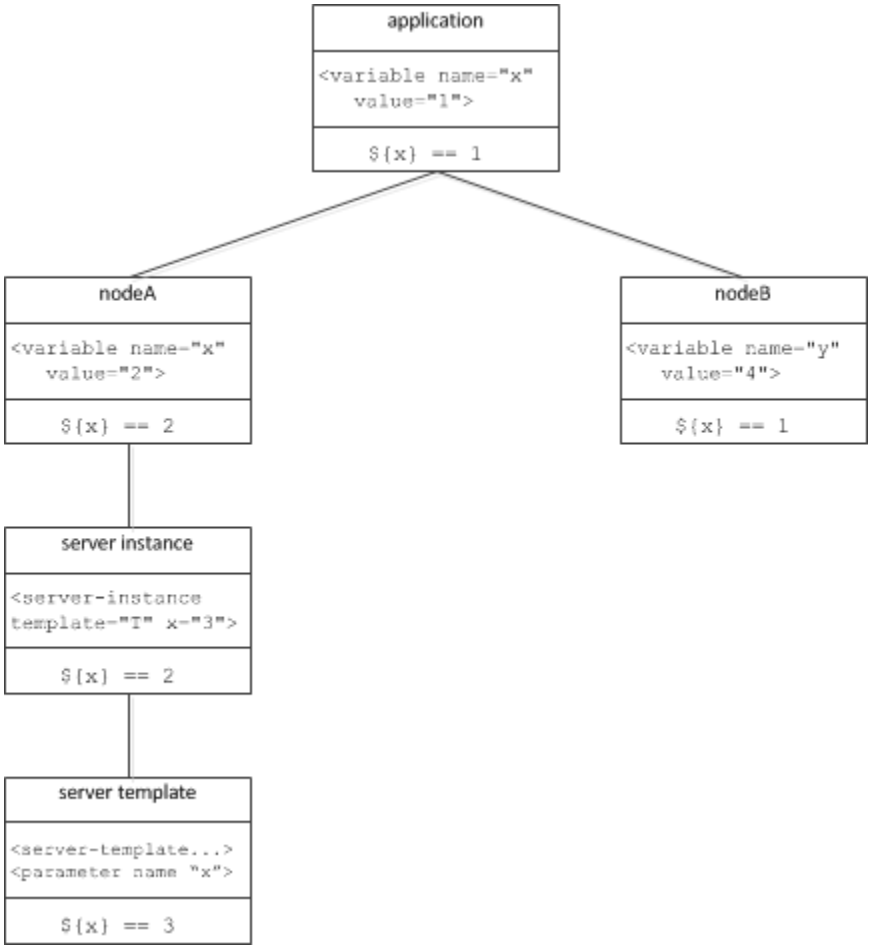
```
<icegrid>
    <application name="App">
        <server-template id="T" ...>
            <parameter name="id"/>
            <server id="${id}" ...>
                <property name="NodeName" value="${node}"/>
                ...
            </server>
        </server-template>
        <node name="TheNode">
            <server-instance template="T" id="TheServer"/>
        </node>
    </application>
</icegrid>
```

Although the server template descriptor is defined as a child of an application descriptor, its variables are not evaluated until it is instantiated. Since a template *instance* is always enclosed within a node, it is able to use the `${node}` variable.

Descriptor Variable Scoping Rules

Descriptors may only define variables at the application and node levels. Each node introduces a new scope, such that defining a variable at the node level overrides (but does not modify) the value of an application variable with the same name. Similarly, a template parameter overrides the value of a variable with the same name in an enclosing scope. A descriptor may refer to a variable defined in any enclosing scope, but its value is determined by the nearest scope. The following figure illustrates these concepts:

Copyright © 2017, ZeroC, Inc.



Variable scoping semantics.

In this diagram, the variable `x` is defined at the application level with the value `1`. In `nodeA`, `x` is overridden with the value 2, whereas `x` remains unchanged in `nodeB`. Within the context of `nodeA`, `x` continues to have the value 2 in a server instance definition. However, when `x` is used as the name of a template parameter, the node's definition of `x` is overridden and `x` has the value 3 in the template's scope.

Resolving a Reference

To resolve a variable reference `${var}`, IceGrid searches for a definition of `var` using the following order of precedence:

1. Pre-defined variables
2. Template parameters, if applicable
3. Node variables, if applicable
4. Application variables

After the initial substitution, any remaining references are resolved recursively using the following order of precedence:

1. Pre-defined variables
2. Node variables, if applicable
3. Application variables

Template Parameters

Template parameters are not visible in nested template instances. This situation can only occur when an IceBox server template instantiates a service template, as shown in the following example:

Copyright © 2017, ZeroC, Inc.

XML

```
<icegrid>
    <application name="IceBoxApp">
        <service-template id="ServiceTemplate">
            <parameter name="name"/>
            <service name="${name}" entry="DemoService:create">
                ...
                <property name="${name}.Identity" value="${id}-${name}"/> <!-- WRONG! -->
            </service>
        </service-template>
        <server-template id="ServerTemplate">
            <parameter name="id"/>
            <icebox id="${id}" endpoints="default" ...>
                <service-instance template="ServiceTemplate" name="Service1"/>
            </icebox>
        </server-template>
        <node name="Node1">
            <server-instance template="ServerTemplate" id="IceBoxServer"/>
        </node>
    </application>
</icegrid>
```

The service template incorrectly refers to `id`, which is a parameter of the server template.

Template parameters can be referenced only in the body of a template; they cannot be used to define other parameters. For example, the following is illegal:

XML

```
<server-template id="ServerTemplate">
    <parameter name="par1"/>
    <parameter name="par2" default="${par1}"/>
    ...
</server-template>
```

Modifying a Variable

A variable definition can be overridden in an inner scope, but the inner definition does not modify the outer variable. If a variable is defined multiple times in the same scope (which is only relevant in XML definitions), the most recent definition is used for all references to that variable. Consider the following example:

XML

```
<application name="MyApp">
    <variable name="x" value="1"/>
    <variable name="y" value="${x}"/>
    <variable name="x" value="2"/>
    ...
</application>
```

When descriptors such as these are created, IceGrid validates their variable references but does not perform substitution until the descriptor is acted upon (such as when a node is generating a configuration file for a server). As a result, the value of y in the above example is 2 because that is the most recent definition of x.

See Also

- Variable Descriptor Element
- IceGrid Templates
- Application Distribution
- Variables in IceGrid Descriptors

Copyright © 2017, ZeroC, Inc.

Copyright © 2017, ZeroC, Inc.