

---

# Some Thoughts on a Resource Specification Language for LCFGng

---

by Paul Anderson <paul@dcs.ed.ac.uk>

Division of Informatics  
University of Edinburgh

## 1 Introduction

The complete configuration of a machine and all its subsystems is described by a configuration *element* [3]. Each subsystem is configured by a *component* script which reads its configuration information from the configuration element.

☞ It isn't yet clear whether each component has access to the full element, or whether each component corresponds to a particular sub-element.

A resource *compiler* constructs the configuration element for a particular machine by combining configuration elements containing subsets of the information. The compiler supports *inheritance* allowing machine descriptions to be constructed by inheriting from various standard classes.

## 2 Resources

An individual configuration *resource* consists of a *key* and a *value*. This is usually expressed as an XML element:

```
<size>2000</size>
```

This allows arbitrary values, including multi-line data, but it must be encoded according to standard XML syntax (for example, < must be encoded as &lt;).

If a value attribute is supplied, this text is prepended to any XML data providing an alternative way to specify resource values:

```
<size/ value="2000">
```

## 3 Elements

A configuration *element* is an XML element representing one or more resources:

```
<partition>
```

```
    <size>100</size>
    <mount>/</mount>
</partition>
```

This allows arbitrary nesting and list representation:

```
<install>
  <disk>
    <partition>
      <size>100</size>
      <mount>/</mount>
    </partition>
    <partition>
      <size>free</size>
      <mount>/home</mount>
    </partition>
  </disk>
  ....
</install>
```

Data and sub-elements can appear in the same element but this is not recommended. All data elements within a single element are concatenated by the compiler to form a single string.

## 4 Naming

Elements may be given a *name*. This is preserved by the compiler and may be used by the component, but it is also used for matching elements when overriding [5].

```
<disk name="system">
  <partition name="root">
    ... </partition>
  <partition name="home">
    ... </partition>
</disk>
```

Elements without explicit names are implicitly named the same as the key.

Names do not have to be unique (although it is likely to be confusing if they are not unique within the enclosing element). Multiple occurrences of the same name

are assigned an *index* within the enclosing element to allow them to be uniquely identified.

```
disk/system/partition[2]
```

An element can be uniquely identified by its *URL*. This is formed from the URL of the server, and the *path* to the element:

```
http://lcfg.dcs.ed.ac.uk/  
disk/system/root
```

If the URL is omitted, then some default search path is assumed.

☞ The default URL should probably be the URL of the enclosing element if there one, otherwise some default search path.

☞ We would still like some way of referencing local names, rather than always referring to global names.

## 5 Inheritance

An element can *inherit* resources from another element by specifying the path of the element to be inherited:

```
<partition/  
  ref="disk/system/root">
```

The compiler will create a new element with the key `partition` whose sub-elements are copies of the sub-elements of the partition `disk/system/root`.

The `ref` attribute does not appear in the compiler output and is not available to the configuration component.

If sub-elements are specified, they will *override* sub-elements of the inherited element as follows (this default behaviour may be modified by special attributes [6]):

- ❑ All inherited sub-elements are copied to the new element.
- ❑ Specified sub-elements which have the same name and index as an inherited sub-element will replace the corresponding inherited sub-element: data will be replaced directly by any specified data and this algorithm will be applied recursively to process any nested elements.
- ❑ Specified sub-elements which have no corresponding sub-element in the inherited element will be added to the end of the end of the inherited sub-elements. The index will be changed if

necessary to reflect the position within the new element.

The following example defines a big root partition. All resources apart from the `size` are the same as the inherited root partition.

```
<partition name="bigroot"  
  ref="disk/system.root">  
  <size>3000</size>  
</partition>
```

Notice that this `size` resource *replaces* the corresponding resource from the `root` partition because they both have the implied name `size`.

Inheritance references can be chained in the obvious way; the above algorithm is used to evaluate the inherited element before applying it to the element being defined:

```
<partition ref="bigroot">  
  <opt>ro</opt>  
</partition>
```

## 6 Actions

The default behaviour for overriding inherited elements can be changed by supplying an explicit `action` attribute with the resource:

**append** - the specified element is appended to the end of the inherited elements, even if there is an inherited element with the same name and index. The index of the appended resource is adjusted to reflect its position in the new element.

**delete** - any inherited element with the same name and index is deleted. Data associated with the specified resource is ignored.

☞ It isn't clear whether the indices of the remaining elements should be adjusted when an element is deleted. I think not.

**replace** - the specified element replaces any corresponding inherited element. If there is no corresponding inherited element, the specified element is ignored.

**mutate** - the value associated with the specified element is treated as a fragment of Perl code to be applied to the value of any corresponding inherited element to obtain the value of the new element. The variable `$_` contains the inherited value on entry and should contain the new value on exit. If

there is no corresponding inherited resource, then  
`$_ = undef.`

The `action` attribute does not appear in the output of the compiler.

In this case, the sub-elements potentially inherit from a corresponding sub-element of the inherited element as well as any explicit references. For example, the `root` sub-element inherits from the `root` sub-element of the disk `system` in addition to any explicit references that might have been specified for the partition. This is treated as a multiple inheritance with any explicit references taking precedence.

## 7 Multiple Inheritance

Multiple inheritance can be specified by including multiple reference attributes. However, simply evaluating the inherited elements and then applying the overrides in sequence does not give the required result if the inherited elements share a common ancestor.

To evaluate multiple references, the compiler flattens the inheritance trees for all the specified references (in postfix order) and appends the results. If the same element occurs more than once in this list (as defined by path equality), all but the first instance are removed. The elements in the list are then applied as successive overrides.

```
<partition name="roroot"
      ref="root">
  <opt>ro</opt> </partition>

<partition name="bigroot"
      ref="root">
  <size>3000</size> </partition>

<partition/ ref="roroot"
      ref="bigroot">
```

Note that the last specified reference takes precedence over earlier ones.

## 8 Overriding Compound Elements

Frequently, an element to be inherited will contain compound sub-elements rather than simple resources:

```
<disk ref="system">

  <partition name="root">
    <size>3000</size>
  </partition>

  <partition name="opt">
    <size>1000</size>
    <mount>/opt</mount>
  </partition>

</disk>
```