Workgroup:	netmod
Internet-Draft:	draft-rajaram-netmod-yang-cfg-template-framework-00
Published:	21 October 2024
Intended Status:	Informational
Expires:	24 April 2025
Authors:	R. Peschi	S. Ashraf	D. Rajaram
---	---	---	---
	Nokia	*Nokia*	*Nokia*

Populating a list of YANG data nodes using templates

Abstract

This document presents a modeling technique for configuring large scale devices in a compact way, when the device contains many similar data node patterns. A device here refers to any such entity that acts as a netconf server. This is realized by instructing the device to locally generate repetitive patterns of data nodes from a master copy called 'template' that is configured in the device. This approach is both convenient and efficient, as it minimizes the size of the running data store and reduces network provisioning time. It leverages existing YANG specification features and can be implemented with standard, off-the-shelf tools.

The RFC Editor will remove this note

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rajaram-netmod-yang-cfg-template-framework/.

Discussion of this document takes place on the netmod Working Group mailing list (mailto:netmod@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netmod/. Subscribe at https://www.ietf.org/mailman/listinfo/netmod/.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 April 2025.

Copyright Notice

Table of Contents

1. Introduction

This draft considers the case of a device that contains a functional entity, characterized by a well-defined data nodes pattern, that is massively replicated and where each replication instance needs individual configuration with only limited variation.

Having a device manager that repetitively configures each data node for every functional instance can become complex and prone to errors. This approach may lead to issues, such as extended configuration times, increased memory usage on the device, and inefficient YANG validation processes due to the large size of the running data store. These challenges only intensify as the system scales.

This draft proposes a technique to improve this, which is based on 'YANG templates' that results in a smaller running data store even when the device is very large.

A 'YANG template' is the configuration of a functional entity that the device is instructed to replicate multiple times to generate copies of the entity. The technique that we address in this draft allows to generate copies with the same data node values as in the template with the possibility, though, to overrule some of these values on an individual copy basis.

The general template technique detailed in this draft does not suffer from the drawbacks mentioned earlier where the device manager is explicitly configuring all device data nodes.

This method is not aimed at becoming a standard itself. It is a YANG modeling technique that can be considered, where appropriate, by any implementor or standardization organization, when designing a YANG module for a specific purpose. For the sake of completeness, note that other IETF documents do mention already the use of YANG template concept for specific applications, for instance draft-ietf-ccamp-optical-impairment-topology-yang and RFC8795. However, they don't provide full context about the method. This draft can help clarifying and formalizing in a generic framework how to define and use YANG templates.

The next sections will elaborate on some of the constructs to apply the template technique and highlight how to mitigate some potential issues. More specifically,

- How to define the template of a functional entity, how to instruct the device to replicate it (including per instance variations) and
- How the replication process takes place when expanding the template into functional entity instances configuration data nodes.

2. Template technique framework

2.1. Provisioning the running data store

This section outlines how the device's running data store is utilized to implement the template technique.

In the YANG model of the device, many functional instances can be organized in a list. Since a template represents the typical configuration pattern of a functional instance, it is often necessary to choose between multiple templates for replication. For Example: the device may host various types of functional instances, each with its own specific data structure. Say, one template might define data nodes for a 'function-1' type, while another template could define data nodes for a 'function-2' type. For a given type of functional instance, different sets of values for the data nodes may be required. Say, a 'function-1' type might have templates for 'function-1-bronze-grade', 'function-1-silver-grade', and 'function-1-gold-grade' variants. As a result, multiple templates can also be organized into a list within the device's YANG model.

Assume for example, that list-a and list-b data nodes need to be configured in the device. Traditionally, the tree structure will be as below:

root

```
+--(...)                     // out of scope
+--rw data-nodes-pattern     // container for functional instances
   +--rw instance* [name]    // the list of functional instances
      +--rw name             string-ascii64
      +--rw description?     string-ascii128
      +--rw data
         +--rw list-a [name] //e.g. a list of interfaces
         |  +--rw name
         |  +--rw parm-x
         |  +--rw parm-y
         +--rw list-b [name] //e.g. a list of hardware components
            +--rw name
            +--rw parm-t
            +--rw parm-u
```

In contrast, the YANG tree of a device using the template technique would appear as below:

root

```
+--(...)                      // out of scope
+--rw data-nodes-pattern      // container for functional instances
   +--rw template* [name]     // the list of templates
   |  +--rw name              string-ascii64
   |  +--rw description?      string-ascii128
   |  +--rw data
   |     +--rw list-a [name] //e.g. a list of interfaces
   |     |  +--rw name
   |     |  +--rw parm-x
   |     |  +--rw parm-y
   |     +--rw list-b [name] //e.g. a list of hardware components
   |        +--rw name
   |        +--rw parm-t
   |        +--rw parm-u
   +--rw instance* [name]     // the list of functional instances
      +--rw name              string-ascii64
      +--rw description?      string-ascii128
      +--rw template?         -> /data-nodes-pattern/template/name
      +--rw data
      +--rw list-a [name] //e.g. a list of interfaces
      |  +--rw name
      |  +--rw parm-x  //overrule if present,or take it from template
      |  +--rw parm-y  //overrule if present,or take it from template
      +--rw list-b [name] //e.g. a list of hardware components
         +--rw name
         +--rw parm-t  //overrule if present,or take it from template
         +--rw parm-u  //overrule if present,or take it from template
```

Each entry in the template list encompasses the generic configuration data nodes that are needed for all the functional entities to be addressed by this template, as contained in the data container, in this example, data nodes of list-a and list-b. In practice, naturally, the more data nodes that can be replicated the more efficient the template technique will be.

Each entry in the instance list represents a copy to be made of the data nodes pattern defined by the leaf-ref template, to create a functional entity instance. The data container contains all the data nodes needed to customize each copy of the template, by overruling one or the other data node value originating from the template (eg: parm-x, parm-y, parm-t, parm-u), or possibly to add to the copy one or the other data nodes not provided by the template.

Although it is recommended that the same data nodes are defined in the data container of template and the data container of instance, only a limited number of such data nodes should be configured in the instance. This reflects the assumption that functional instances should have only limited variations from their template model.

It should be noted that in a good application of the template technique, only few templates would suffice to generate a very large number of functional instances; in other words, the instance list would be much larger, typically by order of magnitudes, than the template list.

A simple configuration example in the running data store of the device can be found at Appendix A.1 (Section 8.1.1)

2.2. Device expanding the running data store

Once the configuration is applied to the device, the device will dynamically create each instance as specified. The process for generating data nodes for a particular instance follows these steps:

-A copy of the template's data nodes is made, serving as the foundation for the instance's configuration.

-If any of these data nodes are also configured in the running data store for this instance, they will override the template values.

If an instance data node is configured in the running data store but not provided by the template, it will be added to the generated instance configuration.

The resulting instance expansion corresponding to the example in appendix A.1 is provided in Appendix A.2 (Section 8.1.2)

2.3. Mandatories and Defaults

While conceptually, the idea of templates improves the re-usability and consistency factor, there are certain nuances, that need to be addressed while handling data nodes with defaults and mandatories.

If certain replicable data patterns contain default or mandatory values, and are used as-is both in the template and in the instance,

-There is a possibility of silent and unintentional overwriting the configured value of the node in the template with the default value in the instance due to the merge operation.

-Mandatory data nodes must be unconditionally configured in the instance although they are already configured in the template, reducing the efficiency of the template mechanism.

Hence, while the same data nodes are used in the templates and instances, it is imperative that instance data nodes are without default and mandatory statements.

there may be different implementation to solve this, one such way that uses the existing YANG constructs is provided in Appendix B (Section 8.2).

3. Benefits of Templates in YANG Design

1. Reusability: Templates encourage reusability by allowing common structures to be defined once and reused multiple times. This reduces duplication and simplifies model management.
2. Consistency: By using templates, similar configurations are defined and applied uniformly, leading to more consistent data modeling and network configuration management.
3. Simplified Maintenance: When a template is modified, all places where it is used will automatically reflect those changes, reducing the effort required to maintain and update the model.

4. Modularity: Templates promote a modular approach to YANG model design. By splitting the model into reusable parts, it becomes easier to scale and extend.

4. Conclusion

Using templates in YANG allows to efficiently configure large amounts of similar data nodes while keeping the running data store size small. This is beneficial in term of device memory footprint, ease of configuration, configuration time and potentially YANG validation processing in the device. This draft explains some practicalities of the template method, including how to ensure that mandatory and default statements do not jeopardize the effectiveness of the method.

5. Conventions and Definitions

6. Security Considerations

TODO Security

7. IANA Considerations

This document has no IANA actions.

8. Appendix

8.1. A. Example of applying the template method.

8.1.1. A.1.Device running data store using the template mechanism

In this example, one template template-1 is configured, and three instances are configured, to be derived from template-1 and with limited overruling of the template values.

```
 <config>
 <data-nodes-pattern>
 <template>
  <name>template-1</name>
  <description>A typical configuration</description>
  <data>
    <list-a>
      <name>templ-1-list-a-entry-1</name>
      <parm-x>1</parm-x>
      <parm-y>30</parm-y>
    </list-a>
    <list-a>
      <name>templ-1-list-a-entry-2</name>
      <parm-x>3</parm-x>
      <parm-y>30</parm-y>
    </list-a>
     <list-a>
      <name>templ-1-list-a-entry-3</name>
      <parm-x>3</parm-x>
      <parm-y>50</parm-y>
    </list-a>
    <list-b>
      <name>templ-1-list-b-entry-1</name>
      <parm-t>2</parm-t>
      <parm-u>40</parm-u>
    </list-b>
    <list-b>
      <name>templ-1-list-b-entry-2</name>
      <parm-t>4</parm-t>
      <parm-u>60</parm-u>
    </list-b>
    <list-b>
      <name>templ-1-list-b-entry-3</name>
      <parm-t>4</parm-t>
      <parm-u>80</parm-u>
    </list-b>
  </data>
</template>

<instance>                          // a first instance
  <name>instance-1</name>
  <template>template-1</template>  //config is derived from template-1
  <data>
    <list-a>
      <name>templ-1-list-a-entry-2</name> // inherited from template
      <parm-y>33</parm-y>      //overrule template value 30 with 33
    </list-a>
  </data>
</instance>

<instance>                          // a second instance
  <name>instance-2</name>
  <template>template-1</template> //config is derived from template-1
  <data>                    // nothing from template to be overruled
  </data>
</instance>
```

```
<instance>                          // a third instance
  <name>instance-3</name>
  <template>template-1</template> //config is derived from template-1
  <data>
    <list-a>
      <name>templ-1-list-a-entry-3</name> //inherited from template
      <parm-y>55</parm-y>      //overrule template value 50 with 55
    </list-a>
    <list-b>
      <name>templ-1-list-b-entry-3</name> //inherited from template
      <parm-u>88</parm-u>       //overrule template value 80 with 88
    </list-b>
  </data>
</instance>
 </data-nodes-pattern>
 </config>
```

8.1.2. A2:Data generated by the template mechanism

The running data store example in section A.1 leads the device to generate the following data used to control the instances (without the aid of the template mechanism, this data would need to explicitly come from the running data store, instead of being locally expanded):

generated through instance-1 merged with template-1 expansion

```
<data>
  <list-a>
    <name>templ-1-list-a-entry-1</name>
    <parm-x>1</parm-x>
    <parm-y>30</parm-y>
  </list-a>
  <list-a>
    <name>templ-1-list-a-entry-2</name>
    <parm-x>3</parm-x>
    <parm-y>33</parm-y>     //deviate from template value
  </list-a>
   <list-a>
    <name>templ-1-list-a-entry-3</name>
    <parm-x>3</parm-x>
    <parm-y>50</parm-y>
  </list-a>
  <list-b>
    <name>templ-1-list-b-entry-1</name>
    <parm-t>2</parm-t>
    <parm-u>40</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-2</name>
    <parm-t>4</parm-t>
    <parm-u>60</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-3</name>
    <parm-t>4</parm-t>
    <parm-u>80</parm-u>
  </list-b>
</data>
```

(generated through instance-2 merged with template-1 expansion)

```
<data>
  <list-a>
    <name>templ-1-list-a-entry-1</name>
    <parm-x>1</parm-x>
    <parm-y>30</parm-y>
  </list-a>
  <list-a>
    <name>templ-1-list-a-entry-2</name>
    <parm-x>3</parm-x>
    <parm-y>30</parm-y>
  </list-a>
   <list-a>
    <name>templ-1-list-a-entry-3</name>
    <parm-x>3</parm-x>
    <parm-y>50</parm-y>
  </list-a>
  <list-b>
    <name>templ-1-list-b-entry-1</name>
    <parm-t>2</parm-t>
    <parm-u>40</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-2</name>
    <parm-t>4</parm-t>
    <parm-u>60</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-3</name>
    <parm-t>4</parm-t>
    <parm-u>80</parm-u>
  </list-b>
</data>
```

(generated through instance-3 merged with template-1 expansion)

```
<data>
  <list-a>
    <name>templ-1-list-a-entry-1</name>
    <parm-x>1</parm-x>
    <parm-y>30</parm-y>
  </list-a>
  <list-a>
    <name>templ-1-list-a-entry-2</name>
    <parm-x>3</parm-x>
    <parm-y>30</parm-y>
  </list-a>
   <list-a>
    <name>templ-1-list-a-entry-3</name>
    <parm-x>3</parm-x>
    <parm-y>55</parm-y>     //deviate from template value
  </list-a>
  <list-b>
    <name>templ-1-list-b-entry-1</name>
    <parm-t>2</parm-t>
    <parm-u>40</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-2</name>
    <parm-t>4</parm-t>
    <parm-u>60</parm-u>
  </list-b>
  <list-b>
    <name>templ-1-list-b-entry-3</name>
    <parm-t>4</parm-t>
    <parm-u>88</parm-u>     //deviate from template value
  </list-b>
</data>
```

8.2. B: Using existing YANG constructs in template and instance YANG definition

This appendix illustrates the use of groupings in the YANG definition of template and instances and more specifically it shows how easily mandatory and default statements can be introduced in the template definition by refining the grouping uses statement.

8.2.1. The grouping construct

By defining common structures using grouping, one avoids repeating code, ensures consistency and makes future changes easier since modifications only need to happen in one place.

Example:

```
grouping interface-config {
          leaf parm-a {
             type string;
          }
          leaf parm-b {
             type boolean;
          }
          leaf parm-c {
             type unint32;
          }
        }
```

This 'interface-config' grouping defines a common structure that can be reused across different YANG modules or different parts of the same module.

8.2.2. The uses construct

The 'uses' statement applies a previously defined grouping where needed in the model (e.g. it copies the data nodes of the grouping at the place of the 'uses' statement.

As an example, the data nodes defined in the grouping above can be used in the template and in the instance definition:

```
 container data-nodes-pattern {
 list template {
  key "name";
  leaf name {
    type string;
  }
  container data {
    list interface {
      key "interface-name";
      leaf name {
        type string;
      }
      uses interface-config;
    }
   }
  }
list instance {
  key "name";
  leaf name {
    type string;
  }
  container data {
    list interface {
      key "interface-name";
      leaf name {
        type string;
      }
      uses interface-config;
    }
   }
  }
 }
```

8.2.3. The refine construct to control default and mandatory statements

As explained in section 2.3, with the template method, some data nodes may need a default or mandatory statement when used for the template definition but should have no mandatory neither default statements when used for the instance definition.

The refine statement available in existing YANG version easily allows to control mandatory and default statements when used along with the uses statement.

Assume in our example that it is desired that parm-b has a default statement and parm-c has a mandatory statement when they are used for the template definition.

Then the YANG becomes:

```
 container data-nodes-pattern {
 list template {
  key "name";
  leaf name {
    type string;
  }
  container data {
    list interface {
      key "interface-name";
      uses interface-config {
        refine parm-b {
          default "true"
        }
        refine parm-c {
          mandatory true
        }
      }
    }
  }
}
list instance {
  key "name";
  leaf name {
    type string;
  }
  container data {
    list interface {
      key "interface-name";
      uses interface-config;
    }
   }
  }
 }
```

Acknowledgments

Authors' Addresses

Robert Peschi
Nokia
Antwerp
Email: robert.peschi@nokia.com

Shiya Ashraf
Nokia
Antwerp
Email: shiya.ashraf@nokia.com

Deepak Rajaram
Nokia
Chennai
Email: deepak.rajaram@nokia.com