Netmod Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: December 18, 2018

Q. Wu
Huawei
A. Farrel
Juniper Networks
B. Claise
Cisco Systems, Inc.
June 16, 2018

Documentation Conventions for lines wrapping and indentation in authored work

draft-wu-netmod-yang-xml-doc-conventions-05

Abstract

Many documents that define YANG modules or YANG fragments also include protocol message instance data examples.

IETF documentation has specific limits on line length (73 characters) and some YANG fragment example or protocol message instance data examples such as XML encoded YANG data node instance examples have to include line wraps that would not normally be allowed according to the XML representation rules of RFC7950 and RFC7952.

This document lays out documentation conventions that allow authored work to be presented in IETF documentation when authored work such as YANG fragment or protocol message instance data example would otherwise exceed the maximum line length and provide consistent representation of authored work within an Internet-Draft or RFC. There are no implications in this document for YANG tools: this document does not change the rules for presenting authored work in data files or in the wire.

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 December 18, 2018.

Table of Contents

1. Introduction

When documenting authored work such as YANG fragments example of example of YANG module represented in XML encoding it is possible that the representation of these authored work will exceed the available line length. Indentation may further aggravate this issue. The line wrapping is needed for formatting purposes, however different document author may take different ways to wrap line which

makes difficult to improve the readability and interoperability of published YANG data models.

This document lays out documentation conventions that allow authored work to be presented in IETF documentation when authored work such as YANG fragment or protocol message instance data example would otherwise exceed the maximum line length and provide consistent representation of authored work within an Internet-Draft or RFC.

Document conventions defined in this document are not representative of how the Authored work must be presented to a software component or carried on the wire. There are no implications in this document for YANG tools(e.g., libyang parser): this document does not change the rules for presenting YANG models or for encoding YANG in data files or in the wire.

2. Conventions Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950] and are not redefined here:

- data node
- leaf
- leaf-list
- instance

The following term is defined in [RFC7951] and [RFC7952] and are not redefined here:

- data node Instance
- data node identifier

The following terms are defined in [RFC8340]and [I-D.ietf-netmod-rfc6087bis].

2.1. Glossary of New Terms

Authored work: A set of text format work representing YANG fragments, and protocol message instance data except YANG Tree Diagrams.

Wrap: Convert authored work with long lines not fitting into an Internet-Draft or RFC into authored work with split line fitting into an Internet-Draft or RFC.

Unwrap: Re-Convert authored work with split line fitting into an Internet-Draft or RFC back to valid authored work without split line that can be consumed by a software component or carried on the wire.

Indent: used to describe the distance, or number of blank spaces used to separate a paragraph from the left or right margins.

Libyang parser: YANG tool and library for parsing and validating YANG schemas and instance data.

3. Long line wrapping Example

An example of the documentation of a leaf node is shown in Figure 1. The container node is called <parent-node-label>, any whitespace, carriage returns, or line feeds between the subelements <parent-node-label> is insignificant, i.e., an implementation MAY insert whitespace, carriage return, or line feed characters between subelements. The leaf is called "long-leaf-node-label" and is assigned the value "long-leaf-node-value". As can be seen in the example, this fits on one line. However it would only take the addition of a few more characters to the node label or value for the example to overflow the 73 character limit if the line of leaf node instance is indented (e.g., start below <parent-node-label> with a whitespace offset of two characters. .

```
<parent-node-label>
  <long-leaf-node-label>long-leaf-node-value</long-leaf-node-label>
</parent-node-label>
```

Figure 1: A Simple Leaf Node Example

For the sake of documentation purpose, the representation shown in Figure 2 SHALL be considered as equivalent to that shown in Figure 1, but when a document uses this convention it MUST also include the text shown in Figure 3. Note that the first example representation in figure 2 is more easily parsed by a human reader than the second example in figure 2.

```
<parent-node-label>
  <long-leaf-node-label>\
    long-leaf-node-value\
  </long-leaf-node-label>
</parent-node-label>
Or
<parent-node-label>
  <long-leaf-node-label> long-leaf-node-value </long-leaf-nod\
    e-label>
</parent-node-label>
```

Figure 2: A Split Leaf Node Example

4. Objectives

In order to allow authored work to be presented in IETF documentation when authored work such as YANG fragment or protocol message instance data example would otherwise exceed the maximum line length and provide consistent representation of the authored work within an Internet-Draft or RFC, the following design criteria are used:

- Allow automatic wrapping line when any line presented in the authored work of I-D or RFCs exceed the maximum line length.

- Allow automatic unwrapping line in the artwork when the artwork needs to be presented to a software component or carried on the wire.

5. Line wrapping and indentation document convention

When the representation of an authored work (e.g.,a leaf node instance representation) in an example would result in a line being longer than the maximum line length for an IETF document the long line must be split and presented on more than one lines. The new line may be indented, if necessary, so that it starts below the first line with a whitespace offset of two characters, which improve readability and interoperability of published YANG data models.

When these authored work with split lines needs to be fed into software component or carried in the wire, these authored work with split lines should be unwrapped and reversed into the valid authored work with long line. If the indentation is applied to authored work with split lines, the indentation should be removed during unwrapped process.

5.1. Long line wrapping

Long line wrapping most likely to happen when the authored work example such as leaf node contains built-in type string or datetime or container node and list node includes metadata attributes. Indeed, if this problem arises for other YANG types it may be indicative of poorly chosen YANG type values, and the YANG definitions should be revised before applying document convention for line wrapping defined in this document.

In the case of long line exceeding 73 characters, the following long line wrapping conventions MUST be observed:

- Split long line in the authored work (e.g.,leaf node instance, YANG data node instance containing metadata annotation attributes) exceeding 73 characters limits with the backslash ("\") and use backslash ("\")to indicate wrapping at the end of the line. The broken line MUST be terminated with a backslash ("\") without the addition of any additional space before the backslash and with no further characters after the backslash.

- Any continuation lines or new line MUST align with the first line and MAY chose be indented with two whitespace offset for readability purposes.

- When a backslash appears in any line not used for split line, the representation of this artwork MUST be arranged so that this backslash is not the final character of a broken line. If this backslash is the second last character (e.g., backslash at the position 72) of a broken line, the line should be split at the position one or several characters before this backslash as the second last character with the backslash ("\") . In extreme case, if a long line is full of backslashes, the backslashs before backslash at position 73 in this line should be treated in the same way as other normal characters.

Furthermore, whenever a document uses long line wrapping conventions it MUST also include the following boilerplate text :

```
[!!! '\' line wrapping is for formatting only and adopt the conventions
shown in BCPXX [RFCYYYY]]
<WRAPPED TEXT BEGIN>
 ....//Authored work
<WRAPPED TEXT END>
RFC Editor Note: Please replace XX and YYYY with the numbers assigned
for this document.
```

Figure 3

Figure 4 shows an example of Backslash appearing in the long line not used for split line.

```
<long-leaf-complex-string-node-label>Punctuation is important. As
are line feeds.Some characters are special,e.g., the backslash\.
Don't forget. </long-leaf-string-node-label>
```

Figure 4: An Example Leaf Node With a Complex String Value

Figure 5 shows a semantically equivalent representation of the example.

```
<long-leaf-complex-string-node-label>Punctuation is important. As \
are line feeds.Some characters are special,e.g., the backslash \.\
Don't forget.</long-leaf-string-node-label>
```

Figure 5

5.2. Line unwrapping

If line wrapping is done for formatting purposes, the line wrapping in the authored work should be reversed back or unwrapped before the authored work is fed into software component for validation or carried in the wire. Therefore line unwrapping help remove backslash and additional carriage return or line feed character and make unwrapped authored work to be effectively compliant with the tool. The line wrapping for formatting purpose is indicated by the above boilerplate text in Figure 3. To unwrap line, the following conventions must be observed:

- Consecutive split lines in the authored work with backslash at the end of the line should be merged into one long line, the last split line in Consecutive split lines should not be terminated with backslash.

- If a backslash character ("\") doesn't appear at the end of the line within authored work, it should not be stripped.

- If a backslash character ("\") appears at the end of the line within authored work, it should be stripped. In the meanwhile, if and only if it is immediately followed by a carriage return or line feed character then all carriage return, line feed, and whitespace characters should be stripped until the next character is encountered.

- In extrem case, if a backslash character ("\") or space character appears full of line, the full line of backslash character ("\") or space character should be stripped.

5.3. Auto indentation and dedentation

Consistent indentation should be used for all authored work in the I-D and RFCs, e.g., if a space or tab characters are used to index the text in the long line during wraping process, the space and tab characters used for indentation should be removed during unwrapping process. If the new line or continuation line indented with a whitespace offset of two characters during wrapping process, the indentation with a whitespace offset of two characters should be removed during unwrapping process.

6. Limitation and complexity

6.1. Limitations

All modules need to be extracted YANG modules from an Internet Draft or RFC and then validated before submission in an Internet Draft. However we don't have automation tool to extract authored work such as YANG fragments or protocol message instance. To extract authored work, the similar strings "<CODE BEGINS>" and "<CODE ENDS>" MUST be defined and populated to identify each authored work component, e.g., the boilerplate text in Section 5 can be used to indicate the begining of authored work.

Applying wrapping and unwrapping functionality to example YANG module or YANG module extracted using existing tool also has limitation, even introduce confusion, e.g.,

1. The data definition description statement has long line exceeding 73 characters, it should be wrapped without using backslash as termination point.

```
"
     grouping link-ref {
       description
         "This grouping can be used to reference a link in a specific
          network.  Although it is not used in this module, it is
          defined here for the convenience of augmenting modules.";
"
```

2. Another example is when a plus character ("+") is used to concatenate two quoted string into one string, using backslash to split the line Confuses with using a plus character ("+") to split the line.

```
"
  container dhcp-relay {
    when "derived-from-or-self(../address-allocation-type, "+
    "'l3vpn-svc:provider-dhcp-relay')" {
      description
      "Only applies when provider is required to implement
      DHCP relay function.";
   }
"
```

6.2. Complexity

We can build tool to support auto wrap and auto indentation. However if the tool is designed to understand various encodings, e.g., XML encoding, JSON encoding or metadata annotation, it adds a lot of complexity to build such tool, therefore the only choice to make tool understand various encodings, is to build encoding specific tool which doesn't scale well,e.g., if the tool understands metadata annotation, we can decide where to insert backslash to split the lines: either inserted between metadata Attributes or insert at any place when the long line exceeding 73 characters limits. See more complexity details in Appendix A.

7. Security Considerations

There is no direct security impact related to the documentation convention for lines wrapping and indentation in authored work described in this document. However, attempting to provide representation of authored work using the documentation conventions described in this document would have unpredictable results. The risk here is that someone uses an example as a template for actual authored work representation. The mandatory boilerplate text provides a mitigation against this risk.

8. IANA Considerations

There are no IANA requests or assignments included in this document.

9. Acknowledgements

Thanks to Kent Watsen for discussions that kept us close to being on the right track. Additional thanks to John Scudder for flagging some nits, Martin Bjorklund, Charles Eckel, Robert Wilton and many others for valuable comments and review, special thanks Xiongjie to help support automation tool building.

10. Normative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-20 (work in progress), March 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

Appendix A. Representing XML and JSON Encodings of Metadata Annotations

[RFC7952] section 5.1 and section 5.2 provide an encoding rule for metadata annotations in XML and JSON respectively.

When an example XML representation of a leaf node element that includes metadata attributes results in a line being longer than the maximum number of characters allowed in a line of an IETF document, the value of the leaf node must be split across more than one line.

Where possible, all line breaks should be inserted between metadata attributes. Continuation lines MUST align with the first line and not be indented with any whitespace. The leading and trailing whitespace of each line MUST be ignored. Figure 6 gives a XML example.

When an example JSON representation of a leaf node element that includes metadata attributes starting with the "@" character results in a line being longer than the maximum number of characters allowed in a line of an IETF document,the value of the leaf node must be split across more than one line. Continuation lines MUST align with the first line and indented with one whitespace character. The leading and trailing whitespace of each line MUST be ignored. Figure 7 gives a JSON example.

Whenever this documentation convention is used, the boilerplate text shown in Figure 3 MUST be present in the document using the convention.

```
<foo xmlns:elm=http://example.org/example-last-modified\
elm:last-modified="2015-09-16T10:27:35+02:00">
  ...
</foo>
```

Figure 6: An XML Example Leaf Node With Metadata Split Across Lines

```
"cask": {
  "@": {
    "example-org-example-last-modified:last-modified":\
    "2015-09-16T10:27:35+02:00"
  },
  ...
}
```

Figure 7: A JSON Example Leaf Node With Metadata Split Across Lines

Appendix B. Auto-wrapping tool code

We provide examples of python code for aspects of line wrapping and unwrapping algorithms. There may be other implementation methods that are faster in particular operating environments or have other advantages. These implementation notes are for informational purposes only and are meant to clarify the this specification for line wrapping and unwrapping.

```
#!/usr/bin/env python2.7
# -*- coding: utf-8 -*-
"""Qin Wu, 2018-06-02
```

Autowrapper.py uses Text Wrap Module as library and support auto wrap and auto indent
two functionalities.
(1)Lines with "\" in position 72 have been handled.
(2)Lines with space in position 73 have been handled.
(3)A line of "\" has been handled.
(4)A line of space has been hanled.
https://github.com/sunseawq/auto-wrap-indent/blob/master/autowrapper.py
Text Wrap module provides two convenience functions, wrap() and fill(), as well as
TextWrapper, the class that does all the work, and a utility function dedent(). If
you're just wrapping or filling one or two text strings, the convenience functions
should be good enough; otherwise, you should use an instance of TextWrapper for efficiency.

https://github.com/python/cpython/blob/2.7/Lib/textwrap.py
"""

```
import textwrap
import string, re
import argparse
import os.path
import sys, getopt

def indent(text, prefix, predicate=None):
    """Adds 'prefix' to the beginning of selected lines in 'text'.

    If 'predicate' is provided, 'prefix' will only be added to the lines
    where 'predicate(line)' is True. If 'predicate' is not provided,
    it will default to adding 'prefix' to all non-empty lines that do not
    consist solely of whitespace characters.
    """
    if predicate is None:
        def predicate(line):
            return line.strip()

    def prefixed_lines():
        for line in text.splitlines(True):
            yield (prefix + line if predicate(line) else line)
    return ''.join(prefixed_lines())

def auto_wrap(input_file, dst_file):
    finput=open(input_file, "r")
    alllines=finput.readlines()
    finput.close()
    foutput = 0
    output_file = dst_file
```

```
    foutput = open(output_file, 'a')
    for eachline in alllines:
        bc = textwrap.fill(eachline,73)
        tmplines = bc.split('\n')
```

```
        tmplen = len(tmplines)
        if tmplen == 1 :
            foutput.writelines(bc)
            foutput.writelines('\n')
        else :
            i = 0
            while i < tmplen-1 :
                foutput.writelines(tmplines[i])
                foutput.writelines('\\')
                foutput.writelines('\n')
                i += 1
                foutput.writelines(tmplines[tmplen-1])
                foutput.writelines('\n')
    foutput.close

def auto_unwrap(input_file, dst_file) :
         finput=open(input_file, "r")
         alllines=finput.readlines()
         finput.close()
         foutput = 0
         output_file = dst_file
         foutput = open(output_file, 'a')
         for eachline in alllines:
             if eachline.endswith('\\\n') :
                 eachline = eachline.strip('\\\n')
             foutput.writelines(eachline)

def auto_wrap_indent(input_file, dst_file,width):
    finput=open(input_file, "r")
    alllines=finput.readlines()
    finput.close()
    foutput = 0
    flag_add = 0
    backslashpos = 0
    output_file = dst_file
    foutput = open(output_file, 'a')
    for eachline in alllines:
        backslashpos = eachline.rstrip('\\\n').rfind('\\',0,width)
        '''handle backslash at position 72'''
        if (backslashpos == width-1) :
            print("backslash appear at the end of the line,
            the line is wrapped at the position one or multiple characters
            before the backslash")
            bc = textwrap.fill(eachline,width-1)
        else :
            bc = textwrap.fill(eachline,73)
        '''handle space at position 71,72,73'''
        if eachline.rstrip('\n').rfind(' ',width-2,width) == width-2 :
```

```
            bc = bc[:width-2] + ' \n' + bc[width-1:]
        if eachline.rstrip(' \n').rfind(' ',width-2,width) == width-1 :
            bc = bc[:width-1] + ' \n' + bc[width:]
        if eachline.rstrip(' \n').rfind(' ',width-2,width+1) == width :
            bc = bc[:width] + ' \n' + bc[width+1:]
        tmplines = bc.split('\n')
        tmplen = len(tmplines)
        if tmplen == 1 :
            foutput.writelines(bc)
            foutput.writelines('\n')
        else :
            flag_add = 0
            i = 0
            while i < tmplen-1 :
                if(flag_add == 1) :
                    tmplines[i] = indent(tmplines[i], '  ')
                foutput.writelines(tmplines[i])
                foutput.writelines('\\')
                flag_add = 1
                foutput.writelines('\n')
                i += 1
                if(flag_add == 1) :
                    tmplines[i] = indent(tmplines[i], '  ')
                foutput.writelines(tmplines[tmplen-1])
                foutput.writelines('\n')
    foutput.close


def auto_unwrap_dedent(input_file, dst_file) :
    finput=open(input_file, "r")
    alllines=finput.readlines()
    finput.close()
    foutput = 0
    flag_del = 0
    flag_space = 0
    output_file = dst_file
    foutput = open(output_file, 'a')
    for eachline in alllines:
        print(eachline)
        if(flag_del == 1) :
            eachline = eachline[2:]
        if eachline.endswith('\\\n') :
            flag_del = 1
            eachline = eachline.rstrip('\\\n')
            if eachline == '':
                flag_del = 0
        else :
            flag_del = 0
```

```
        if eachline == '\n' :
            continue
        foutput.writelines(eachline)

if __name__ == "__main__":
    auto_wrap("in-1.txt","out-1.txt")
    auto_unwrap("out-1.txt", "out-2.txt")
    auto_wrap_indent("in-1.txt","out-1.txt",73)
    auto_unwrap_dedent("out-1.txt", "out-2.txt")
```

Authors' Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Adrian Farrel
Juniper Networks

Email: afarrel@juniper.net

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com