NETMOD Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: April 19, 2019

K. Watsen
Juniper Networks
Q. Wu
Huawei Technologies
A. Farrel
Juniper Networks
B. Claise
Cisco Systems, Inc.

October 16, 2018

Handling Long Lines in Artwork in Internet-Drafts and RFCs
draft-kwatsen-netmod-artwork-folding-08

Abstract

This document introduces a simple and yet time-proven strategy for handling long lines in artwork in drafts using a backslash ('\') character where line-folding has occurred. The strategy works on any text based artwork, but is primarily intended for sample text and formatted examples and code, rather than for graphical artwork. The approach produces consistent results regardless of the content and uses a per-artwork header. The strategy is both self-documenting and enables automated reconstitution of the original artwork.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Requirements Language 3
3. Goals 3
  3.1. Automated Folding of Long Lines in Artwork 3
  3.2. Automated Reconstitution of Original Artwork 4
4. Limitations 4
  4.1. Not Recommended for Graphical Artwork 4
  4.2. Doesn't Work as Well as Format-Specific Options 4
5. Folded Structure 5
  5.1. Header 5
  5.2. Body 5
6. Algorithm 6
  6.1. Automated Folding 6
    6.1.1. Manual Folding 7
  6.2. Automated Unfolding 7
7. Considerations for xml2rfc v3 8
8. Examples 8
  8.1. Simple Example Showing Boundary Conditions 8
  8.2. Example Showing Multiple Wraps of a Single Line 9
  8.3. Example With Native Backslash 9
  8.4. Example With Native Whitespace 9
  8.5. Example of Manual Wrapping 9
9. Security Considerations 10
10. IANA Considerations 10
11. References 10
  11.1. Normative References 10
  11.2. Informative References 10
Appendix A. POSIX Shell Script 12
Acknowledgements 16
Authors' Addresses 17

1. Introduction

[RFC7994]sets out the requirements for plain-text RFCs and states that each line of an RFC (and hence of an Internet-Draft) must be

limited to 72 characters followed by the character sequence that denotes an end-of-line (EOL).

Internet-Drafts and RFCs often include example text or code fragments. In order to render the formatting of such text it is usually presented as a figure using the "<artwork>" element in the source XML. Many times the example text or code exceeds the 72 character line-length limit and the "xml2rfc" utility does not attempt to wrap the content of artwork, simply issuing a warning whenever artwork lines exceed 69 characters. According to the RFC Editor, there is currently no convention in place for how to handle long lines, other than advising authors to clearly indicate what manipulation has occurred.

This document introduces a simple and yet time-proven strategy for handling long lines using a backslash ('\') character where line-folding has occurred. The strategy works on any text based artwork, but is primarily intended for sample text and formatted examples and code, rather than for graphical artwork. The approach produces consistent results regardless of the content and uses a per-artwork header. The strategy is both self-documenting and enables automated reconstitution of the original artwork.

Note that text files are represent as lines having their first character in column 1, and a line length of N where the last character is in the Nth column and is immediately followed by an end of line character sequence.

2. Requirements Language

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.

3. Goals

3.1. Automated Folding of Long Lines in Artwork

Automated folding of long lines is needed in order to support draft compilations that entail a) validation of source input files (e.g., XML, JSON, ABNF, ASN.1) and/or b) dynamic generation of output, using a tool that doesn't observe line lengths, that is stitched into the final document to be submitted.

Generally, in order for tooling to be able to process input files, the files must be in their original/natural state, which may include

having some long lines. Thus, these source files need to be modified before inclusion in the document in order to satisfy the line length limits. This modification SHOULD be automated to reduce effort and errors resulting from manual effort.

Similarly, dynamically generated output (e.g., tree diagrams) must also be modified, if necessary, in order for the resulting document to satisfy the line length limits. When needed, this effort again SHOULD be automated to reduce effort and errors resulting from manual effort.

3.2. Automated Reconstitution of Original Artwork

Automated reconstitution of the original artwork is needed to support validation of artwork extracted from documents. YANG [RFC7950] modules are already extracted from Internet-Drafts and validated as part of the draft-submission process. Additionally, there has been some discussion regarding needing to do the same for example YANG fragments contained within Internet-Drafts ([yang-doctors-thread]). Thus, it SHOULD be possible to mechanically reconstitute artwork in order to satisfy the tooling input parsers.

4. Limitations

4.1. Not Recommended for Graphical Artwork

While the solution presented in this document will work on any kind of text-based artwork, it is most useful on artwork that represents sourcecode (XML, JSON, etc.) or, more generally, on artwork that has not been laid out in two dimensions (e.g., diagrams).

Fundamentally, the issue is whether the artwork remains readable once folded. Artwork that is unpredictable is especially susceptible to looking bad when folded; falling into this category are most UML diagrams.

It is NOT RECOMMENDED to use the solution presented in this document on graphical artwork.

4.2. Doesn't Work as Well as Format-Specific Options

The solution presented in this document works generically for all artwork, as it only views artwork as plain text. However, various formats sometimes have built-in mechanisms that can be used to prevent long lines.

For instance, both the `pyang` and `yanglint` utilities have the command line option "--tree-line-length" that can be used to indicate

a desired maximum line length for when generating tree diagrams [RFC8340].

In another example, some source formats (e.g., YANG [RFC7950]) allow any quoted string to be broken up into substrings separated by a concatenation character (e.g., '+'), any of which can be on a different line.

In yet another example, some languages allow factoring chunks of code into call outs, such as functions. Using such call outs is especially helpful when in some deeply-nested code, as they typically reset the indentation back to the first column.

As such, it is RECOMMENDED that authors do as much as possible within the selected format to avoid long lines.

5. Folded Structure

Artwork that has been folded as specified by this document MUST contain the following structure.

5.1. Header

The header is two lines long.

The first line is the following 46-character string that MAY be surrounded by any number of printable characters. This first line cannot itself be folded.

```
NOTE: '\\' line wrapping per BCP XX (RFC XXXX)
```

[Note to RFC Editor: Please replace XX and XXXX with the numbers assigned to this document and delete this note. Please make this change in multiple places in this document.]

The second line is a blank line. This line provides visual separation for readability.

5.2. Body

The character encoding is the same as described in Section 2 of [RFC7994], except that, per [RFC7991], tab characters are prohibited.

Lines that have a backslash ('\') occurring as the last character in a line immediately followed by the end of line character sequence, when the subsequent line starts with a backslash ('\') as the first non-space (' ') character, are considered "folded".

Really long lines may be folded multiple times.

6. Algorithm

6.1. Automated Folding

Determine the desired maximum line length from input. If no value is explicitly specified, the value "69" SHOULD be used.

Ensure that the desired maximum line length is not less than the minimum header, which is 46 characters. If the desired maximum line length is less than this minimum, exit (this artwork can not be folded).

Scan the artwork to see if any line exceeds the desired maximum. If no line exceeds the desired maximum, exit (this artwork does not need to be folded).

Scan the artwork for horizontal tab characters. If any horizontal tab characters appear, either resolve them to space characters or exit, forcing the input provider to convert them to space characters themselves first.

Scan the artwork to ensure no existing lines already end with a backslash ('\') character when the subsequent line starts with a backslash ('\') character as the first non-space (' ') character, as this would lead to an ambiguous result. If such a line is found, exit (this artwork cannot be folded).

For each line in the artwork, from top-to-bottom, if the line exceeds the desired maximum, then fold the line at the desired maximum column by 1) inserting the character backslash ('\') character at the maximum column, 2) inserting the end of line character sequence, inserting any number of space (' ') characters, and 4) inserting a further backslash ('\') character.

The result of this previous operation is that the next line starts with an arbitrary number of space (' ') characters, followed by a backslash ('\') character, immediately followed by the character that was previously in the maximum column.

Continue in this manner until reaching the end of the artwork. Note that this algorithm naturally addresses the case where the remainder of a folded line is still longer than the desired maximum, and hence needs to be folded again, ad infinitum.

6.1.1. Manual Folding

Authors may choose to fold text examples and source code by hand to produce a document that is more pleasant for a human reader but which can still be automatically unfolded (as described in Section 6.2) to produce single lines that are longer than the maximum document line length.

For example, an author may choose to make the fold at convenient gaps between words such that the backslash is placed in a lower column number than the artwork's maximum column value.

Additionally, an author may choose to indent the start of a continuation line by inserting space characters before the line continuation marker backslash character.

Manual folding may also help handle the cases that cannot be automatically folded as described in Section 6.

6.2. Automated Unfolding

All unfolding is assumed to be automated although a reader will mentally perform the act of unfolding the text to understand the true nature of the artwork or source code.

Scan the beginning of the artwork for the header described in Section 5.1. If the header is not present, starting on the first line of the artwork, exit (this artwork does not need to be unfolded).

Remove the 2-line header from the artwork.

For each line in the artwork, from top-to-bottom, if the line has a backslash ('\') character immediately followed by the end of line character sequence, and if the next line has a backslash ('\') character as the first non-space (' ') character, then the lines can be unfolded. Remove the first backslash ('\') character, the end of line character sequence, any leading space (' ') characters, and the second backslash ('\') character, which will bring up the next line. Then continue to scan each line in the artwork starting with the current line (in case it was multiply folded).

Continue in this manner until reaching the end of the artwork.

7. Considerations for xml2rfc v3

[RFC7991] introduces the vocabulary for version 3 of the xml2rfc tool. This includes a new element, "<sourcecode>" used to present sourcecode examples and fragments and to distinguish them from general artwork and in particular figures and graphics.

The folding and unfolding described in this document is applicable to the "<artwork>" element in both v2 and v3 of xml2rfc, and is equally applicable to the "<sourcecode>" element in xml2rfc v3.

8. Examples

The following self-documenting examples illustrate a folded document.

The source artwork cannot be presented here, as it would again need to be folded. Alas, only the result can be provided.

The examples in Sections 8.1 through 8.4 were automatically folded on column 69, the default value. Section 8.5 shows an example of manual folding.

8.1. Simple Example Showing Boundary Conditions

This example illustrates a boundary condition test using numbers for counting purposes. The input contains 5 lines, each line one character longer than the previous.

Any printable character (including ' ' and '\') can be used as a substitute for any number, except for on the 4th row, the trailing '9' is not allowed to be a '\' character if the first non-space character of the next line is a '\' character, as that would lead to an ambiguous result.

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

1234567890123456789012345678901234567890123456789012345678901234566
12345678901234567890123456789012345678901234567890123456789012345677
123456789012345678901234567890123456789012345678901234567890123456788
1234567890123456789012345678901234567890123456789012345678901234567899
12345678901234567890123456789012345678901234567890123456789012345678\
\90
12345678901234567890123456789012345678901234567890123456789012345678\
\901
12345678901234567890123456789012345678901234567890123456789012345678\
\9012
```

8.2. Example Showing Multiple Wraps of a Single Line

This example illustrates one very long line (280 characters).

Any printable character (including ' ' and '\') can be used as a substitute for any number.

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

1234567890123456789012345678901234567890123456789012345678901234567\
\9012345678901234567890123456789012345678901234567890123456789012345\
\6789012345678901234567890123456789012345678901234567890123456789012\
\3456789012345678901234567890123456789012345678901234567890123456789\
\01234567890
```

8.3. Example With Native Backslash

This example has a '\' character in the wrapping column. The native text includes the sequence "fish\fowl" with the '\' character occurring on the 69th column.

```
string1="The quick brown dog jumps over the lazy dog which is a fish\
\\fowl as appropriate"
```

8.4. Example With Native Whitespace

This example has whitespace spanning the wrapping column. The native input contains 15 space (' ') characters between "like" and "white".

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

Sometimes our strings include multiple spaces such as "We like      \
\        white space."
```

8.5. Example of Manual Wrapping

This example was manually wrapped to cause the folding to occur after each term, putting each term on its own line. Indentation is used to additionally improve readability. Also note that the mandatory header is surrounded by different printable characters than shown in the other examples.

```
   [NOTE: '\\' line wrapping per BCP XX (RFC XXXX)]

   <request>::= <RP> \
                \<END-POINTS> \
                \[<LSPA>] \
                \[<BANDWIDTH>] \
                \[<metric-list>] \
                \[<RRO>[<BANDWIDTH>]] \
                \[<IRO>] \
                \[<LOAD-BALANCING>]
```

The manual folding produces a more readable result than the following equivalent folding that contains no indentation.

```
   ========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

   <request>::= <RP> <END-POINTS> [<LSPA>] [<BANDWIDTH>] [<metric-list>\
   \] [<RRO>[<BANDWIDTH>]] [<IRO>] [<LOAD-BALANCING>]
```

9. Security Considerations

This BCP has no Security Considerations.

10. IANA Considerations

This BCP has no IANA Considerations.

11. References

11.1. Normative References

[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>.

[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>.

11.2. Informative References

[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>.

[RFC7991] Hoffman, P., "The "xml2rfc" Version 3 Vocabulary", RFC 7991, DOI 10.17487/RFC7991, December 2016, <https://www.rfc-editor.org/info/rfc7991>.

[RFC7994] Flanagan, H., "Requirements for Plain-Text RFCs", RFC 7994, DOI 10.17487/RFC7994, December 2016, <https://www.rfc-editor.org/info/rfc7994>.

[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>.

[yang-doctors-thread]
"[yang-doctors] automating yang doctor reviews", <https://mailarchive.ietf.org/arch/msg/yang-doctors/DCfBqgfZPAD7afzeDFlQ1Xm2X3g>.

Appendix A. POSIX Shell Script

This non-normative appendix section includes a shell script that can both fold and unfold artwork.

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

#!/bin/bash

print_usage() {
  echo
  echo "Folds the text file, only if needed, at the specified"
  echo "column, according to BCP XX."
  echo
  echo "Usage: $0 [-c <col>] [-r] -i <infile> -o <outfile>"
  echo
  echo "  -c: column to fold on (default: 69)"
  echo "  -r: reverses the operation"
  echo "  -i: the input filename"
  echo "  -o: the output filename"
  echo "  -d: show debug messages"
  echo "  -h: show this message"
  echo
  echo "Exit status code: zero on success, non-zero otherwise."
  echo
}


# global vars, do not edit
debug=0
reversed=0
infile=""
outfile=""
maxcol=69  # default, may be overridden by param
hdr_txt="NOTE: '\\\\' line wrapping per BCP XX (RFC XXXX)"
equal_chars="=============================================="
space_chars="                                              "

fold_it() {
  # since upcomming tests are >= (not >)
  testcol=`expr "$maxcol" + 1`

  # check if file needs folding
  grep ".\{$testcol\}" $infile >> /dev/null 2>&1
  if [ $? -ne 0 ]; then
    if [[ $debug -eq 1 ]]; then
      echo "nothing to do"
    fi
```

```
       cp $infile $outfile
       return -1
     fi

     foldcol=`expr "$maxcol" - 1` # for the inserted '\' char

     # ensure input file doesn't contain a TAB
     grep "\t" $infile >> /dev/null 2>&1
     if [ $? -eq 0 ]; then
       echo
       echo "Error: infile contains a TAB character, which is not allow\
   \ed."
       echo
       return 1
     fi

     # ensure input file doesn't contain the fold-sequence already
     pcregrep -M  "\\\\\n[\ ]*\\\\" $infile >> /dev/null 2>&1
     if [ $? -eq 0 ]; then
       echo
       echo "Error: infile has a line ending with a '\' character follo\
   \wed"
       echo "       by '\' as the first non-space character on the next\
   \ line."
       echo "       This file cannot be folded."
       echo
       return 1
     fi

     # center header text
     length=`expr ${#hdr_txt} + 2`
     left_sp=`expr \( "$maxcol" - "$length" \) / 2`
     right_sp=`expr "$maxcol" - "$length" - "$left_sp"`
     header=`printf "%.*s %s %.*s" "$left_sp" "$equal_chars" "$hdr_txt"\
   \ "$right_sp" "$equal_chars"`

     # fold using recursive passes ('g' didn't work)
     if [ -z "$1" ]; then
       # init recursive env
       cp $infile /tmp/wip
     fi
     gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\n\\\\/" < /tmp/wip\
   \ >> /tmp/wip2
     diff /tmp/wip /tmp/wip2 > /dev/null 2>&1
     if [ $? -eq 1 ]; then
       mv /tmp/wip2 /tmp/wip
       fold_it "recursing"
     else
```

```
      echo "$header" > $outfile
      echo "" >> $outfile
      cat /tmp/wip2 >> $outfile
      rm /tmp/wip*
    fi

    ## following two lines represent a non-functional variant to the r\
  \ecursive
    ## logic presented in the block above.  It used to work before the\
  \ '\'
    ## on the next line was added to the format (i.e., the trailing '\\
  \\\\'
    ## in the substitution below), but now there is an off-by-one erro\
  \r.
    ## Leaving here in case anyone can fix it.
    #echo "$header" > $outfile
    #echo "" >> $outfile
    #gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\n\\\\/g" < $infil\
  \e >> $outfile

    return 0
  }


  unfold_it() {
    # check if file needs unfolding
    line=`head -n 1 $infile | fgrep "$hdr_txt"`
    if [ $? -ne 0 ]; then
      if [[ $debug -eq 1 ]]; then
        echo "nothing to do"
      fi
      cp $infile $outfile
      return -1
    fi

    # output all but the first two lines (the header) to wip (work in \
  \progress) file
    awk "NR>2" $infile > /tmp/wip

    # unfold wip file
    gsed ":x; /.*\\\\\$/N; s/\\\\\n[ ]*\\\\//; tx; s/\t//g" /tmp/wip >\
  \ $outfile

    # clean up and return
    rm /tmp/wip
    return 0
  }
```

```
   process_input() {
     while [ "$1" != "" ]; do
       if [ "$1" == "-h" -o "$1" == "--help" ]; then
         print_usage
         exit 1
       fi
       if [ "$1" == "-d" ]; then
         debug=1
       fi
       if [ "$1" == "-c" ]; then
         maxcol="$2"
         shift
       fi
       if [ "$1" == "-r" ]; then
         reversed=1
       fi
       if [ "$1" == "-i" ]; then
         infile="$2"
         shift
       fi
       if [ "$1" == "-o" ]; then
         outfile="$2"
         shift
       fi
       shift
     done

     if [ -z "$infile" ]; then
       echo
       echo "Error: infile parameter missing (use -h for help)"
       echo
       exit 1
     fi

     if [ -z "$outfile" ]; then
       echo
       echo "Error: outfile parameter missing (use -h for help)"
       echo
       exit 1
     fi

     if [ ! -f "$infile" ]; then
       echo
       echo "Error: specified file \"$infile\" is does not exist."
       echo
       exit 1
     fi
```

```
     min_supported=`expr ${#hdr_txt} + 8`
     if [ $maxcol -lt $min_supported ]; then
       echo
       echo "Error: the folding column cannot be less than $min_support\
   \ed"
       echo
       exit 1
     fi

     max_supported=`expr ${#equal_chars} + 1 + ${#hdr_txt} + 1 + ${#equ\
   \al_chars}`
     if [ $maxcol -gt $max_supported ]; then
       echo
       echo "Error: the folding column cannot be more than $max_support\
   \ed"
       echo
       exit 1
     fi

   }


   main() {
     if [ "$#" == "0" ]; then
        print_usage
        exit 1
     fi

     process_input $@

     if [[ $reversed -eq 0 ]]; then
       fold_it
       code=$?
     else
       unfold_it
       code=$?
     fi
     exit $code
   }

   main "$@"
```

Acknowledgements

The authors thank the following folks for their various contributions (sorted by first name): Jonathan Hansford, Joel Jaeggli, Lou Berger, Martin Bjorklund, Italo Busi, and Rob Wilton.

The authors additionally thank the RFC Editor, for confirming that there is no set convention today for handling long lines in artwork.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Qin Wu
Huawei Technologies

EMail: bill.wu@huawei.com

Adrian Farrel
Juniper Networks

EMail: afarrel@juniper.net

Benoit Claise
Cisco Systems, Inc.

EMail: bclaise@cisco.com