Handling Long Lines in Artwork in Drafts
draft-kwatsen-netmod-artwork-folding-03

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, producing consistent results regardless the
   artwork content. Using a per-artwork header, the strategy is both
   self-documenting and enables automated reconstitution of the original
   artwork.

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 20, 2018.

Table of Contents

1. Introduction

Internet drafts many times contain artwork that exceed the 72
character limit specified by RFC 7994 [RFC7994]. The "xml2rfc"
utility, in an effort to maintain clean formatting, issues a warning
whenever artwork lines exceed 69 characters. According to RFC
Editor, there is currently no convention in place for how to handle
long lines, other than clearly indicating that some 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,
producing consistent results regardless the artwork content. Using a
per-artwork header, the strategy is both self-documenting and enables
automated reconstitution of the original artwork.

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.,
YANG, XML, JSON, ABNF, ASN.1) and/or b) dynamic generation of output
(e.g., tree diagrams) that are stitched into the final draft 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 draft 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 I-D 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 drafts. Already YANG modules
are extracted from drafts and validated as part of the draft-
submission process. Additionally, there has been some discussion
regarding needing to do the same for examples contained within drafts
([yang-doctors-list]). Thus, it SHOULD be possible to mechanically
reconstitute artwork in order to satisfy the tooling input parsers.

4. Limitations

4.1. Doesn't work well on 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 (e.g., YANG, XML, JSON, etc.) or, more generally, on

artwork that has not been laid out in two dimensions (e.g., diagrams).

The issue regards the readability of the folded artwork in the draft. Artwork that is unpredictable is especially susceptible is looking bad when folded; falling into this category are most UML diagrams. Artwork that is somewhat structured (e.g., YANG tree diagrams [RFC8340]) fair better when folded, as the eyes seem to be able to still see the vertical lines, even when they are interrupted.

It is thus 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 mechanisms that can be used to prevent long lines.

For instance, some source formats allow any quoted string to be broken up into substrings separated by a concatenation character ('+'), any of which can by on a different line.

In another example, some languages allow factoring out chucks of code out into "functions" or "groupings". Using such call outs is especially helpful when in some deeply-nested code, as it typically resets 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. Solution

The following two sections provide the folding and unfolding algorithms that MUST be implemented to align with this BCP.

5.1. Header

Any artwork that has been folded as specified by this document MUST contain the header described in this section.

The header is two lines long.

The first line is the following 53-character string that has been padded with roughly equal numbers of equal ('=') characters to reach the artwork's maximum line length. This line is self-describing in three ways: use of '\' character, identification of BCP/RFC, and

identification of what the maximum line length is for the artwork.
Following is the mimimal header string (53-characters):

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

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

5.2. Folding

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).

Ensure that the desired maximum is not less than then minumum header,
which is 53 characters. If the desired maximum is less than this
minimum, exit (this artwork can not be folded).

Scan the artwork to ensure no existing lines already end with a '\'
character on the desired maximum column, as this would be 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 inserting the string "\\n" (backlash followed by line return) at
the column before the maximum column. Note that the column before
needs to be used in order to enable the '\' character to be placed on
the desired maximum column. The result of this operation is that the
character that was on the maximum colomn is now the first character
of the next line.

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.

5.3. Unfolding

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).

Caluculate the folding-column used from the length of the provided
header.

Remove the 2-line header from the artwork.

For each line in the artwork, from top-to-bottom, if the line has a
'\' on the folding-column followed by a '\n' character, then remove
both the '\' and '\n' characters, which will bring up the next line,
and then scan the remainder of the line to see if it again has a '\'
after folding-column characters followed by a '\n' character, and so
on.

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

5.4. Example

The following self-documenting example illustrates the result of the
folding algorithm running over a specific artwork input.

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

Some things to note about the following example:

o This artwork is exactly 69 characters wide, the widest possible
   before `xml2rfc` starts to issue warnings.

o The line having the 'x' character on the 69th column would've been
   illegal input had the '\' been used.

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

# boundary condition tests using numbers for counting purposes
123456789012345678901234567890123456789012345678901234567890123456
123456789012345678901234567890123456789012345678901234567890123456
1234567890123456789012345678901234567890123456789012345678901234567
12345678901234567890123456789012345678901234567890123456789012345678
123456789012345678901234567890123456789012345678901234567890123456789
12345678901234567890123456789012345678901234567890123456789012345678\
90
12345678901234567890123456789012345678901234567890123456789012345678\
901

# same as above, but every character converted to a backslash
# ...and the offending "\\n" on column 69 is removed...
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\x
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\

# one very long line (280 characters)
12345678901234567890123456789012345678901234567890123456789012345678\
90123456789012345678901234567890123456789012345678901234567890123456\
78901234567890123456789012345678901234567890123456789012345678901234\
56789012345678901234567890123456789012345678901234567890123456789012\
34567890

# same as above, but every character converted to a backslash
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\
```

6. Security Considerations

This BCP has no Security Considerations.

7. IANA Considerations

This BCP has no IANA Considerations.

8. References

8.1. 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>.

8.2. Informative References

[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-list]
           "[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 based on the solution presented in this
   document.

   As a testament for the simplicity of this solution, note that at the
   core of the script are the following two one-liners:

   For folding:
     gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\n/g"

   For unfolding:
     gsed ":x; /[^\t]\\{$foldcol\\}\\\\\$/N; s/\\\\\n/\t/; tx; s/\t//g"

   Disclaimer: this script has the limitation of disallowing the input
   file from containing any TAB ('\t') characters.

   =====START SCRIPT=====

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

```
   #!/bin/bash
   #
   # the only reason why /bin/sh isn't being used
   # is because "echo -n" is broken on the Mac.

   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="======================================="

  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 0
    fi

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

    # ensure file doesn't have any '\' char on $maxcol already
    #  - as this would lead to false positives...
    grep "^.\{$foldcol\}\\\\$" $infile >> /dev/null 2>&1
    if [ $? -eq 0 ]; then
      echo
      echo "Error: infile has a '\\\' on colomn $maxcol already."
      echo
      exit 1
    fi

    # calculate '=' character-filled header
    length=${#hdr_txt}
    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"`

    # generate outfile and return
    echo -ne "$header\n\n" > $outfile
    gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\\n/g" < $infile >> \
$outfile
    return 0
  }
```

```
  unfold_it() {
    # check if it looks like a BCP XX header
    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 0
    fi

    # determine what maxcol value was used
    maxcol=${#line}

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

    # unfold wip file
    foldcol=`expr "$maxcol" - 1` # for the inserted '\' char
    gsed ":x; /[^\t]\\{$foldcol\\}\\\\\\$/N; s/\\\\\\n/\t/; 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=${#hdr_txt}
    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} + ${#hdr_txt} + ${#equal_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 "$@"

    =====END SCRIPT=====
```

Acknowledgements

   The authors thank the RFC Editor for confirming that there are 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