

Handling Long Lines in Artwork in Drafts
draft-kwatsen-netmod-artwork-folding-00

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 notice, 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 November 30, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4](#).e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Requirements Language	2
3.	Goals	3
3.1.	Automated folding of long lines in artwork	3
3.2.	Automated reconstitution of original artwork	3
4.	Limitations	3
4.1.	Doesn't work well on graphical artwork	3
4.2.	Doesn't work as well as format-specific options	4
5.	Solution	4
5.1.	Folding	4
5.2.	Unfolding	5
6.	Security Considerations	5
7.	IANA Considerations	5
8.	References	5
8.1.	Normative References	5
8.2.	Informative References	5
Appendix A.	POSIX Shell Script	7
	Acknowledgements	10
	Author's Address	10

[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 notice, 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 to 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 be on a different line.

In another example, some languages allow factoring out chunks 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. 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).

Otherwise, as it is determined the artwork needs to be folded, prepend the 3-line header (RFC Ed. please replace XX below with the assigned value for this BCP):

```
\n[Note: '\' line wrapping added per BCP XX]\n\n
```

For each line in the artwork, from top-to-bottom, if the line exceeds the desired maximum, then fold the line at the desired column by inserting the string "\\n" at the column before the maximum column. Note that the spacer is needed to give room for the '\' character.

Continue in this manner until reaching the end of the artwork. Note that the algorithm naturally addresses the case where the remainder of a folded line is still longer than the desired maximum.

5.2. Unfolding

Scan the artwork for the above-mentioned header. If the header is not present in the artwork, exit (this artwork does not need to be unfolded).

Remove the 3-line header from the artwork.

For each line in the artwork, from top-to-bottom, if the line ends in "\\n", then remove the "\\n" and then scan the remainder of the line to see if it again ends in "\\n", and so on.

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

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/
DCfBqgfZPAD7afzeDF1Q1Xm2X3g](https://mailarchive.ietf.org/arch/msg/yang-doctors/DCfBqgfZPAD7afzeDF1Q1Xm2X3g)>.

[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 "s/\(.\\{$foldcol\\}\)/\1\\n/"
For unfolding: gsed ':x; /\n$/ { N; s/\n//; tx }'
```

====START SCRIPT====

```
#!/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 "Wraps file representing IETF artwork at specified column"
    echo "according to BCP XX. Note, this routine does nothing if"
    echo "the infile has no lines longer than specified."
    echo
    echo "Usage: $0 [-r] [-c <col>] -i <infile> -o <outfile>"
    echo
    echo "  -c: column to wrap on (default: 69)"
    echo "  -r: reverses the operation"
    echo "  -i: the input filename"
    echo "  -o: the output filename"
    echo "  -h: show this message"
    echo
    echo "Exit status code: zero on success, non-zero otherwise."
    echo
}

# global vars, do not edit
reversed=0
infile=""
outfile=""
maxcol=69 # default, may be overridden by param
header="\n[Note: '\n' line wrapping added per BCP XX]\n\n"

fold_it() {
    # check if file needs folding
```



```

grep "\.${maxcol}" $infile >> /dev/null 2>&1
if [ $? -ne 0 ]; then
    # nothing to do
    cp $infile $outfile
    return 1
fi

echo -ne "$header" > $outfile
foldcol=`expr "$maxcol" - 1` # spacer for the inserted '\' char
gsed "s/\(.\${foldcol}\)/\1\\\n/" < $infile >> $outfile
return 0
}

```

```

unfold_it() {
    # count lines in header
    numlines=`echo -ne "$header" | wc -l`

    # check if file needs unfolding
    echo -ne "$header" > /tmp/header
    head -n $numlines $infile > /tmp/header2
    diff -q /tmp/header /tmp/header2 >> /dev/null
    code=$?
    rm /tmp/header /tmp/header2
    if [ $code -ne 0 ]; then
        # nothing to do
        cp $infile $outfile
        return 1
    fi

    awk "NR>$numlines" $infile > /tmp/wip
    gsed ':x; /\n$/ { N; s/\n//; tx }' /tmp/wip > $outfile
    rm /tmp/wip
    return 0
}

```

```

process_input() {
    while [ "$1" != "" ]; do
        if [ "$1" == "-h" -o "$1" == "--help" ]; then
            print_usage
            exit 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 "error: infile parameter missing."
    exit 1
fi

if [ -z "$outfile" ]; then
    echo "error: outfile parameter missing."
    exit 1
fi

if [ ! -f "$infile" ]; then
    echo "error: infile \"$infile\" does not exist."
    exit 1
fi

if [ -f "$outfile" ]; then
    echo "warning: outfile \"$outfile\" already exists."
fi
}

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

    process_input $@

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



```
}
```

```
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.

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

