

NETMOD Working Group  
Internet-Draft  
Intended status: Best Current Practice  
Expires: December 28, 2018

K. Watsen  
Juniper Networks  
Q. Wu  
Huawei Technologies  
A. Farrel  
Juniper Networks  
B. Claise  
Cisco Systems, Inc.  
June 26, 2018

**Handling Long Lines in Artwork in Drafts**  
**draft-kwatsen-netmod-artwork-folding-06**

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 28, 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

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">2.</a>	<a href="#">Requirements Language</a>	<a href="#">3</a>
<a href="#">3.</a>	<a href="#">Goals</a>	<a href="#">3</a>
<a href="#">3.1.</a>	<a href="#">Automated folding of long lines in artwork</a>	<a href="#">3</a>
<a href="#">3.2.</a>	<a href="#">Automated reconstitution of original artwork</a>	<a href="#">3</a>
<a href="#">4.</a>	<a href="#">Limitations</a>	<a href="#">4</a>
<a href="#">4.1.</a>	<a href="#">Doesn't work well on graphical artwork</a>	<a href="#">4</a>
<a href="#">4.2.</a>	<a href="#">Doesn't work as well as format-specific options</a>	<a href="#">4</a>
<a href="#">5.</a>	<a href="#">Folded Structure</a>	<a href="#">4</a>
<a href="#">5.1.</a>	<a href="#">Header</a>	<a href="#">4</a>
<a href="#">5.2.</a>	<a href="#">Body</a>	<a href="#">5</a>
<a href="#">6.</a>	<a href="#">Algorithm</a>	<a href="#">5</a>
<a href="#">6.1.</a>	<a href="#">Folding</a>	<a href="#">5</a>
<a href="#">6.2.</a>	<a href="#">Unfolding</a>	<a href="#">6</a>
<a href="#">7.</a>	<a href="#">Example</a>	<a href="#">6</a>
<a href="#">8.</a>	<a href="#">Security Considerations</a>	<a href="#">7</a>
<a href="#">9.</a>	<a href="#">IANA Considerations</a>	<a href="#">7</a>
<a href="#">10.</a>	<a href="#">References</a>	<a href="#">7</a>
<a href="#">10.1.</a>	<a href="#">Normative References</a>	<a href="#">7</a>
<a href="#">10.2.</a>	<a href="#">Informative References</a>	<a href="#">8</a>
<a href="#">Appendix A.</a>	<a href="#">POSIX Shell Script</a>	<a href="#">9</a>
	<a href="#">Acknowledgements</a>	<a href="#">13</a>
	<a href="#">Authors' Addresses</a>	<a href="#">13</a>

## [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 ([\[yang-doctors-thread\]](#)). 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 (YANG, 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. Artwork that is somewhat structured (e.g., YANG tree diagrams [[RFC8340](#)]) fairs better when folded, as the eyes seem to be able to still see the vertical lines, even when they are interrupted.

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, 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 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 an N-character string:



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

where N is the column on which folding occurred (the minimal value is 53, the length of the string above) padded with roughly equivalent number of equal '=' characters on both sides of the string to reach the artwork's maximum line length.

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

## 5.2. Body

The character encoding is the same as described in [Section 2 of \[RFC7994\]](#), except that, per [\[RFC7991\]](#), tab ('\t') characters are prohibited.

The backslash ('\') character may appear anywhere in the artwork, including at the end of any line.

Lines that have a backslash occurring on the artwork's maximum column value (N) followed by the '\n' character are considered "folded".

A really long line may be folded multiple times.

Folded lines are continued on the next line on column 0.

## 6. Algorithm

### 6.1. 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 53 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 to ensure no existing lines already end with a '\' character on the desired maximum column, as this would lead to an ambiguous result. If such a line is found, exit (this artwork cannot be folded).





Scan the artwork to ensure the horizontal tab character '\t' does not appear. If any horizontal tab character appears, 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" (backslash 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 column 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.

## **6.2. 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).

Calculate 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.

## **7. Example**

The following self-documenting example illustrates 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.

This artwork was folded on column 69, the default value.



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

```
# Boundary condition tests using numbers for counting purposes.
#
# 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, as
# that leads to an ambiguous result.
123456789012345678901234567890123456789012345678901234567890123456
1234567890123456789012345678901234567890123456789012345678901234567
12345678901234567890123456789012345678901234567890123456789012345678
123456789012345678901234567890123456789012345678901234567890123456789
12345678901234567890123456789012345678901234567890123456789012345678\
90
12345678901234567890123456789012345678901234567890123456789012345678\
901

# One very long line (280 characters)
#
# Any printable character (including ' ' and '\\') can be used
# as a substitute for any number, except the 69th character
# is not allowed to be a '\\' character, as that leads to an
# ambiguous result.
12345678901234567890123456789012345678901234567890123456789012345678\
90123456789012345678901234567890123456789012345678901234567890123456\
78901234567890123456789012345678901234567890123456789012345678901234\
56789012345678901234567890123456789012345678901234567890123456789012\
34567890
```

## **8. Security Considerations**

This BCP has no Security Considerations.

## **9. IANA Considerations**

This BCP has no IANA Considerations.

## **10. References**

### **10.1. 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>>.



- [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>>.
- [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>>.

## **10.2. Informative References**

- [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 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 HTAB ('\t') characters. It does not, for instance, make an attempt to convert an HTAB character to a fixed number of SPACE (' ') 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 column $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/\t/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
    done
```



```
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 following folks for their various contributions (sorted by first name): Martin Bjorklund, Jonathan Hansford, 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