

[Mailinglist](#) [Github](#)[HOME](#) [PROJECT ▼](#) [HELP ▼](#) [TOOLS ▼](#)[rsyslog](#)[PROFESSIONAL SERVICES ▼](#) [WINDOWS AGENT ▼](#) [Posts](#)

The rocket-fast system for log processing



RSyslog Documentation

[Home](#) > RSyslog Documentation

rsyslog 8.2104.0 documentation

The Property Replacer

The **property replacer** is a core component in **rsyslogd's** [string template system](#). A syslog message has a number of well-defined properties. Each of these properties can be accessed **and** manipulated by the property replacer. With it, it is easy to use only part of a property value or manipulate the value, e.g. by converting all characters to lower case.

Accessing Properties

Syslog message properties are used inside templates. They are accessed by putting them between percent signs. Properties can be modified by the property replacer. The full syntax is as follows:

```
%property:fromChar:toChar:options%
```

Available Properties

The property replacer can use all [rsyslog properties](#).

Character Positions

FromChar and **toChar** are used to build substrings. They specify the offset within the string that should be copied. Offset counting starts at 1, so if you need to obtain the first 2 characters of the message text, you can use syntax: "%msg:1:2%". If you do not wish to specify from and to, but you want to specify options, you still include the colons. For example, if you would like to convert the full message text to lower case, use "%msg:::lowercase%". If you would like to extract from a position until the end of the string, you can place a dollar-

sign (“\$”) in toChar (e.g. %msg:10:\$%, which will extract from position 10 to the end of the string).

There is also support for **regular expressions**. To use them, you need to place a “R” into FromChar. This tells rsyslog that a regular expression instead of position-based extraction is desired. The actual regular expression must then be provided in toChar. The regular expression **must** be followed by the string “-end”. It denotes the end of the regular expression and will not become part of it. If you are using regular expressions, the property replacer will return the part of the property text that matches the regular expression. An example for a property replacer sequence with a regular expression is: “%msg:R:.*Sev:. \(.*) \[.*-end%”

It is possible to specify some parameters after the “R”. These are comma-separated. They are:

R,<regex-type>,<submatch>,<nomatch>,<match-number>

regex-type is either “BRE” for Posix basic regular expressions or “ERE” for extended ones. The string must be given in upper case. The default is “BRE” to be consistent with earlier versions of rsyslog that did not support ERE. The submatch identifies the submatch to be used with the result. A single digit is supported. Match 0 is the full match, while 1 to 9 are the actual submatches. The match-number identifies which match to use, if the expression occurs more than once inside the string. Please note that the first match is number 0, the second 1 and so on. Up to 10 matches (up to number 9) are supported. Please note that it would be more natural to have the match-number in front of submatch, but this would break backward-compatibility. So the match-number must be specified after “nomatch”.

nomatch specifies what should be used in case no match is found.

The following is a sample of an ERE expression that takes the first submatch from the message string and replaces the expression with the full field if no match is found:

```
%msg:R,ERE,1,FIELD:for (vlan[0-9]\*):--end%
```

and this takes the first submatch of the second match of said expression:

```
%msg:R,ERE,1,FIELD,1:for (vlan[0-9]\*):--end%
```

Please note: there is also a [rsyslog regular expression checker/generator online tool available](#). With that tool, you can check your regular expressions and also generate a valid property replacer sequence. Usage of this tool is recommended. Depending on the version offered, the tool may not cover all subtleties that can be done with the property replacer. It concentrates on the most often used cases. So it is still useful to hand-craft expressions for demanding environments.

Also, extraction can be done based on so-called “fields”. To do so, place a “F” into FromChar. A field in its current definition is anything that is delimited by a delimiter character. The delimiter by default is TAB (US-ASCII value 9). However, it can be changed to any other US-ASCII character by specifying a comma and the **decimal** US-ASCII value of the delimiter immediately after the “F”. For example, to use comma (“,”) as a delimiter, use this field specifier: “F,44”. If your syslog data is delimited, this is a quicker way to extract than via regular expressions (actually, a *much* quicker way). Field counting starts at 1. Field zero is accepted, but will always lead to a “field not found” error. The same happens if a field number higher than the number of fields in the property is requested. The field number must be placed in the “ToChar” parameter. An example where the 3rd field (delimited by TAB) from the msg property is extracted is as follows: “%msg:F:3%”. The same example with semicolon as delimiter is “%msg:F,59:3%”.

The use of fields does not permit to select substrings, what is rather unfortunate. To solve this issue, starting with [rsyslog 6.3.9](#), fromPos and toPos can be specified for strings as well. However, the syntax is quite ugly, but it was the only



way to integrate this functionality into the already-existing system. To do so, use “,fromPos” and “,toPos” during field extraction. Let’s assume you want to extract the substring from position 5 to 9 in the previous example. Then, the syntax is as follows: “%msg:F,59,5:3,9%”. As you can see, “F,59” means field-mode, with semicolon delimiter and “,5” means starting at position 5. Then “3,9” means field 3 and string extraction to position 9.

Please note that the special characters “F” and “R” are case-sensitive. Only upper case works, lower case will return an error. There are no white spaces permitted inside the sequence (that will lead to error messages and will NOT provide the intended result).

Each occurrence of the field delimiter starts a new field. However, if you add a plus sign (“+”) after the field delimiter, multiple delimiters, one immediately after the others, are treated as separate fields. This can be useful in cases where the syslog message contains such sequences. A frequent case may be with code that is written as follows:

```
int n, m;
...
syslog(LOG_ERR, "%d test %6d", n, m);
```

This will result into things like this in syslog messages: “1 test 2”, “1 test 23”, “1 test 234567”

As you can see, the fields are delimited by space characters, but their exact number is unknown. They can properly be extracted as follows:

```
"%msg:F,32:2%" to "%msg:F,32+:2%".
```

This feature was suggested by Zhuang Yuyao and implemented by him. It is modeled after perl compatible regular expressions.

Property Options

Property options are case-insensitive. Currently, the following options are defined:

uppercase

convert property to uppercase only

lowercase

convert property text to lowercase only


fixed-width

changes behaviour of toChar so that it pads the source string with spaces up to the value of toChar if the source string is shorter. *This feature was introduced in rsyslog 8.13.0*

json

encode the value so that it can be used inside a JSON field. This means that several characters (according to the JSON spec) are being escaped, for example US-ASCII LF is replaced by “\n”. The json option cannot be used together with either jsonf or csv options.

jsonf[:outname]

(available in 6.3.9+) This signifies that the property should be expressed as a JSON field. That means only the property is written, but rather a complete JSON field in the format 

```
"fieldname"="value"
```

where “fieldname” is given in the *outname* property (or the property name if none was assigned) and value is the end result of property replacer operation. Note that value supports all property replacer options, like substrings, case conversion and the like. Values are properly JSON-escaped, however field names are (currently) not, so it is expected that proper field names are configured. The *jsonf* option cannot be used together with either *json* or *csv* options.

For more information you can read [this article from Rainer's blog](#).

csv

formats the resulting field (after all modifications) in CSV format as specified in [RFC 4180](#). Rsyslog will always use double quotes. Note that in order to have full CSV-formatted text, you need to define a proper template. An example is this one: `$template csvline,"%syslogtag:::csv%,%msg:::csv%"` Most importantly, you need to provide the commas between the fields inside the template. *This feature was introduced in rsyslog 4.1.6.*

drop-last-lf

The last LF in the message (if any), is dropped. Especially useful for PIX.

date-utc

convert data to UTC prior to outputting it (available since 8.18.0)

date-mysql

format as mysql date

date-rfc3164

format as RFC 3164 date

date-rfc3164-buggyday

similar to *date-rfc3164*, but emulates a common coding error: RFC 3164 demands that a space is written for single-digit days. With this option, a zero is written instead. This format seems to be used by *syslog-ng* and the *date-rfc3164-buggyday* option can be used in migration scenarios where otherwise lots of scripts would need to be adjusted. It is recommended *not* to use this option when forwarding to remote hosts - they may treat the date as invalid (especially when parsing strictly according to RFC 3164).

This feature was introduced in rsyslog 4.6.2 and v4 versions above and 5.5.3 and all versions above.

date-rfc3339

format as RFC 3339 date

date-unixtimestamp

Format as a unix timestamp (seconds since epoch)

date-year

just the year part (4-digit) of a timestamp

date-month

just the month part (2-digit) of a timestamp

date-day

just the day part (2-digit) of a timestamp

date-hour



just the hour part (2-digit, 24-hour clock) of a timestamp

date-minute

just the minute part (2-digit) of a timestamp

date-second

just the second part (2-digit) of a timestamp

date-subseconds

just the subseconds of a timestamp (always 0 for a low precision timestamp)

date-tzoffshour

just the timezone offset hour part (2-digit) of a timestamp

date-tzoffmin

just the timezone offset minute part (2-digit) of a timestamp. Note that this is usually 0, but there are some time zones that have offsets which are not hourly-granular. If so, this is the minute offset.

date-tzoffsdirection

just the timezone offset direction part of a timestamp. This specifies if the offsets needs to be added (“+”) or subtracted (“-“) to the timestamp in order to get UTC.

date-ordinal

returns the ordinal for the given day, e.g. it is 2 for January, 2nd

date-week

returns the week number

date-wday

just the weekday number of the timestamp. This is a single digit, with 0=Sunday, 1=Monday, ..., 6=Saturday.

date-wdayname

just the abbreviated english name of the weekday (e.g. “Mon”, “Sat”) of the timestamp.

escape-cc

replace control characters (ASCII value 127 and values less then 32) with an escape sequence. The sequence is “#<charval>” where charval is the 3-digit decimal value of the control character. For example, a tabulator would be replaced by “#009”. Note: using this option requires that [\\$EscapeControlCharactersOnReceive](#) is set to off.

space-cc

replace control characters by spaces Note: using this option requires that [\\$EscapeControlCharactersOnReceive](#) is set to off.

drop-cc

drop control characters - the resulting string will neither contain control characters, escape sequences nor any other replacement character like space. Note: using this option requires that [\\$EscapeControlCharactersOnReceive](#) is set to off.

compressspace

compresses multiple spaces (US-ASCII SP character) inside the string to a single one. This compression happens at a very late stage in processing. Most importantly, it happens after substring extraction, so **FromChar** and **ToChar** positions are **NOT** affected by this option. (available since v8.18.0)



sp-if-no-1st-sp

This option looks scary and should probably not be used by a user. For any field given, it returns either a single space character or no character at all. Field content is never returned. A space is returned if (and only if) the first character of the field's content is NOT a space. This option is kind of a hack to solve a problem rooted in RFC 3164: 3164 specifies no delimiter between the syslog tag sequence and the actual message text. Almost all implementation in fact delimit the two by a space. As of RFC 3164, this space is part of the message text itself. This leads to a problem when building the message (e.g. when writing to disk or forwarding). Should a delimiting space be included if the message does not start with one? If not, the tag is immediately followed by another non-space character, which can lead some log parsers to misinterpret what is the tag and what the message. The problem finally surfaced when the klog module was restructured and the tag correctly written. It exists with other message sources, too. The solution was the introduction of this special property replacer option. Now, the default template can contain a conditional space, which exists only if the message does not start with one. While this does not solve all issues, it should work good enough in the far majority of all cases. If you read this text and have no idea of what it is talking about - relax: this is a good indication you will never need this option. Simply forget about it ;)

secpath-drop

Drops slashes inside the field (e.g. "a/b" becomes "ab"). Useful for secure pathname generation (with dynafiles).

secpath-replace

Replace slashes inside the field by an underscore. (e.g. "a/b" becomes "a_b"). Useful for secure pathname generation (with dynafiles).

To use multiple options, simply place them one after each other with a comma delimiting them. For example "escape-cc,sp-if-no-1st-sp". If you use conflicting options together, the last one will override the previous one. For example, using "escape-cc,drop-cc" will use drop-cc and "drop-cc,escape-cc" will use escape-cc mode.

Further Links

- Article on "[Recording the Priority of Syslog Messages](#)" (describes use of templates to record severity and facility of a message)
- [Configuration file syntax](#), this is where you actually use the property replacer.
- [Property Replacer nomatch mode](#)
 - [Summary of nomatch Modes](#)

See also: Help with configuring/using Rsyslog :

- [Mailing list](#) - best route for general questions
- GitHub: [rsyslog source project](#) - detailed questions, reporting issues that are believed to be bugs with Rsyslog
- Stack Exchange ([View](#), [Ask](#)) - experimental support from rsyslog community

See also: Contributing to Rsyslog :

- Source project: [rsyslog project README](#).
- Documentation: [rsyslog-doc project README](#)



Copyright 2008-2020 [Rainer Gerhards \(Großrinderfeld\)](#), and Others.

This site uses the [“better”](#) theme for Sphinx.

About

[About Adiscon / Impressum](#)

[Contact Us](#)

[Privacy policy /
Datenschutzrichtlinien](#)

[Rainer's Blog](#)

Related Products

[LogAnalyzer](#)

[WinSyslog](#)

Copyright © 2008-2020 [Adiscon GmbH](#). Theme: [Zakra](#) By ThemeGrill.

