

NAME

rsyslog.conf – rsyslogd(8) configuration file

DESCRIPTION

The *rsyslog.conf* file is the main configuration file for the **rsyslogd**(8) which logs system messages on *nix systems. This file specifies rules for logging. For special features see the **rsyslogd**(8) manpage. Rsyslog.conf is backward-compatible with syslogd's syslog.conf file. So if you migrate from syslogd you can rename it and it should work.

Note that this version of rsyslog ships with extensive documentation in HTML format. This is provided in the *.doc* subdirectory and probably in a separate package if you installed rsyslog via a packaging system. To use rsyslog's advanced features, you **need** to look at the HTML documentation, because the man pages only cover basic aspects of operation.

MODULES

Rsyslog has a modular design. Consequently, there is a growing number of modules. See the HTML documentation for their full description.

omsnmp

SNMP trap output module

omgssapi

Output module for GSS-enabled syslog

ommysql

Output module for MySQL

omrelp Output module for the reliable RELP protocol (prevents message loss). For details, see below at *imrelp* and the HTML documentation. It can be used like this:

```
*.* :omrelp:server:port
```

```
*.* :omrelp:192.168.0.1:2514 # actual sample
```

ompgsql

Output module for PostgreSQL

omlibdbi

Generic database output module (Firebird/Interbase, MS SQL, Sybase, SQLite, Ingres, Oracle, mSQL)

imfile Input module for text files

imudp Input plugin for UDP syslog. Replaces the deprecated *-r* option. Can be used like this:

```
$ModLoad imudp
```

```
$UDPServerRun 514
```

imtcp Input plugin for plain TCP syslog. Replaces the deprecated *-t* option. Can be used like this:

```
$ModLoad imtcp
```

```
$InputTCPServerRun 514
```

imrelp Input plugin for the RELP protocol. RELP can be used instead of UDP or plain TCP syslog to provide reliable delivery of syslog messages. Please note that plain TCP syslog does NOT provide truly reliable delivery, with it messages may be lost when there is a connection problem or the server shuts down. RELP prevents message loss in those cases. It can be used like this:

```
$ModLoad imrelp
```

\$InputRELPServerRun 2514

imgssapi

Input plugin for plain TCP and GSS-enable syslog

immark

Support for mark messages

imklog Kernel logging. To include kernel log messages, you need to do

\$ModLoad imklog

Please note that the klogd daemon is no longer necessary and consequently no longer provided by the rsyslog package.

imuxsock

Unix sockets, including the system log socket. You need to specify

\$ModLoad imuxsock

in order to receive log messages from local system processes. This config directive should only left out if you know exactly what you are doing.

BASIC STRUCTURE

Lines starting with a hash mark ('#') and empty lines are ignored. Rsyslog.conf should contain following sections (sorted by recommended order in file):

Global directives

Global directives set some global properties of whole rsyslog daemon, for example size of main message queue (\$MainMessageQueueSize), loading external modules (\$ModLoad) and so on. All global directives need to be specified on a line by their own and must start with a dollar-sign. The complete list of global directives can be found in HTML documentation in doc directory or online on web pages.

Templates

Templates allow you to specify format of the logged message. They are also used for dynamic file name generation. They have to be defined before they are used in rules. For more info about templates see TEMPLATES section of this manpage.

Output channels

Output channels provide an umbrella for any type of output that the user might want. They have to be defined before they are used in rules. For more info about output channels see OUTPUT CHANNELS section of this manpage.

Rules (selector + action)

Every rule line consists of two fields, a selector field and an action field. These two fields are separated by one or more spaces or tabs. The selector field specifies a pattern of facilities and priorities belonging to the specified action.

SELECTORS

The selector field itself again consists of two parts, a facility and a priority, separated by a period ('.'). Both parts are case insensitive and can also be specified as decimal numbers, but don't do that, you have been warned. Both facilities and priorities are described in syslog(3). The names mentioned below correspond to the similar LOG_-values in /usr/include/syslog.h.

The facility is one of the following keywords: auth, authpriv, cron, daemon, kern, lpr, mail, mark, news, security (same as auth), syslog, user, uucp and local0 through local7. The keyword security should not be used anymore and mark is only for internal use and therefore should not be used in applications. Anyway, you may want to specify and redirect these messages here. The facility specifies the subsystem that produced the message, i.e. all mail programs log with the mail facility (LOG_MAIL) if they log using syslog.

The priority is one of the following keywords, in ascending order: debug, info, notice, warning, warn (same as warning), err, error (same as err), crit, alert, emerg, panic (same as emerg). The keywords error, warn and panic are deprecated and should not be used anymore. The priority defines the severity of the message.

The behavior of the original BSD syslogd is that all messages of the specified priority and higher are logged according to the given action. Rsyslogd behaves the same, but has some extensions.

In addition to the above mentioned names the rsyslogd(8) understands the following extensions: An asterisk ('*') stands for all facilities or all priorities, depending on where it is used (before or after the period). The keyword none stands for no priority of the given facility.

You can specify multiple facilities with the same priority pattern in one statement using the comma (',') operator. You may specify as much facilities as you want. Remember that only the facility part from such a statement is taken, a priority part would be skipped.

Multiple selectors may be specified for a single action using the semicolon (';') separator. Remember that each selector in the selector field is capable to overwrite the preceding ones. Using this behavior you can exclude some priorities from the pattern.

Rsyslogd has a syntax extension to the original BSD source, that makes its use more intuitively. You may precede every priority with an equals sign ('=') to specify only this single priority and not any of the above. You may also (both is valid, too) precede the priority with an exclamation mark ('!') to ignore all that priorities, either exact this one or this and any higher priority. If you use both extensions then the exclamation mark must occur before the equals sign, just use it intuitively.

ACTIONS

The action field of a rule describes what to do with the message. In general, message content is written to a kind of "logfile". But also other actions might be done, like writing to a database table or forwarding to another host.

Regular file

Typically messages are logged to real files. The file has to be specified with full pathname, beginning with a slash ('/').

Example:

```
*.* /var/log/traditionalfile.log;RSYSLOG_TraditionalFileFormat # log to a file in the traditional format
```

Note: if you would like to use high-precision timestamps in your log files, just remove the ";RSYSLOG_TraditionalFormat". That will select the default template, which, if not changed, uses RFC 3339 timestamps.

Example:

```
*.* /var/log/file.log # log to a file with RFC3339 timestamps
```

By default, files are not synced after each write. To enable syncing of log files globally, use either the "\$ActionFileEnableSync" directive or the "sync" parameter to omfile. Enabling this option degrades performance and it is advised not to enable syncing unless you know what you are doing. To selectively disable

syncing for certain files, you may prefix the file path with a minus sign ("-").

Named pipes

This version of rsyslogd(8) has support for logging output to named pipes (fifos). A fifo or named pipe can be used as a destination for log messages by prepending a pipe symbol ('|') to the name of the file. This is handy for debugging. Note that the fifo must be created with the mkfifo(1) command before rsyslogd(8) is started.

Terminal and console

If the file you specified is a tty, special tty-handling is done, same with /dev/console.

Remote machine

There are three ways to forward message: the traditional UDP transport, which is extremely lossy but standard, the plain TCP based transport which loses messages only during certain situations but is widely available and the RELP transport which does not lose messages but is currently available only as part of rsyslogd 3.15.0 and above.

To forward messages to another host via UDP, prepend the hostname with the at sign ("@"). To forward it via plain tcp, prepend two at signs ("@@"). To forward via RELP, prepend the string ":omrelp:" in front of the hostname.

Example:

```
*.* @192.168.0.1
```

In the example above, messages are forwarded via UDP to the machine 192.168.0.1, the destination port defaults to 514. Due to the nature of UDP, you will probably lose some messages in transit. If you expect high traffic volume, you can expect to lose a quite noticeable number of messages (the higher the traffic, the more likely and severe is message loss).

Sockets for forwarded messages can be bound to a specific device using the "device" option for the omfwd module.

Example:

```
action(type="omfwd" Target="192.168.0.1" Device="eth0" Port=514 Protocol="udp")
```

In the example above, messages are forwarded via UDP to the machine 192.168.0.1 at port 514 over the device eth0. TCP can be used by setting Protocol to "tcp" in the above example.

For Linux with VRF support, the device option is used to specify the VRF to send messages.

If you would like to prevent message loss, use RELP:

```
*.* :omrelp:192.168.0.1:2514
```

Note that a port number was given as there is no standard port for relp.

Keep in mind that you need to load the correct input and output plugins (see "Modules" above).

Please note that rsyslogd offers a variety of options in regarding to remote forwarding. For full details, please see the HTML documentation.

List of users

Usually critical messages are also directed to "root" on that machine. You can specify a list of users that shall get the message by simply writing ":omusrmsg:" followed by the login name. You may specify more

than one user by separating them with commas (','). If they're logged in they get the message (for example: ":omusrmsg:root,user1,user2").

Everyone logged on

Emergency messages often go to all users currently online to notify them that something strange is happening with the system. To specify this wall(1)-feature use an ":omusrmsg:*".

Database table

This allows logging of the message to a database table. By default, a MonitorWare-compatible schema is required for this to work. You can create that schema with the createDB.SQL file that came with the rsyslog package. You can also use any other schema of your liking - you just need to define a proper template and assign this template to the action.

See the HTML documentation for further details on database logging.

Discard

If the discard action is carried out, the received message is immediately discarded. Discard can be highly effective if you want to filter out some annoying messages that otherwise would fill your log files. To do that, place the discard actions early in your log files. This often plays well with property-based filters, giving you great freedom in specifying what you do not want.

Discard is just the single 'stop' command with no further parameters.

Example:

```
*.* stop # discards everything.
```

Output channel

Binds an output channel definition (see there for details) to this action. Output channel actions must start with a \$-sign, e.g. if you would like to bind your output channel definition "mychannel" to the action, use "\$mychannel". Output channels support template definitions like all other actions.

Shell execute

This executes a program in a subshell. The program is passed the template-generated message as the only command line parameter. Rsyslog waits until the program terminates and only then continues to run.

Example:

```
^program-to-execute;template
```

The program-to-execute can be any valid executable. It receives the template string as a single parameter (argv[1]).

FILTER CONDITIONS

Rsyslog offers three different types "filter conditions":

- * "traditional" severity and facility based selectors
- * property-based filters
- * expression-based filters

Selectors

Selectors are the traditional way of filtering syslog messages. They have been kept in rsyslog with their original syntax, because it is well-known, highly effective and also needed for compatibility with stock syslogd configuration files. If you just need to filter based on priority and facility, you should do this with

selector lines. They are not second-class citizens in rsyslog and offer the best performance for this job.

Property-Based Filters

Property-based filters are unique to rsyslogd. They allow to filter on any property, like HOSTNAME, syslogtag and msg.

A property-based filter must start with a colon in column 0. This tells rsyslogd that it is the new filter type. The colon must be followed by the property name, a comma, the name of the compare operation to carry out, another comma and then the value to compare against. This value must be quoted. There can be spaces and tabs between the commas. Property names and compare operations are case-sensitive, so "msg" works, while "MSG" is an invalid property name. In brief, the syntax is as follows:

```
:property, [!]compare-operation, "value"
```

The following compare-operations are currently supported:

contains

Checks if the string provided in value is contained in the property

isequal

Compares the "value" string provided and the property contents. These two values must be exactly equal to match.

startswith

Checks if the value is found exactly at the beginning of the property value

regex

Compares the property against the provided regular expression.

Expression-Based Filters

See the HTML documentation for this feature.

TEMPLATES

Every output in rsyslog uses templates - this holds true for files, user messages and so on. Templates compatible with the stock syslogd formats are hardcoded into rsyslogd. If no template is specified, we use one of these hardcoded templates. Search for "template_" in syslogd.c and you will find the hardcoded ones.

A template consists of a template directive, a name, the actual template text and optional options. A sample is:

```
$template MyTemplateName, "\7Text %property% some more text\n", <options>
```

The "\$template" is the template directive. It tells rsyslog that this line contains a template. The backslash is an escape character. For example, \7 rings the bell (this is an ASCII value), \n is a new line. The set in rsyslog is a bit restricted currently.

All text in the template is used literally, except for things within percent signs. These are properties and allow you access to the contents of the syslog message. Properties are accessed via the property replacer and it can for example pick a substring or do date-specific formatting. More on this is the PROPERTY REPLACER section of this manpage.

To escape:

```
% = \%
\ = \\ --> `\' is used to escape (as in C)
$template TraditionalFormat,"%timegenerated% %HOSTNAME% %syslogtag%%msg%\n"
```

Properties can be accessed by the property replacer (see there for details).

Please note that templates can also be used to generate selector lines with dynamic file names. For example, if you would like to split syslog messages from different hosts to different files (one per host), you can define the following template:

```
$template DynFile,"/var/log/system-%HOSTNAME%.log"
```

This template can then be used when defining an output selector line. It will result in something like `"/var/log/system-localhost.log"`

Template options

The `<options>` part is optional. It carries options influencing the template as whole. See details below. Be sure NOT to mistake template options with property options - the later ones are processed by the property replacer and apply to a SINGLE property, only (and not the whole template).

Template options are case-insensitive. Currently defined are:

- `sql` format the string suitable for a SQL statement in MySQL format. This will replace single quotes (") and the backslash character by their backslash-escaped counterpart (" and "\") inside each field. Please note that in MySQL configuration, the `NO_BACKSLASH_ESCAPES` mode must be turned off for this format to work (this is the default).
- `stdsql` format the string suitable for a SQL statement that is to be sent to a standards-compliant sql server. This will replace single quotes (") by two single quotes (") inside each field. You must use `stdsql` together with MySQL if in MySQL configuration the `NO_BACKSLASH_ESCAPES` is turned on.

Either the `sql` or `stdsql` option **MUST** be specified when a template is used for writing to a database, otherwise injection might occur. Please note that due to the unfortunate fact that several vendors have violated the sql standard and introduced their own escape methods, it is impossible to have a single option doing all the work. So you yourself must make sure you are using the right format. **If you choose the wrong one, you are still vulnerable to sql injection.**

Please note that the database writer **checks** that the `sql` option is present in the template. If it is not present, the write database action is disabled. This is to guard you against accidental forgetting it and then becoming vulnerable to SQL injection. The `sql` option can also be useful with files - especially if you want to import them into a database on another machine for performance reasons. However, do NOT use it if you do not have a real need for it - among others, it takes some toll on the processing time. Not much, but on a really busy system you might notice it ;)

The default template for the write to database action has the `sql` option set.

Template examples

Please note that the samples are split across multiple lines. A template **MUST NOT** actually be split across multiple lines.

A template that resembles traditional syslogd file output:

```
$template TraditionalFormat,"%timegenerated% %HOSTNAME%
%syslogtag%%msg:::drop-last-lf%\n"
```

A template that tells you a little more about the message:

```
$template precise,"%syslogpriority%,%syslogfacility%,%timegenerated%,%HOSTNAME%,
%syslogtag%,%msg%\n"
```

A template for RFC 3164 format:

```
$template RFC3164fmt,"<%PRI%>%TIMESTAMP% %HOSTNAME% %syslogtag%%msg%"
```

A template for the format traditionally used for user messages:

```
$template usermsg," XXXX%syslogtag%%msg%\n\r"
```

And a template with the traditional wall-message format:

```
$template wallmsg,"\r\n\7Message from syslogd@%HOSTNAME% at %timegenerated%"
```

A template that can be used for writing to a database (please note the SQL template option)

```
$template MySQLInsert,"insert iut, message, receivedat values ('%iut%', '%msg:::UPPER-
CASE%', '%timegenerated:::date-mysql%') into systemevents\r\n", SQL
```

NOTE 1: This template is embedded into core application under name **StdDBFmt** , so you don't need to define it.

NOTE 2: You have to have MySQL module installed to use this template.

OUTPUT CHANNELS

Output Channels are a new concept first introduced in rsyslog 0.9.0. As of this writing, it is most likely that they will be replaced by something different in the future. So if you use them, be prepared to change your configuration file syntax when you upgrade to a later release.

Output channels are defined via an `$outchannel` directive. It's syntax is as follows:

```
$outchannel name,file-name,max-size,action-on-max-size
```

`name` is the name of the output channel (not the file), `file-name` is the file name to be written to, `max-size` the maximum allowed size and `action-on-max-size` a command to be issued when the max size is reached. This command always has exactly one parameter. The binary is that part of `action-on-max-size` before the first space, its parameter is everything behind that space.

Keep in mind that `$outchannel` just defines a channel with "name". It does not activate it. To do so, you must use a selector line (see below). That selector line includes the channel name plus `":omfile:$"` in front of it. A sample might be:

```
*.* :omfile:$mychannel
```

PROPERTY REPLACER

The property replacer is a core component in rsyslogd's output system. A syslog message has a number of well-defined properties (see below). 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:

%proprname:fromChar:toChar:options%

proprname is the name of the property to access. **It is case-sensitive.**

Available Properties

msg the MSG part of the message (aka "the message" ;))

rawmsg

the message exactly as it was received from the socket. Should be useful for debugging.

HOSTNAME

hostname from the message

FROMHOST

hostname of the system the message was received from (in a relay chain, this is the system immediately in front of us and not necessarily the original sender)

syslogtag

TAG from the message

programname

the "static" part of the tag, as defined by BSD syslogd. For example, when TAG is "named[12345]", programname is "named".

PRI PRI part of the message - undecoded (single value)

PRI-text

the PRI part of the message in a textual form (e.g. "syslog.info")

IUT the monitorware InfoUnitType - used when talking to a MonitorWare backend (also for phpLog-Con)

syslogfacility

the facility from the message - in numerical form

syslogfacility-text

the facility from the message - in text form

syslogseverity

severity from the message - in numerical form

syslogseverity-text

severity from the message - in text form

timegenerated

timestamp when the message was RECEIVED. Always in high resolution

timereported

timestamp from the message. Resolution depends on what was provided in the message (in most cases, only seconds)

TIMESTAMP

alias for timereported

PROTOCOL-VERSION

The contents of the PROTOCOL-VERSION field from IETF draft draft-ietf-syslog-protocol

STRUCTURED-DATA

The contents of the STRUCTURED-DATA field from IETF draft draft-ietf-syslog-protocol

APP-NAME

The contents of the APP-NAME field from IETF draft draft-ietf-syslog-protocol

PROCID

The contents of the PROCID field from IETF draft draft-ietf-syslog-protocol

MSGID

The contents of the MSGID field from IETF draft draft-ietf-syslog-protocol

\$NOW The current date stamp in the format YYYY-MM-DD

\$YEAR

The current year (4-digit)

\$MONTH

The current month (2-digit)

\$DAY The current day of the month (2-digit)

\$HOUR

The current hour in military (24 hour) time (2-digit)

\$MINUTE

The current minute (2-digit)

Properties starting with a \$-sign are so-called system properties. These do NOT stem from the message but are rather internally-generated.

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 this syntax: "%msg:1:2%". If you do not wish to specify from and to, but you want to specify options, you still need to 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%"

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%".

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).

Property Options

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

uppercase

convert property to lowercase only

lowercase

convert property text to uppercase only

drop-last-lf

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

date-mysql

format as mysql date

date-rfc3164

format as RFC 3164 date

date-rfc3339

format as RFC 3339 date

escape-cc

replace control characters (ASCII value 127 and values less than 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".

space-cc

replace control characters by spaces

drop-cc drop control characters - the resulting string will neither contain control characters, escape sequences nor any other replacement character like space.

QUEUED OPERATIONS

Rsyslogd supports queued operations to handle offline outputs (like remote syslogd's or database servers being down). When running in queued mode, rsyslogd buffers messages to memory and optionally to disk (on an as-needed basis). Queues survive rsyslogd restarts.

It is highly suggested to use remote forwarding and database writing in queued mode, only.

To learn more about queued operations, see the HTML documentation.

FILES

/etc/rsyslog.conf

Configuration file for **rsyslogd**

SEE ALSO

rsyslogd(8), **logger(1)**, **syslog(3)**

The complete documentation can be found in the doc folder of the rsyslog distribution or online at

<https://www.rsyslog.com/doc/>

Please note that the man page reflects only a subset of the configuration options. Be sure to read the HTML documentation for all features and details. This is especially vital if you plan to set up a more-than-extremely-simple system.

AUTHORS

rsyslogd is taken from syslogd sources, which have been heavily modified by Rainer Gerhards (rgerhards@adiscon.com) and others.