



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

### ***Rocky Enterprise Linux 9.2 Manual Pages on command 'sudo\_logsrv.proto.5'***

**\$ man sudo\_logsrv.proto.5**

SUDO\_LOGSRV.PROTO(5)

BSD File Formats Manual

SUDO\_LOGSRV.PROTO(5)

#### **NAME**

sudo\_logsrv.proto ? Sudo log server protocol

#### **DESCRIPTION**

Starting with version 1.9.0, sudo supports sending event and I/O logs to a log server. The protocol used is written in Google's Protocol Buffers domain specific language. The EXAMPLES section includes a complete description of the protocol in Protocol Buffers format. Because there is no way to determine message boundaries when using Protocol Buffers, the wire size of each message is sent immediately preceding the message itself as a 32-bit unsigned integer in network byte order. This is referred to as ?length-prefix framing? and is how Google suggests handling the lack of message delimiters.

The protocol is made up of two basic messages, ClientMessage and ServerMessage, described below. The server must accept messages up to two megabytes in size. The server may return an error if the client tries to send a message larger than two megabytes.

#### **Client Messages**

A ClientMessage is a container used to encapsulate all the possible message types a client may send to the server.

```
message ClientMessage {
```

```
    oneof type {
```

```
        AcceptMessage accept_msg = 1;
```

```
        RejectMessage reject_msg = 2;
```

```
        ExitMessage exit_msg = 3;
```

```
        RestartMessage restart_msg = 4;
```

```

AlertMessage alert_msg = 5;
IoBuffer ttyin_buf = 6;
IoBuffer ttyout_buf = 7;
IoBuffer stdin_buf = 8;
IoBuffer stdout_buf = 9;
IoBuffer stderr_buf = 10;
ChangeWindowSize winsize_event = 11;
CommandSuspend suspend_event = 12;
ClientHello hello_msg = 13;
}

}

```

The different ClientMessage sub-messages the client may sent to the server are described be?

low.

#### TimeSpec

```

message TimeSpec {
    int64 tv_sec = 1;
    int32 tv_nsec = 2;
}

```

A TimeSpec is the equivalent of a POSIX struct timespec, containing seconds and nanoseconds members. The tv\_sec member is a 64-bit integer to support dates after the year 2038.

#### InfoMessage

```

message InfoMessage {
    message StringList {
        repeated string strings = 1;
    }
    message NumberList {
        repeated int64 numbers = 1;
    }
    string key = 1;
    oneof value {
        int64 numval = 2;
        string strval = 3;
        StringList strlistval = 4;
    }
}

```

```
    NumberList numlistval = 5;  
}  
}
```

An InfoMessage is used to represent information about the invoking user as well as the execution environment the command runs in the form of key-value pairs. The key is always a string but the value may be a 64-bit integer, a string, an array of strings, or an array of 64-bit integers. The event log data is composed of InfoMessage entries. See the EVENT LOG VARIABLES section for more information.

```
ClientHello hello_msg  
message ClientHello {  
    string client_id = 1;  
}
```

A ClientHello message consists of client information that may be sent to the server when the client first connects.

client\_id

A free-form client description. This usually includes the name and version of the client implementation.

```
AcceptMessage accept_msg  
message AcceptMessage {  
    TimeSpec submit_time = 1;  
    repeated InfoMessage info_msgs = 2;  
    bool expect_iobufs = 3;  
}
```

An AcceptMessage is sent by the client when a command is allowed by the security policy. It contains the following members:

submit\_time

The wall clock time when the command was submitted to the security policy.

info\_msgs

An array of InfoMessage describing the user who submitted the command as well as the execution environment of the command. This information is used to generate an event log entry and may also be used by server to determine where and how the I/O log is stored.

expect\_iobufs

Set to true if the server should expect IoBuffer messages to follow (for I/O log?)

ging) or false if the server should only store the event log.

If an AcceptMessage is sent, the client must not send a RejectMessage or RestartMessage.

RejectMessage reject\_msg

message RejectMessage {

TimeSpec submit\_time = 1;

string reason = 2;

repeated InfoMessage info\_msgs = 3;

}

A RejectMessage is sent by the client when a command is denied by the security policy. It

contains the following members:

submit\_time

The wall clock time when the command was submitted to the security policy.

reason The reason the security policy gave for denying the command.

info\_msgs

An array of InfoMessage describing the user who submitted the command as well as the execution environment of the command. This information is used to generate an event log entry.

If a RejectMessage is sent, the client must not send an AcceptMessage or RestartMessage.

ExitMessage exit\_msg

message ExitMessage {

TimeSpec run\_time = 1;

int32 exit\_value = 2;

bool dumped\_core = 3;

string signal = 4;

string error = 5;

}

An ExitMessage is sent by the client after the command has exited or has been terminated by

a signal. It contains the following members:

run\_time

The total amount of elapsed time since the command started, calculated using a mono? tonic clock where possible. This is not the wall clock time.

exit\_value

The command's exit value in the range 0-255.

dumped\_core

True if the command was terminated by a signal and dumped core.

signal If the command was terminated by a signal, this is set to the name of the signal without the leading ?SIG?. For example, INT, TERM, KILL, SEGV.

error A message from the client indicating that the command was terminated unexpectedly due to an error.

When performing I/O logging, the client should wait for a commit\_point corresponding to the final IoBuffer before closing the connection unless the final commit\_point has already been received.

RestartMessage restart\_msg

```
message RestartMessage {
```

```
    string log_id = 1;
```

```
    TimeSpec resume_point = 2;
```

```
}
```

A RestartMessage is sent by the client to resume sending an existing I/O log that was previously interrupted. It contains the following members:

log\_id The the server-side name for an I/O log that was previously sent to the client by the server. This may be a path name on the server or some other kind of server-side identifier.

resume\_point

The point in time after which to resume the I/O log. This is in the form of a TimeSpec representing the amount of time since the command started, not the wall clock time. The resume\_point should correspond to a commit\_point previously sent to the client by the server. If the server receives a RestartMessage containing a resume\_point it has not previously seen, an error will be returned to the client and the connection will be dropped.

If a RestartMessage is sent, the client must not send an AcceptMessage or RejectMessage.

AlertMessage alert\_msg

```
message AlertMessage {
```

```
    TimeSpec alert_time = 1;
```

```
    string reason = 2;
```

```
    repeated InfoMessage info_msgs = 3;
```

```
}
```

An AlertMessage is sent by the client to indicate a problem detected by the security policy while the command is running that should be stored in the event log. It contains the following members:

alert\_time

The wall clock time when the alert occurred.

reason The reason for the alert.

info\_msgs

An optional array of InfoMessage describing the user who submitted the command as well as the execution environment of the command. This information is used to generate an event log entry.

```
IoBuffer ttyin_buf | ttyout_buf | stdin_buf | stdout_buf | stderr_buf
```

message IoBuffer {

TimeSpec delay = 1;

bytes data = 2;

```
}
```

An IoBuffer is used to represent data from terminal input, terminal output, standard input, standard output, or standard error. It contains the following members:

delay The elapsed time since the last record in the form of a TimeSpec. The delay should be calculated using a monotonic clock where possible.

data The binary I/O log data from terminal input, terminal output, standard input, standard output, or standard error.

```
ChangeWindowSize winsize_event
```

message ChangeWindowSize {

TimeSpec delay = 1;

int32 rows = 2;

int32 cols = 3;

```
}
```

A ChangeWindowSize message is sent by the client when the terminal running the command changes size. It contains the following members:

delay The elapsed time since the last record in the form of a TimeSpec. The delay should be calculated using a monotonic clock where possible.

rows The new number of terminal rows.

cols The new number of terminal columns.

CommandSuspend suspend\_event

```
message CommandSuspend {
```

```
    TimeSpec delay = 1;
```

```
    string signal = 2;
```

```
}
```

A CommandSuspend message is sent by the client when the command is either suspended or resumed. It contains the following members:

delay The elapsed time since the last record in the form of a TimeSpec. The delay should

```
be calculated using a monotonic clock where possible.
```

signal The signal name without the leading ?SIG?. For example, STOP, TSTP, CONT.

## Server Messages

A ServerMessage is a container used to encapsulate all the possible message types the server may send to a client.

```
message ServerMessage {
```

```
    oneof type {
```

```
        ServerHello hello = 1;
```

```
        TimeSpec commit_point = 2;
```

```
        string log_id = 3;
```

```
        string error = 4;
```

```
        string abort = 5;
```

```
}
```

```
}
```

The different ServerMessage sub-messages the server may sent to the client are described below.

ServerHello hello

```
message ServerHello {
```

```
    string server_id = 1;
```

```
    string redirect = 2;
```

```
    repeated string servers = 3;
```

```
    bool subcommands = 4;
```

```
}
```

The ServerHello message consists of server information sent when the client first connects.

It contains the following members:

server\_id

A free-form server description. Usually this includes the name and version of the implementation running on the log server. This member is always present.

redirect

A host and port separated by a colon (??): that the client should connect to in? stead. The host may be a host name, an IPv4 address, or an IPv6 address in square brackets. This may be used for server load balancing. The server will disconnect after sending the ServerHello when it includes a redirect.

servers

A list of other known log servers. This can be used to implement log server redundancy and allows the client to discover all other log servers simply by connecting to one known server. This member may be omitted when there is only a single log server.

subcommands

If set, the server supports logging additional commands during a session. The client may send an AcceptMessage or RejectMessage when sudo is running in intercept mode. In this mode, commands spawned from the initial command authorized by sudo are subject to policy restrictions and/or are logged. If subcommands is false, the client must not attempt to log additional commands.

TimeSpec commit\_point

A periodic time stamp sent by the server to indicate when I/O log buffers have been committed to storage. This message is not sent after every IoBuffer but rather at a server-configurable interval. When the server receives an ExitMessage, it will respond with a commit\_point corresponding to the last received IoBuffer before closing the connection.

string log\_id

The server-side ID of the I/O log being stored, sent in response to an AcceptMessage where expect\_iobufs is true.

string error

A fatal server-side error. The server will close the connection after sending the error message.

string abort

An abort message from the server indicates that the client should kill the command and ter?

minate the session. It may be used to implement simple server-side policy. The server will close the connection after sending the abort message.

#### Protocol flow of control

The expected protocol flow is as follows:

1. Client connects to the first available server. If the client is configured to use TLS, a TLS handshake will be attempted.
2. Client sends ClientHello. This is currently optional but allows the server to detect a non-TLS connection on the TLS port.
3. Server sends ServerHello.
4. Client responds with either AcceptMessage, RejectMessage, or RestartMessage.
5. If client sent a AcceptMessage with expect\_iobufs set, server creates a new I/O log and responds with a log\_id.
6. Client sends zero or more IoBuffer messages.
7. Server periodically responds to IoBuffer messages with a commit\_point.
8. Client sends an ExitMessage when the command exits or is killed.
9. Server sends the final commit\_point if one is pending.
10. Server closes the connection. After receiving the final commit\_point, the client shuts down its side of the TLS connection if TLS is in use, and closes the connection.
11. Server shuts down its side of the TLS connection if TLS is in use, and closes the connection.

At any point, the server may send an error or abort message to the client at which point the server will close the connection. If an abort message is received, the client should terminate the running command.

#### EVENT LOG VARIABLES

AcceptMessage, AlertMessage and RejectMessage classes contain an array of InfoMessage that should contain information about the user who submitted the command as well as information about the execution environment of the command if it was accepted.

Some variables have a client, run, or submit prefix. These prefixes are used to eliminate ambiguity for variables that could apply to the client program, the user submitting the command, or the command being run. Variables with a client prefix pertain to the program performing the connection to the log server, for example sudo. Variables with a run prefix pertain to the command that the user requested be run. Variables with a submit prefix pertain to the user submitting the request (the user running sudo).

The following InfoMessage entries are required:

Key	Type	Description
command	string	command that was submitted
runuser	string	name of user the command was run as
submithost	string	name of host the command was submitted on
submituser	string	name of user submitting the command

The following InfoMessage entries are recognized, but not required:

Key	Type	Description
clientargv	StringList	client's original argument vector
clientpid	int64	client's process ID
clientppid	int64	client's parent process ID
clientsid	int64	client's terminal session ID
columns	int64	number of columns in the terminal
lines	int64	number of lines in the terminal
runargv	StringList	argument vector of command to run
runchroot	string	root directory of command to run
runcwd	string	running command's working directory
runenv	StringList	the running command's environment
rungid	int64	primary group-ID of the command
rungids	NumberList	supplementary group-IDs for the command
rungroup	string	primary group name of the command
rungroups	StringList	supplementary group names for the command
runuid	int64	run user's user-ID
submitcwd	string	submit user's current working directory
submitenv	StringList	the submit user's environment
submitgid	int64	submit user's primary group-ID
submitgids	NumberList	submit user's supplementary group-IDs
submitgroup	string	submitting user's primary group name
submitgroups	StringList	submit user's supplementary group names
submituid	int64	submit user's user-ID
ttyname	string	the terminal the command was submitted from

The server must accept other variables not listed above but may ignore them.

The Protocol Buffers description of the log server protocol is included in full below. Note that this uses the newer ?proto3? syntax.

```
syntax = "proto3";

/*
 * Client message to the server. Messages on the wire are
 * prefixed with a 32-bit size in network byte order.
 */

message ClientMessage {
    oneof type {
        AcceptMessage accept_msg = 1;
        RejectMessage reject_msg = 2;
        ExitMessage exit_msg = 3;
        RestartMessage restart_msg = 4;
        AlertMessage alert_msg = 5;
        IoBuffer ttyin_buf = 6;
        IoBuffer ttyout_buf = 7;
        IoBuffer stdin_buf = 8;
        IoBuffer stdout_buf = 9;
        IoBuffer stderr_buf = 10;
        ChangeWindowSize winsize_event = 11;
        CommandSuspend suspend_event = 12;
    }
}

/* Equivalent of POSIX struct timespec */

message TimeSpec {
    int64 tv_sec = 1;      /* seconds */
    int32 tv_nsec = 2;     /* nanoseconds */
}

/* I/O buffer with keystroke data */

message IoBuffer {
    TimeSpec delay = 1;    /* elapsed time since last record */
    bytes data = 2;        /* keystroke data */
}
```

```

/*
 * Key/value pairs, like Privilege Manager struct info.
 * The value may be a number, a string, or a list of strings.
 */

message InfoMessage {

message StringList {
    repeated string strings = 1;
}

message NumberList {
    repeated int64 numbers = 1;
}

    string key = 1;
    oneof value {
        int64 numval = 2;
        string strval = 3;
        StringList strlistval = 4;
        NumberList numlistval = 5;
    }
}
}

/*
 * Event log data for command accepted by the policy.
*/
message AcceptMessage {
    TimeSpec submit_time = 1;          /* when command was submitted */
    repeated InfoMessage info_msgs = 2; /* key,value event log data */
    bool expect_iobufs = 3;           /* true if I/O logging enabled */
}

/*
 * Event log data for command rejected by the policy.
*/
message RejectMessage {
    TimeSpec submit_time = 1;          /* when command was submitted */
    string reason = 2;                /* reason command was rejected */
}

```

```

repeated InfoMessage info_msgs = 3; /* key,value event log data */

}

/* Message sent by client when command exits. */

/* Might revisit runtime and use end_time instead */

message ExitMessage {

    TimeSpec run_time = 1;      /* total elapsed run time */

    int32 exit_value = 2;      /* 0-255 */

    bool dumped_core = 3;      /* true if command dumped core */

    string signal = 4;         /* signal name if killed by signal */

    string error = 5;          /* if killed due to other error */

}

/* Alert message, policy module-specific. */

message AlertMessage {

    TimeSpec alert_time = 1;    /* time alert message occurred */

    string reason = 2;          /* policy alert error string */

    repeated InfoMessage info_msgs = 3; /* key,value event log data */

}

/* Used to restart an existing I/O log on the server. */

message RestartMessage {

    string log_id = 1;          /* ID of log being restarted */

    TimeSpec resume_point = 2;  /* resume point (elapsed time) */

}

/* Window size change event. */

message ChangeWindowSize {

    TimeSpec delay = 1;         /* elapsed time since last record */

    int32 rows = 2;             /* new number of rows */

    int32 cols = 3;             /* new number of columns */

}

/* Command suspend/resume event. */

message CommandSuspend {

    TimeSpec delay = 1;         /* elapsed time since last record */

    string signal = 2;          /* signal that caused suspend/resume */

}

```

```

/*
 * Server messages to the client. Messages on the wire are
 * prefixed with a 32-bit size in network byte order.
 */

message ServerMessage {

oneof type {

    ServerHello hello = 1;      /* server hello message */

    TimeSpec commit_point = 2; /* cumulative time of records stored */

    string log_id = 3;         /* ID of server-side I/O log */

    string error = 4;          /* error message from server */

    string abort = 5;          /* abort message, kill command */

}

}

/* Hello message from server when client connects. */

message ServerHello {

    string server_id = 1;      /* free-form server description */

    string redirect = 2;        /* optional redirect if busy */

    repeated string servers = 3; /* optional list of known servers */

}

```

## SEE ALSO

[sudo\\_logsrvd.conf\(5\)](#), [sudoers\(5\)](#), [sudo\(8\)](#), [sudo\\_logsrvd\(8\)](#)  
[Protocol Buffers](https://developers.google.com/protocol-buffers/), <https://developers.google.com/protocol-buffers/>.

## HISTORY

See the HISTORY file in the sudo distribution (<https://www.sudo.ws/history.html>) for a brief history of sudo.

## AUTHORS

Many people have worked on sudo over the years; this version consists of code written primarily by:

Todd C. Miller

See the CONTRIBUTORS file in the sudo distribution (<https://www.sudo.ws/contributors.html>) for an exhaustive list of people who have contributed to sudo.

## BUGS

If you feel you have found a bug in sudo, please submit a bug report at

<https://bugzilla.sudo.ws/>

## SUPPORT

Limited free support is available via the sudo-users mailing list, see

<https://www.sudo.ws/mailman/listinfo/sudo-users> to subscribe or search the archives.

## DISCLAIMER

sudo is provided **?AS IS?** and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are dis**?claimed**. See the LICENSE file distributed with sudo or <https://www.sudo.ws/license.html> for complete details.

Sudo 1.9.9

January 19, 2022

Sudo 1.9.9