GO-SUSI Operator's Manual

Overview

go-susi is a daemon that performs the following functions:

- Maintain various databases for GOsa. For instance go-susi manages a database of Debian packages that GOsa presents to the user for creating package lists.
- Send messages to clients on behalf of GOsa to trigger some action such as wake-on-lan.
- Maintain a schedule of jobs and trigger their execution at the appropriate time.
- Monitor the progress of long-running jobs such as installations.
- Communicate the job schedule and progress to other servers so that each instance of GOsa can query up-to-date data from its respective go-susi instance.
- Collect data from clients and make it available to GOsa (e.g. installation logs).
- Query an LDAP directory on behalf of GOsa.
- Make changes to LDAP data on behalf of GOsa or in reaction to other events (such as job completion).

Invocation

Most of the time you would not invoke go-susi directly. The preferred way to invoke go-susi is to use appropriate Upstart/SystemD jobs.
When invoked directly, go-susi understands the following command line arguments:

-h

print usage and exit.

-V

print version and exit.

-t

send a running go-susi process an -t message, print out the returned data and exit.

-c

go-susi will read its configuration from <file> instead of the default location. If both --test and -c are specified, the last switch on the command line will determine the config file location.

-f

start with a fresh database. go-susi will not load data from

-v

go-susi will log INFO level log messages (by default only ERRORs are logged). INFO level messages may aid the administrator in debugging problems.

-vv

go-susi will log INFO and DEBUG level log messages. DEBUG level messages are useful only for developers and may produce so much data that it affects performance. They also contain cleartext passwords.

-d

- go-susi will read configuration files from <dir> instead of
- the default log file will be
- the database files will be stored in instead of
- installation/softupdate logs will be stored in instead of

Background operation

go-susi does not put itself into the background, does not create a new session and does not ignore SIGHUP. Nor does go-susi create a PID file. If you want to do any of these things from a shell script (e.g. a classic init script), use utilities such as setsid(1), nohup(1) and start-stop-daemon(8) in combination with the

shell's & operator, subshells, exec and variables like $BASHPID (which gives the PID of the subshell unlike $$). Note that none of these things is necessary when go-susi is invoked by Upstart/SystemD.

Configuration

DNS configuration

go-susi will use DNS to locate other go-susi and gosa-si instances. When go-susi starts it will contact other servers listed in DNS to update its job schedule and announce its presence so that the other servers will pass on future information to the new go-susi instance. This can be disabled via

To list a server in DNS, create an SRV record for the service "gosa-si" and protocol "tcp" within the same subdomain of the go-susi that should pick it up.

TCP Keep-Alive

In order to keep its job database consistent with its peers, go-susi needs to detect when a peer is unreachable. Error conditions that close the TCP connection to the peer, such as when the peer process crashes or is shut down, are always detected reliably. However error conditions at the network level, such as a broken network cable, can only be detected if TCP keep-alive is properly configured.
By default the Linux kernel will not send the first keep-alive packet until hours after the last data transmission. This is too long if you want go-susi to have an accurate up-to-date view of peer jobs. The following commands configure the kernel to start keep-alive when no data has been transmitted for 30s, to wait 10s between keep-alive packets and to mark the connection as broken if 5 keep-alive packets remain unanswered. This causes broken connections to be detected after about 1 ½ minutes.

Configuration file /etc/gosa-si/server.conf

/etc/gosa-si/server.conf configures various aspects of go-susi's behaviour. It has the following general structure:

go-susi evaluates the following sections/parameters from this file. All other sections/parameters are ignored.

go-susi collects all keys from all sections and will use them to decrypt messages.

The path of go-susi’s log file. Default is

The directory where the subdirectories for client log files received via will be created. Default is

The path of a program that generates a list of kernels supported for each release. See the section **list-hook** further below. Default is

The path of a program that generates a list of packages included in each release. See the section **-list-hook** further below. Default is

The path of a program to handle messages. See the section further below. Default is

(deprecated, see [tftp]//path for the replacement feature) The path of a program that will be called when go-susi's TFTP server is asked for a file that matches "". See the section **pxelinux-cfg-hook** further below. Default is

The path of a program that will be called when go-susi receives a message of type (for all kinds of "foo"). See the section **new-config-hook** further below. Default is /

The path of a program that will be called when go-susi receives a message of type (for all kinds of "foo", except "audit"). See the section **n-hook** further below. Default is

The path of a program that will be called when go-susi has completed a successful registration at an si-server. See the section **registered-hook** further below. Default is

The path of a program that will be called when go-susi receives a message. See the section **activated-hook** further below. Default is

k

The path of a program that will be called when go-susi receives a message. See the section **detect-hardware-hook** further below. Default is

The path of a program that go-susi will launch and whose output it will read continually to provide FAI progress events that go-susi will convert into messages. See the section **gress-hook** further below. Default is

The path of a program go-susi calls when it sees in the output from fai-progress-hook. The output from this program is sent as to the registration server. See the section **fai-savelog-hook** further below. Default is

The path of a program go-susi calls when it sees in the output from fai-progress-hook. The output from this program is sent as to the registration server. See the section **fai-audit-hook** further below. Default is

The port the server should listen at for XML messages. Default is 20081.

URI for connecting with the LDAP server. Default is

A DN. LDAP lookups will be restricted to the subtree rooted here. Default is

Full DN or partial DN ending in ",". In the latter case **ldap-base** will be appended. LDAP objects for new systems will be created under this base. Default is "

DN of the account to use for write access to LDAP. There is no default value. If this option is not set, go-susi will function in client-only mode, i.e. basically behave like a gosa-si-client, whereas otherwise it would behave like a combination of gosa-si-server and gosa-si-client.

Password of the account to use for LDAP write access. Default is

DN of the account to use for read access to LDAP. Default is empty which means anonymous access.

Password of the account to use for LDAP read access. Default is empty.

Name or IP address with or without port of the preferred server to register at when operating in client-only mode. If go-susi operates as a full server, it ignores this setting and only registers at itself. If no port is specified, is used.
If registration at this server fails and lists servers, these will be tried and if is true, servers listed in DNS will be tried, too.

See The option may be specified in either section, but takes precedence.

A list of port numbers separated by commas and/or whitespace. The server expects standard clients to listen on one of these ports. Clients listening on other ports will be considered test clients and some functionality will be disabled for them. The is automatically appended to this list. Default is 20083,

The TCP port go-susi will listen on for FAI status messages. Default is “disabled”.

The path of the certificate in PEM format used to authenticate certificates presented by parties connecting to go-susi via TLS. Default is

The path to the certificate in PEM format go-susi will present to the other party when connecting to them via TLS. Default is

The path to the PKCS#8 private key in PEM format that corresponds to . Default is

The UDP port for go-susi's built-in TFTP server. Default is 69. Use “disable”, “none” or anything else that is not a valid port to disable the functionality.

Every parameter in the section that starts with a slash "/" specifies a mapping from a virtual path that may be requested from the TFTP server (without the leading slash) to the data the TFTP server should send in response.
If there is nothing on the right side of the "=" for the parameter, the TFTP server will respond with a "File not found" error, however this will be logged as an INFO! rather than an ERROR!.
If the value on the right side of the "=" does *not* start with "|", it is the actual filesystem path of the file whose contents should be returned.
If the value on the right side of the "=" *does* start with "|", it is the path of a hook to execute. See the section *TFTP hooks* for details.

Like (see above) but is treated as a regular expression (including the "^" which means that the regex has to match at the start of the TFTP request).
If the value on the right side of the "=" *does* start with "|", it is the path of a hook to execute, optionally followed by arguments to pass. The stdout of that hook will be sent as response to the TFTP request.
The value to the left of the "=" may contain references to capturing groups in the regex. refer to the 1st, 2nd, 3rd,... capturing group. refers to a named capturing group. A literal $ has to be written as $ The curly braces may be omitted, but be aware that e.g. is interpreted as " which is probably not what you want.
If multiple and/or lines in the section match an incoming TFTP request, the *last* one wins.

A list of peer servers (format *host:port*) separated by commas and/or whitespace to communicate with in addition to those listed in DNS.

If , then SRV records from DNS for tcp/gosa-si servers will be ignored. Note, however, that if another server contacts go-susi of its own accord, go-susi will start talking to this peer regardless of

When looking up machine names (e.g. for wake-on-lan) that are not fully qualified, if DNS can not resolve the name, re-attempt with each of the domains from this list appended. The list’s entries may be separated by commas or spaces and each entry may or may not start with a dot.

Configuration file /etc/gosa-si/client.conf

For backwards compatibility with gosa-si-client, go-susi reads the configuration file
in addition to Both files are interpreted exactly the same.
If both are present, settings from server.conf override conflicting settings from

Configuration file /etc/gosa/ou=servers.conf

If this file exists, each line is interpreted as a DN. Whenever go-susi looks for repository servers, it will in addition to searching the complete tree under also search under each of the DNs listed in this file with one-level scope. The same configuration file is used by from the package. Use this file if you have repository server objects located in a part of the LDAP tree that is not under

Hooks

go-susi outsources several functions that are directly integrated into gosa-si-server to external programs. These hook programs are configured in [redacted]. To disable a hook completely, set it to [redacted] Do *not* use [redacted]! That will cause unnecessary ERROR messages. Only use [redacted] in places where a *non*-executable file is to be suppressed.

Hook environment

All hooks get a standard set of environment variables: [redacted] and [redacted] All of these refer to the machine running the hook.

kernel-list-hook

go-susi relies on an external program to provide the list of kernels supported for each release (see message [redacted]). go-susi calls this program after calling [redacted] and passes it the same **PackageListCacheDir** environment variable, so the [redacted] may use the cache created by [redacted]
The hook is expected to print to standard output a list in LDIF format that lists all supported releases with all supported kernels. Each supported release should have at least one entry called [redacted]. The following example demonstrates the syntax:

```
[redacted]

[redacted]

[redacted]
```

The above example lists two releases. Release [redacted] has two available kernel options. Release [redacted] on the other hand has only the [redacted] option.
The DNs are arbitrary and not evaluated by go-susi. They can even be left out completely.
The attribute names are not case-sensitive.
You can use base64-encoding with LDIF's double-colon syntax.

package-list-hook

go-susi relies on an external program to provide the list of packages included in each release and which debconf parameters they support (see message [illegible]). go-susi calls this hook on startup and when it receives the **SIGUSR2** signal. The hook is expected to print to standard output a list in LDIF format that contains the complete package database. The hook will receive the following environment variables:

PackageListCacheDir:
: If the hook maintains a package cache, it should store it in this directory. Note that this directory contains other go-susi files that the hook must not touch.

PackageListDebconf:
: When go-susi calls the hook for the first time on startup it passes the value [illegible] in this variable. This tells the hook that it should not perform time-consuming scans of packages to extract debconf templates. It should only report templates if it can do so from cached data. The purpose is to get the list of packages as quickly as possible.
When the hook has finished executing the [illegible] run, go-susi will immediately call the hook again, this time with the value [illegible] in this variable. This tells the hook that it may perform package scans to extract debconf templates, but should restrict itself to scanning packages whose entry in the Packages file includes the sub-string [illegible] on the [illegible] or [illegible] line. The purpose of this hook call is to get a mostly complete list of debconf templates as quickly as possible.
When go-susi calls the hook in reaction to a **SIGUSR2** signal, it passes the value [illegible] in this variable. At this point the hook may want to scan all packages for debconf templates to be able to report a complete list. However the hook should still return in a reasonable time. It is recommended that after a maximum of 1h running time the hook should present the results it has and make a note of any packages that are still unscanned so that it can continue the scan the next time the **SIGUSR2** signal is sent (which typically happens each night via a cron job).

PackageListFAIrepository:
: This variable contains a space separated list of all FAIrepository attributes from LDAP (e.g. [illegible] and tells the hook which repositories it should scan for packages.

Example ouput from hook:

The attributes , , and correspond directly to their counterparts from the Debian file. The attribute specifies the release (aka "distribution") to which the entry belongs. Multiple attributes may be present if multiple versions are available for the same release.
The attribute (without is accepted as an alias) is the complete contents of the file describing the debconf parameters of the package (if it has any).
The attribute, if present, informs go-susi that the entry refers to a (version of a) package that is not found in the main repository for the respective release. All values are collected and added (with a "+" prefix) to the list of available FAI classes for the corresponding release. The FAI backend recognizes these pseudo-FAI classes and creates appropriate files for them.
The hook may include groups in its output without any package information to inform go-susi about existing repository paths even if they do not (yet) contain packages.
To reduce go-susi's memory usage and improve its performance, it is possible to combine the information for multiple releases for the same package into a single entry. Such an entry must start with the attribute and must be followed by at least one attribute. In order to determine the attribute values for attribute `A` for release `R`, the lines following will be scanned for the first line and then for the first `A` attribute line after that. This and all following `A` attribute lines up to the next line are `A` attribute values for `R`.

user-msg-hook

go-susi relies on an external program to process user notifications from [illegible] and [illegible]. When a [illegible] is up for execution or when a [illegible] message is received, go-susi calls the program configured in [illegible]. The hook will receive the following environment variables:

job:
: The XML of the job entry as returned by [illegible] message (with either [illegible] or [illegible] as the outermost tag).

foo:
: For each child element [illegible] of [illegible] there will be a corresponding environment variable with the element's text content (*not* using XML escaping like [illegible]). If there are multiple child elements with the same name, their text values will be concatenated separated by newlines.

Example:

The following hook script transmits messages to users via email.

```
[illegible]

[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
```

pxelinux-cfg-hook

This configuration option is deprecated. It is converted to the following settings:

```
[
]
]
]
```

These settings are assumed to be at the very top of the configuration file so that they will only be used for requests that have no matching pattern in the [section. See the section *TFTP hooks* for more information.
If you need to disable this backwards-compatibility behaviour completely, use

```
[
]
```

TFTP hooks

When go-susi's built-in TFTP server is asked for a file that matches ▯▯ or ▯▯ in the ▯▯ section and the mapping on the right side of the "=" starts with "|", the remainder is used as the path of a hook to execute. If the program exits with 0 status its standard output is sent as response to the TFTP request.
The hook will receive the following environment variables:

tftp_request: (*always present*)
: The path requested in the TFTP request. Note that this path does not usually start with "/".

groupname:
: If the hook was called because the request was matched by a ▯▯ and ▯▯ contains named capturing groups, the captured subexpressions will be passed in environment variables whose names correspond to the group names.

macaddress:
: A named capturing group (see previous paragraph) whose name is "macaddress" gets special treatment. All characters that are not hex digits will be removed from the captured subexpression, it will be converted to lower case and 0-padded or truncated to 12 characters. Colons will be inserted to form a MAC address in canonical syntax.

dn, faistate,...:
: If a capturing group with name "macaddress" was matched (see previous paragraph) and an LDAP object exists for that MAC address, its attributes will be passed in environment variables named after the attributes (in all lower case). The ▯▯ is always present in that case. The set of other attributes may vary.

new-config-hook

When go-susi receives a [illegible] message (e.g. [illegible]) it calls this hook to update system configuration files.
Depending on the received message the hook will receive some of the following environment variables:

new_ldap_config:
When this variable is non-empty, it means that LDAP configuration should be updated. The following environment variables may be present in that case, depending on whether the corresponding elements were part of the [illegible] message that triggered the hook:
admin_base, department, ldap_base, ldap_uri, release, unit_tag:
See the description of [illegible] for details.
Note:
Multiple **ldap_uri** values may be present. If so they will be separated by newlines in the **ldap_uri** environment variable.
new_ntp_config:
When this variable is non-empty, it means that NTP configuration should be updated. The following environment variable will usually be present (but may not be in case the [illegible] did not contain any servers):
server:
If present and non-empty this is a newline-separated list of NTP servers.

trigger-action-hook

Client messages of the type t are usually intended to trigger a popup message to inform the user about an imminent action. go-susi calls the program configured as to perform these actions. The exception is , which has its own hook
The hook will receive the following environment variables in addition to the standard variables:

xml:
The XML of the t message.

foo:
For each child element of there will be a corresponding environment variable with the element's text content (*not* using XML escaping like). If there are multiple child elements with the same name, their text values will be concatenated separated by newlines.

registered-hook

Whenever go-susi has successfully registered at a server (via), it calls the hook configured as The hook will not be called for erroneous messages. Note that this does not mean that the registration server has changed. The hook will receive the following environment variables in addition to the standard variables:

xml:
The XML of the r message from the successful registration.

foo:
For each child element of there will be a corresponding environment variable with the element's text content (*not* using XML escaping like). If there are multiple child elements with the same name, their text values will be concatenated separated by newlines.

activated-hook

When go-susi receives a message, it calls the hook configured as The hook will receive the following environment variables in addition to the standard variables:

xml:
The XML of the s message.

faistate:
This reflects the element of the message. The sending server sets this to the value of the attribute of the target machine's LDAP object.

detect-hardware-hook

When go-susi receives a [illegible] message, it calls the hook configured as [illegible] The hook only receives standard environment variables. The hook is expected to print on its standard output the system's hardware configuration in LDIF format. The DN is ignored and may be omitted. The output from the hook will be sent back to the si-server in a [illegible] message.

Be careful when including attributes like [illegible] or [illegible] go-susi permits changing these via [illegible] messages.

Attention:

The hook is expected to complete quickly. If it takes too long, go-susi will time out, ignore the hook's output and send an empty [illegible] message to the server, so that a pending installation does not stall.

Example hook output:

[illegible]
[illegible]
[illegible]
[illegible]
.

fai-progress-hook

When go-susi starts up, it will launch the program configured as [illegible] and will continually read its standard output line by line and will translate each line to a corresponding [illegible] message that it will send to the si-server where it is currently registered.

Example:

[illegible]

is converted into

```
[illegible]
    [illegible]      [illegible]              </header>
    [illegible]      [illegible]                 </source>
    [illegible]      [illegible]               </target>
    [illegible]           GIN>[illegible]      </CLMSG_TASKBEGIN>
    [illegible]          [illegible]              </macaddress>
    [illegible]        [illegible]              </timestamp>
[illegible]
```

Example:

[illegible]

is converted into

```
[illegible]
    [illegible]      [illegible]              </header>
    [illegible]      [illegible]                 </source>
    [illegible]      [illegible]               </target>
    [illegible]           ROR>
    [illegible]
    [illegible]           RROR>
    [illegible]          [illegible]              </macaddress>
    [illegible]        [illegible]              </timestamp>
[illegible]
```

fai-savelog-hook

go-susi launches the program configured as [illegible] when it sees [illegible] in the output from [illegible] If the hook writes to its standard output data in the format described below, it will be passed on to the registration server in a [illegible] message. When the logs have been transferred to the server, the hook will receive a newline on its standard input.

Hook output example:

```
[illegible]
[illegible]
...
[illegible]
```

Format Explanation:

Each line in the output begins with the string [illegible] followed by the name of a log file, followed by [illegible] followed by the base64 encoded contents of the log file. Newline signals the end of the base64 block, so make sure that the base64 encoder used does not wrap the data. The final line of the output has to be either [illegible] or [illegible] and specifies the type of job the logs are for. The final line has to be terminated by a newline. Any further output past that will be ignored.

fai-audit-hook

go-susi launches the program configured as [illegible] when it sees [illegible] in the output from [illegible] The hook works the same way as [illegible] except that the final line of output has to be [illegible]
While there are no restrictions on the file names and formats that may be transmitted to the server using this hook, only XML files with the [illegible] file extension that have the proper audit file format can be queried using [illegible] and [illegible]

Interfacing with go-susi

TCP connection to [server]/port

The main way of accessing go-susi's functionality is by connecting to it via TCP on the port configured in [server] and sending XML-formatted requests. Depending on the kind of request go-susi may return an XML-formatted answer. Separate go-susi instances will communicate via TCP connections. GOsa also uses TCP to interface with go-susi.

TCP connection to [log]/port

go-susi listens on the TCP port configured as [log] and logs messages received there if the loglevel is set to DEBUG. Nothing else is done with these messages.

TFTP server on [tftp] port (UDP)

go-susi runs a read-only TFTP service. This service supports the [tftp] and [tftp] options. It serves 2 kinds of files. The first are those pre-configured in the [tftp] section of the configuration. In addition to these, go-susi will call the [tftp] program to generate [tftp] configuration files on the fly. See the section **pxelinux-cfg-hook** for more details.

Signals

[tftp] - this signal causes go-susi to call the **kernel-list-hook** and **package-list-hook** programs to rebuild the kernel and packages databases. While the databases are being rebuilt, the old data continues to be available. This is different from gosa-si-server.

[tftp] - ignored

[tftp] **SIGTERM** - these signals cause a clean shutdown of go-susi after all persistent databases have been saved.

[tftp] - causes an immediate unclean exit with backtraces of all goroutines. Useful for debugging.

database

go-susi maintains its database in memory but copies changes to XML files stored in the directory . When go-susi starts up, it will populate its internal database with the data from these files. This allows data to be persistent across restarts of the go-susi daemon.

Tip:

Use the command line tool to view database files. E.g.:

jobdb.xml

This file stores jobs scheduled to be executed at a later time as well as progress information for long-running jobs such as installations.

Each entry in has the same structure as an element from a reply to the message. See that message's description for details.

serverdb.xml

This file stores the other servers known to go-susi as well as the current encryption keys used for communicating with them.

Each entry in has the same structure as a message. See that message's description for details. Note, however, that only and are required elements in a entry. The other elements described at may or may not be present.

clientdb.xml

This file stores both the clients registered at this server as well as clients registered at peers.

Each entry in has the same structure as a message. See that message's description for details.

Messages

This chapter lists the XML messages exchanged between the various parties communicating with go-susi. The messages are grouped into topics with their own sub-sections.

Modern communication protocol (go-susi only)

This communication protocol is not compatible with gosa-si. It offers better security and should be used whenever compatibility with legacy clients is not required. In order to use the modern protocol you need to create and configure TLS certificates.

Communication begins with one party connecting to go-susi's TCP port. The party that initiates the connection sends the string [illegible] followed by E (N) or R (N). Then both parties perform the TLS handshake. The party that initiated the connection acts as the client and the other party as the server. These designations are purely for purposes of the TLS handshake. The party initiating the connection will often be a server instance of go-susi but will act as the TLS client during the TLS handshake.
TLS client authentication is required in all cases. This means that both the party initiating the connection and the other party need to present a certificate to the other party. Certificate verification is symmetrical as far as go-susi is confirmed. Both parties perform the exact same checks on each other's certificate.

Once TLS handshake has succeeded, the party that initiated the connection may send messages and potentially received replies. Each message and reply is an XML fragment sent as plain text in UTF-8 encoding. Each message or reply is terminated by E (N) or R (N). The message itself must not contain any "\n" characters, but other whitespace characters are permitted.
Multiple messages may be sent over the same TCP connection. It is the responsibility of the process initiating the connection (i.e. the sender of the 1st message) to close it when it has finished sending all of its messages and reading all replies. However, if any party detects an error, it should drop the connection as soon as possible (after possibly returning an error reply) to avoid lockups.

Security notes: The protocol intentionally does not provide for a downgrading mechanism if TLS negotiation fails or the receiving party does not understand "STARTTLS". Such downgrading would be a security risk. If a modern go-susi is configured without certificates (and hence cannot do TLS) and receives a "STARTTLS" it will treat it as a garbage message and will drop the connection. If a modern go-susi initiates a TLS connection and it fails, it will not try again without TLS even if [illegible] lines are present in the configuration. However, if a modern go-susi receives a connection without "STARTTLS" it will remember that the connecting party does not understand TLS and will use the legacy protocol when contacting that party, provided that the configuration file provides keys for the legacy protocol.

sibridge has a more lenient approach when contacting the target server. If both certificates and legacy keys are configured, sibridge will attempt a TLS connection first and if that fails will try to contact the target server with the legacy protocol.

Legacy communication protocol (compatible with gosa-si)

This communication protocol is still supported by go-susi if the configuration file contains at least one [illegible] line. Because this weakens security, it is recommended to disable the legacy protocol by removing all [illegible] lines from the configuration, unless you have actual legacy clients on your network.

Every message and reply is the base64-encoding of an encrypted XML fragment. The encryption key used

depends on the kind of message. See the code in [illegible] in the functions [illegible] and [illegible] [illegible] for details on the encryption scheme.
Each message or reply is terminated by [illegible] ([illegible]) or [illegible] ([illegible]) following the base64-text (which must not be broken by whitespace). It is permissible to send multiple messages over the same TCP connection. It is the responsibility of the process initiating the connection (i.e. the sender of the 1st message) to close it when it has finished sending all of its messages and reading all replies. However, if any party detects an error, it should drop the connection as soon as possible (after possibly returning an error reply) to avoid lockups.

Message structure and common elements

Note that for readability all XML in this manual will be formatted with line breaks. However embedded line breaks are not permitted in messages when transmitted with the modern protocol.

Most messages follow a common structure and share common elements.

```
[illegible]
    [illegible] [illegible]          </header>
    [illegible] [illegible]  </source>
    [illegible] [illegible]                </target>
[illegible]
```

The message elements have the following meaning:

<header>

This identifies the type of message. Most headers have a [illegible] component that a group of related messages share. A frequent prefix is [illegible] which is used by messages sent by GOsa.

<source>

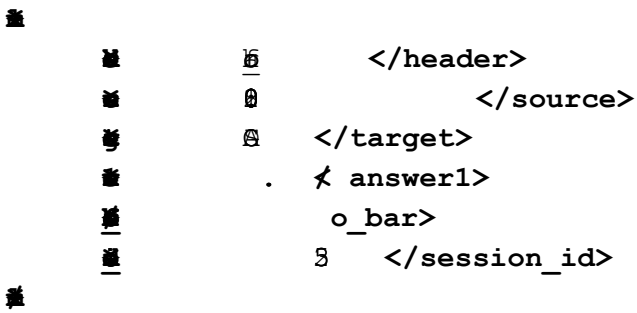
The sender of the message. Most of the time this is an [illegible] address of an si-server or client. However if the message is sent by GOsa, the **<source>** is [illegible]
Developers wishing to interface with go-susi should always fill in an [illegible] address with the proper IP of the sending computer. Because the **<source>** is used as a key in some places the same sender should always use the exact same **<source>**.

<target>

The counterpart of **<source>**. Most of the time this is simply the [illegible] of the recipient, but some message types have the MAC address of an affected system in the **<target>** element and GOsa, for whatever reasons, usually sets **<target>** to [illegible], even though this makes no sense.
Because **<target>** is *not* used for message forwarding, it is basically useless.
Developers wishing to interface with go-susi should always fill in the [illegible] address of the go-susi process they target.

Reply structure

The term reply as used in this manual refers only to something that is sent back over the same TCP connection as the message the reply refers to. Most messages do not have replies and the sender typically closes the connection after sending the message. Some messages trigger return messages sent over a new connection initiated by the recipient of the first message. These return messages are not referred to as replies in this manual.
Like messages in general, most replies follow a common structure.



The message elements have the following meaning:

<header>

The header is often derived from the message the reply is answering by dropping the

<source>, <target>

While these are often simply the swapped values of the corresponding elements from the initial message, as the example shows they can be even more meaningless.

<foo_bar>

Many replies feature empty elements named like the header.

<session_id>

An implementation detail from gosa-si-server that may be useful in log entries. For compatibility reasons go-susi adds this element to replies but the contained value is arbitrary.

<answer1>

When a reply may contain multiple datasets with the same structure, they are usually wrapped in **<answerX>** elements where **X** is the index of the answer.

Error replies

If an error occurs while processing a request, the si-server may send an error reply. Usually this is only done for messages that have regular replies, in particular for messages sent by GOsa. GOsa understands error replies with the following structure and may present the contained error message as a popup to the user.

```
[illegible]
        [illegible]    [illegible]    </header>
        [illegible]    [illegible]                    </source>
        [illegible]    [illegible]  </target>
        [illegible]        >[illegible]                </error_string>
        [illegible]      1[illegible]  swer1>
[illegible]
```

The message elements have the following meaning:

<header> always [illegible]

<source> the sending server

<target> always [illegible]

<answer1> always "1".

<error_string>

A human-readable description of the error. The presence of this element, and not <answer1>1</answer1>, is the best indicator of an error.

Encryption keys

Messages are encrypted with different keys depending on sender and recipient. Unlike gosa-si-server go-susi accepts all messages encrypted with all keys. When sending, go-susi will follow the gosa-si-server rules on choosing the key to encrypt the message.

GOsa ↔ Server:

Messages by GOsa to the si-server and the server's replies are encrypted with the **[GOsaPackages]** key.

Server ➔ Client:

Messages sent by the si-server to si-clients registered at that server are always encrypted with the key from the most recent [illegible] or [illegible] message sent by the client.

Server ➔ Server:

Messages sent from one server to another are encrypted with the most recent key exchanged between sender and receiver via [illegible] messages.
The only exception to this rule is the [illegible] message which is encrypted with the **[ServerPackages]** key, because when it is sent there is not yet an agreed key.

Client ➔ Server:

Messages sent by an si-client to the si-server where it is registered are encrypted with the key from the most recent [illegible] or [illegible] message sent by the client. The only exception to this rule is the [illegible] message itself which is encrypted with the **[ClientPackages]** key.

Jobs

One of the core features of the si-server is to keep a persistent database of jobs. Each job has a that determines when the job is to be executed and an field that identifies the server responsible for launching that job when its time has come. Depending on the job's type there may be other fields. Usually there is a that specifies the machine to be affected by the job.

Jobs are created by GOsa in response to user actions. There is a distinction between jobs to be executed immediately and jobs to be launched at a future time. However, as far as the messages are concerned the differences are so minor that go-susi (but not gosa-si-server) treats the two forms as synonyms.

Jobs are stored in the jobdb, which in go-susi is stored in a file described further above. The message can be used to extract data from the jobdb. GOsa uses this message for its deployment status page.

The message can modify the fields of an existing job and the message can be used to remove jobs. Note that GOsa does not allow the removal of running install jobs. Instead GOsa offers to cancel them. It does this by sending a message.

When it's time to execute a job, the si-server first notifies the target machine (if there is one and it's reachable) of the job by sending a message. The target machine may react to this message by presenting a logged in user with a popup asking the user to log out.

A job has a lifecycle that is reflected in its and fields which are documented at the message The job starts out as , progresses to and is removed from the jobdb when its becomes The field is only used for long-running jobs like installations.

Jobs are an important part of server-to-server communication. The relevant message is described not in the present section but the one on server-to-server communication.

A special role is played by the message which is used by GOsa for its CSV-import feature. go-susi reacts to this message by creating an LDAP object for the target system if there isn't one, as well as creating an installation job and activating the system. With go-susi effectively acts as a combination of and gosa-si-server handles things differently.

GOsa's feature for sending messages to users and/or groups results in messages which differ from ordinary job messages in that they do not target a machine. Other than that they are treated like other jobs, are stored in the jobdb and can be read with modified with and removed with

job_trigger_action_*

gosa_trigger_action_*

Purpose:

GOsa↔Server. Schedule a job for execution at a later time ([illegible] *) or execute it at once [illegible] The server that gets this message from GOsa will tell peer servers about the new job via [illegible] [illegible] .

When the job's time has come, a matching [illegible] message will be sent to the affected client (if it is reachable).

Example message:

```
[illegible]
    [illegible]  [illegible]                    </header>
    [illegible]  [illegible]  </source>
    [illegible]  [illegible]               </target>
    [illegible]     [illegible] [illegible]        </timestamp>
    [illegible]      [illegible]                 </macaddress>
    [illegible]    [illegible]     </periodic>
[illegible]
```

The message elements have the following meaning:

<header> identifies the kind of job. The following jobs are supported:

- ***_trigger_action_halt**
  - tell client to shut down (allowing logged in users to log out first)
- ***_trigger_action_reboot**
  - tell client to reboot (allowing logged in users to log out first)
- ***_trigger_action_audit**
  - tell client to perform an audit
- ***_trigger_action_faireboot**
  - abort FAI operation (e.g. installation) in progress
- ***_trigger_action_reinstall**
  - set faiState to "install"; tell users to log out; wake client if necessary
- ***_trigger_action_update**
  - set faiState to "softupdate"; tell users to log out; wake client if necs.
- ***_trigger_action_localboot**
  - set faiState to "localboot" and remove pending install/softupdate jobs
- ***_trigger_action_wake**
  - send wake-on-lan (WOL) to target
- ***_trigger_action_lock**
  - set gotoMode to locked
- ***_trigger_action_activate**
  - set gotoMode "active"; send [illegible]

<header> will become **<headertag>** in [illegible] etc.

<macaddress>, <target>

At least one of these elements must contain a valid MAC address that identifies the machine to be affected by the job. If **<macaddress>** is present, it will be preferred.

The **<target>** value will become **<targettag>** in [illegible] etc.

<timestamp> *(optional)*

When the job should be executed. The format is "YYYYMMDDHHMMSS". The time is local time of the server that receives the message and takes time zone (in particular daylight saving time) into account. IOW the job will be executed at the earliest time that the server's clock has a value greater than the job's timestamp.

If **<timestamp>** is missing, the timestamp will be considered to be "now" (meaning the job will be executed as soon as possible). " " messages do not usually have a timestamp.

<periodic> *(optional)*

The job will be repeated in regular intervals. The format is a *number* followed by " " followed by either or

Also permitted is **<periodic></periodic>** which is the same as not having **<periodic>**.

If a job is scheduled with **<periodic>** it will be scheduled to run for the first time at the time specified in **<timestamp>**. It will run for the 2nd time at the time that results from adding the **<periodic>** duration to the timestamp, for the 3rd time at the time that results from adding the duration to the timestamp of the 2nd time, and so on. If, at the time the job finishes, one or more of the repeat times have already passed, they will be skipped. For example, if a job is scheduled every and the 1st run takes 1 ½ minutes, the 2nd run will be skipped and the next run will be the 3rd which will start ½ minute after the end of the 1st run.

Note that using the unit is different from using multiples of 24 because of daylight savings time which may cause days to be longer or shorter. A is the same as 7

Note that using the unit " " is not the same as any particular number of days because months vary in length. The same applies to E.g. a job scheduled with **<timestamp>** 0 **</timestamp>** and **<periodic></periodic>** will run on 2012-01-01, then 2012-02-01, then 2012-03-01,...

29 February will wrap around to 1 March during a non-leap year and from that point on the timestamp will stay at the 1st of the month, even during future leap years. 31 will wrap around to 1 during short months and will stay at 1.

IOW, never schedule jobs with a periodic unit of on days >= 29.

gosa-si-server notes:

This message is handled in .

gosa-si-server does not return a reply to the messages only to

go-susi notes:

<periodic> units and are go-susi extensions not supported by gosa-si. The unit is for testing only.

go-susi treats and the same. In particular go-susi will send messages for both. gosa-si doesn't do this for messages.

gosa-si-server checks its foreign_clients database and if it finds that the affected client is registered at a peer server it will forward the job request to the peer with an added element. go-susi does not do this at the time it receives the message but waits until the time the job is up for execution and then forwards the job if necessary.

go-susi treats as a synonym for .

go-susi treats different from gosa-si-server. gosa-si-server locks the machine and reboots it, but keeps it in (if it is installing). The job does not actually disappear. While this makes sense for the name of the job, it doesn't match how GOsa uses this job. GOsa sends when "Abort" is selected in the jobs overview. To better match GOsa's use of this job, go-susi interprets "faireboot" as "abort job".

Example reply:

```
</header>
</source>
</target>
swer1>
</session_id>
```

The reply elements have the following meaning:

<answer1> 0 if the job was successfully added to the jobdb or a numeric error code if there was a problem. In the latter case there will be an **<error_string>** element with a human language description of the error.

GOsa notes:

GOsa sends this message in . The string comes from or which are defined in and whose values are set in the respective subclass, e.g.

GOsa ignores the reply and in fact appears to close the connection after sending the message, so that the reply cannot even be delivered.

job_trigger_activate_new

Purpose:

GOsa➔Server. Create a new system object and matching install job. This message is sent by GOsa for each entry when using the CSV import feature. It's typically used to import a whole batch of new systems ready for installation, so that they only need to be plugged in and turned on.

Example message:

```
</header>
</source>
</target>
</timestamp>
</ip>
</fqdn>
</ogroup>
</base>
</mac>
</macaddress>
```

The message elements have the following meaning:

<timestamp> *(optional)*

A job with this **<timestamp>** will be created. If **<timestamp>** is missing, it defaults to now.

<base> *(optional)*

The new system object will be put into or depending on whether it is or (based on the **<ogroup>** information or an existing object). If no information about the system's type is available, it will be put in

If an object with the given MAC address already exists elsewhere, it will be moved to the new location.

If **<base>** is missing and **<ogroup>** is an object group, the system will be put into the same as the alphabetically first member system of the object group (even if that is not called or).

If **<base>** is missing and **<ogroup>** is also missing, **<base>** will default to for new entries and keep the old location if the system already exists.

If **<base>** is missing and **<ogroup>** is a system, the new system object will be put into the same as the template system (even if that is not called or).

<mac>, <macaddress>, <target>

One of these elements must be present. If the message has both **<mac>** and **<macaddress>**, **<mac>** will take precedence over **<macaddress>** and an error will be logged if they differ.

If either **<mac>** or **<macaddress>** is present, **<target>** is ignored. Otherwise it will be used as MAC and it is a fatal error if it is not a valid MAC.

<ogroup> *(optional)*

Either the plain or full-qualified name of a system (i.e. LDAP object with) to use as a template or the name of an object group (i.e. LDAP object with).

If **<ogroup>** is an object group, its member list will be sorted alphabetically and the first member will be used as template system.

If no LDAP object exists for the given MAC address, an object will created with a generated name and the relevant attributes will be copied from the template system.

The [illegible] will always be set to [illegible], irrespective of the template object's value.

If **<ogroup>** is missing and there is no existing LDAP object, an incomplete entry will be created in [illegible]

<ip> *(optional)*

If a new system object is created, its [illegible] will be set to this value. This is mainly useful to make WOL more reliable by telling the server about the subnet of the new system.

<fqdn> *(optional)*

If **<ip>** is missing but **<fqdn>** is present, the **<fqdn>** will be resolved to an IP via DNS. Note that the **<fqdn>** will *not* be used to name the generated LDAP entry. Like **<ip>** the purpose of **<fqdn>** is to make WOL more reliable.

<dhcp> ignored.

gosa-si-server notes:

gosa-si-server ignores **<dhcp>** and **<target>**.

go-susi notes:

go-susi's handling of this message is different from gosa-si-server's in the following respects:

- go-susi creates the system's LDAP object immediately with a generated name, whereas gosa-si-server does not create a system object until the system contacts the server.
- go-susi sends a single WOL when the install job triggers at the given **<timestamp>**, whereas gosa-si-server keeps sending WOLs until the system contacts the server.
- **<ogroup>** can be the name of a system to use as template, whereas gosa-si-server requires it to be an object group.
- If **<base>** is missing and **<ogroup>** is an object group, gosa-si-server will derive **<base>** from **<ogroup>**'s DN whereas go-susi derives **<base>** from the group's first member.
- go-susi creates a normal [illegible] job when it receives this message.

Example reply:

see [illegible]

gosa_set_activated_for_installation

job_set_activated_for_installation

Purpose:

GOsa➔Server. When go-susi executes this job, it treats this as a synonym for [illegible] except that there's no reply. However, when go-susi forwards this job to another server, it is forwarded with its original header. Because gosa-si-server treats the 2 messages differently, it is important that the correct message is used if there are gosa-si-server peers.

go-susi note:

go-susi accepts the alias [illegible] and supports all of the other elements possible for jobs such as **<periodic>**. gosa-si-server supports only the [illegible] form with no job planning elements.

Example message:

```
[illegible]
    [illegible]    [illegible]                    </header>
    [illegible]    [illegible] [illegible]   e>
    [illegible]    [illegible]              </target>
    [illegible]        [illegible]              </macaddress>
[illegible]
```

job_send_user_msg

Purpose:

GOsa↔Server. Send a text message to one or more users and/or groups of users.

go-susi note:

go-susi does not implement messaging services directly. When the job's execution time as determined by **<timestamp>** has come, the external program configured as will be executed. See the manual section on that hook for details.

Example message:

```
</header>
</source>
</target>
</from>
ser>
ser>
.
</group>
</group>
.
</subject>
</message>
</timestamp>
e>
</delivery_time>
</periodic>
</macaddress>
```

The message elements have the following meaning:

<header>, **<periodic>** and **<timestamp>** are the only elements used by go-susi and they have the same meaning as for .

<macaddress> will be replaced with the go-susi server's MAC.

Example reply:

gosa_query_jobdb

Purpose:

GOsa↔Server. Returns all entries from the jobdb that match a given filter.

Example message:

```
                              </header>
              arget>
              ource>


              tor>  </connector>
              >
              erator>  </operator>
              caddress>               </macaddress>
              e>


              derby>

         0     m>
         9    </to>


```

The message elements have the following meaning:

<source>,<target> always "GOSA"

<where> *(exactly 1)* The filter that selects the jobs to return.

<clause> *(0 or more)*

A filter condition. All **<clause>** filter conditions within **<where>** are ANDed. If no **<clause>** element is present, all datasets will be selected.

<connector> *(0 or 1)*

If not provided, the default connector is "AND". All **<phrase>** filter conditions within a **<clause>** are combined by this operator like this:

$$P_1\ c\ P_2\ c\ P_3\ c\ \ldots\ P_n$$

where P_i are the phrase filters and *c* is the connector. Possible values for **<connector>** are and (The case of the word doesn't matter).

<phrase> *(0 or more)*

A single primitive filter condition. In addition to one **<operator>** element (see below) a **<phrase>** must contain exactly one other element. The element's name specifies the column name in the database and the element's text content the value to compare against. The comparison is performed according to **<operator>**.

In the case of the jobdb, the valid elements inside **<phrase>** are **<id>**, **<timestamp>**, **<status>**, **<result>**, **<progress>**, **<headertag>**, **<targettag>**, **<xmlmessage>**, **<macaddress>**, **<plainname>**, **<siserver>** and **<modified>**.

<operator> *(optional, assumed to be "eq" if missing)*

The comparison operator for the **<phrase>**. Permitted operators are with their obvious meanings, as well as . The case of the operator name doesn't matter.

performs a case-insensitive match against a pattern that may include "%" to match any

sequence of 0 or more characters and "_" to match exactly one character. A literal "%" or "_" cannot be embedded in such a pattern. [illegible] is the negation of [illegible].

The operators [illegible] [illegible] and [illegible] will attempt to convert their arguments to numbers and do a numeric comparison. If that fails, a string comparison is performed. The other operators always perform string comparison. go-susi performs all string comparisons case-insensitive, but gosa-si is case-sensitive (which is a gotcha especially with respect to MAC addresses).

The comparison is performed with the database value as the first operand. I.e. the [illegible] operator will return datasets whose value for the respective column is greater than the comparison value from **<phrase>**.

<orderby> *(optional)*

The name of the column of the database to use for sorting the results, optionally followed by "ASC" or "DESC" for ascending or descending sort.

<limit> *(optional)* Requests that only parts of the result set are returned.

<from>

After sorting query results by **<orderby>**, skip the first N results (i.e. do not include them in the reply) where N is the integer inside **<from>**. Note: The name **<from>** is misleading because it suggests that N is the value of the first item to be returned. A negative value is treated like 0.

<to>

Return a maximum of N results where N is the integer inside **<to>**. A negative value means no limit. Note: The name **<to>** is misleading since it suggests that N is the value of the last item to be returned.

gosa-si-server notes:

gosa-si-server probably permits any **<connector>** that results in a valid SQL statement, not just [illegible] and [illegible]. But I have not seen this used in the wild, so go-susi doesn't implement any other connectors.

The implementation of [illegible] is found in the file [illegible]. The parsing of the XML filter into an SQL statement is done in

[illegible]

[illegible]

The names **<from>** and **<to>** suggest that these tags are intended to specify a range of values to return but the actual implementation translates **<from>** to SQL’s OFFSET and **<to>** to SQL’s LIMIT.

The **<operator>** is case-insensitive in go-susi (i.e. "eQ" means the same as "eq") but gosa-si-server requires operators to be lowercase.

gosa-si-server does not support the **<operator>** [illegible]

go-susi performs all string comparisons case-insensitive, but gosa-si is case-sensitive (which is a gotcha especially with respect to MAC addresses).

GOsa notes:

GOsa sends these requests in

[illegible] which is called by

[illegible]

Example reply (empty jobdb):

```
</header>
</source>
arget>
</session_id>
```

Example reply (2 jobs):

```
</header>
</source>
arget>
</plainname>
</progress>
</status>
</siserver>
0</modified>
</targettag>
</macaddress>
</timestamp>
1
d>1</original_id>
</headertag>
</result>
e>
</plainname>
</progress>
</status>
</siserver>
0</modified>
</targettag>
</macaddress>
</timestamp>
</periodic>
2
</headertag>
</result>
```

```
e>
</session_id>
```

where the base64-encoded <xmlmessage> strings are

```
</header>
ource>
</target>
</timestamp>
</macaddress>
```

and

```
</header>
ource>
</target>
</timestamp>
</periodic>
</macaddress>
```

The reply elements have the following meaning:

<plainname> The name (without domain) of the machine affected by the job.
<progress> Possible values:

- The job has not started yet.
- A message has been sent to the client.
- has been received from client.
- An integer between 1 and 100 (inclusive) gives a percentage of how far along the installation has progressed.

<status> Possible values:

- No action has been performed yet. The server is waiting for the time specified by the job's **<timestamp>**.
- The server has started taking action on the job.
- gosa-si uses this to precede "done" in some cases. Not used by go-susi.
- **FIXME???**
- The job has completed. gosa-si allows finished jobs to be observed for a short period of time. go-susi however removes them immediately, so that the "done" status can not be observed. Note that a message can of course contain a "done" status.
- Something went wrong. **<result>** contains more details.

<siserver>
The listen address (IP:port) of the server responsible for processing the job. When gosa-si-server sends this message, this can also be the word “localhost”. go-susi never returns "localhost".
<modified> Always "1". Ignore.

<targettag>

The **<target>** from the message that created the job. This is typically the same as **<macaddress>**.

<macaddress> The affected machine's MAC address.

<timestamp>

When the job should be executed. The format is "YYYYMMDDHHMMSS". The time is local time of the server responsible for the job (see **<siserver>**) and takes daylight saving time into account. IOW the job will be executed at the earliest time that the responsible server's clock has a value greater than the job's timestamp.

<periodic> (*optional*)

The format is a *number* followed by "_" followed by either or Also possible is **<periodic>** **</periodic>** which is the same as **<periodic>** not being present.

See for more information on the exact interpretation of **<periodic>**.

<id>

A string that identifies the job (not necessarily a number; in particular not a 32bit number). Jobs are *not* reindexed when a job is deleted, so some ids may be "missing". In particular the **<id>** is *not* identical to the XX in **<answerXX>**. Answers are always numbered starting from 1 with no missing numbers.

<original_id> *(optional)*

the **<id>** of the job on the responsible **<siserver>** (which may be different from **<id>**).

<headertag>

Identifies the type of job. Derived from the <header> of the message that created the job by removing the prefix. E.g. ⇒

<result> Further information about the job. Possible values:

"none": No information available.

error description: If **<status>** **</status>**, this is a human-readable description with more information.

"TASKBEGIN foo" where "foo" is the most recent task sent by the client in a message during installation.

<xmlmessage>

base64-encoded <xml>...</xml> message that was used to add this job to the database, usually a message.

gosa-si-server notes:

New jobs always get the highest existing job **<id>** + 1. Because unfortunately GOsa uses the id column to identify a job when it sends [illegible] this seems to allow for a race condition where user X wants to delete the highest-numbered job but before the request reaches the gosa-si-server another user Y deletes said job and adds a new job. This would cause user X to incorrectly delete user Y's new job.

go-susi does not have this problem because it doesn't repeat **<id>** values.

gosa-si-server splits up the **<xmlmessage>** with whitespace which breaks base64-decoding unless the whitespace is removed first.

gosa-si-server does not report **<original_id>**.

gosa-si treats **<macaddress>** as case-sensitive.

GOsa notes:

AFAICT GOsa doesn't care about the reply's **<source>**, **<target>** and **<session_id>**.

AFAICT GOsa doesn't care about **<targettag>** and there's no other use I can find. IOW it's completely useless.

gosa_delete_jobdb_entry

Purpose:

GOsa↔Server. Remove planned jobs from the database. Sends with **<status>**done**</status>** and **<periodic>**none**</periodic>** (or no **<periodic>** at all) to peer servers to make them remove the jobs from their databases as well.

Example message:

```
                                   </header>
              </target>
              </source>

                  >
<                 connector>  </connector>
<                 phrase>
                      <operator>  </operator>
                      <id>1</id>
<                 /phrase>
                  e>

```

The message elements have the following meaning:

Aside from the **<header>** the elements are the same as for .

Example reply:

```
        </header>
              </source>
     </target>
   0   swer1>
        </session_id>
```

The reply elements have the following meaning:

<answer1> 0 if the jobs were successfully deleted or a numeric error code if there was a problem. In the latter case there will be an **<error_string>** element with a human language description of the error.
If no job matches the query that is not an error.

gosa-si-server notes:

Some versions of gosa-si-server return broken replies to this message. The contained answer element is unusable, e.g. **<answer1>** **</answer1>**.

go-susi notes:

When the job to be deleted is another server's responsibility, go-susi will forward the deletion request to that server. go-susi will not remove the job from its own database unless the responsible server reacts to the forwarded request. This means that jobs from servers that are down cannot be deleted. However, when go-susi cannot establish a connection to a server for some time, it will automatically purge that server's jobs from its database.

GOsa notes:

This message is sent in .
GOsa always uses the id column to identify the jobs. See the notes for
regarding **<id>**.
GOsa does not care about the actual reply. The only requirement is that the outer-most tag must be either **<xml>** or **<count>**. If it isn't GOsa will log an error.

gosa_update_status_jobdb_entry

Purpose:

GOsa↔Server. Change properties of a scheduled job. Sends with the new data to peer servers.

Example message:

```
                                        </header>
                </target>
                </source>



              nnector> </connector>
              rase>
                 <operator> </operator>
                 <id>1</id>
              hrase>



                    p>              </timestamp>


```

The message elements have the following meaning:

Aside from **<header>** and **<update>** the elements are the same as for .

<update>

Each subelement of **<update>** sets a new value for the respective aspect of the job(s). All other aspects of the job remain unchanged. To unset **<periodic>**, include **<periodic>** **</periodic>**. It is permissible to change **<status>** with this command although this only make sense in a few cases.

Note that it is possible to change multiple jobs at the same time with an appropriate **<where>**, but there is only one **<update>** element that is applied to all selected jobs.

Example reply:

```
        </header>
                    </source>
      </target>
 0     swer1>
          </session_id>
```

The reply elements have the following meaning:

<answer1> 0 if the jobs were successfully deleted or a numeric error code if there was a problem. In the latter case there will be an **<error_string>** element with a human language description of the error.

If no job matches the query that is not an error.

gosa-si-server notes:

gosa-si-server disallows changing the timestamp on jobs that have status “processing”.

go-susi notes:

When go-susi receives an update request for a job that is another server’s responsibility, it will forward the request to that server. go-susi will not change its own job information immediately but will wait for the responsible server to react. This means that if the responsible server is down, will have no observable effect on go-susi’s database.

GOsa notes:

GOsa has a bug in its edit job page (if you select a job’s “Edit” icon on the ”Deployment status” page). If a job has **<periodic>** set and you uncheck the respective checkbox when editing the job, the **<periodic>** will not actually be cleared, because GOsa omits the **<periodic>** from the **<update>** element instead of setting it to “none”.

This message is sent in .

GOsa always uses the id column to identify the jobs. See the notes for regarding **<id>**.

GOsa does not care about the actual reply. The only requirement is that the outer-most tag must be **<xml>**. If there is an **<error_string>** element, GOsa will log an error.

Server - Server

There are various reasons for running multiple servers with GOsa and an accompanying si-server. To make administration easier, si-servers exchange information that allows different instances of GOsa to be used interchangeably. For example you can plan a job in one GOsa and cancel the job in another.

When an si-server starts up it looks into its configuration file and DNS for peer si-servers. It will then send a [illegible] message to each peer. Peers that are reachable will send [illegible] messages in return. After this exchange the servers have a common encryption key and know about each other's list of registered clients.

Whenever one server changes the information about a job in its database, e.g. because of a message like [illegible] or because a client performing an installation has sent a [illegible], the server informs all of its peers about the change with a [illegible] message, so that the peers can apply the same change to their databases.

Because a gosa-si-client only accepts messages from the server it registered at, only that server can properly execute jobs that require sending messages to that client. In particular this affects jobs that require the client to reboot or shut down. To solve this problem the server will forward such jobs to the peer where the client is registered.

To make forwarding work, each server must know at all times which clients are registered at which servers. Therefore, whenever a client registers at a server, that server will send a [illegible] message to all of its peers to inform them about the new client. There is no corresponding deregistration. A client is considered to belong to a specific server until it registers at another one.

A job is forwarded to a peer by sending that peer the appropriate [illegible] message.

A special case are wakeup jobs. These are not forwarded in the sense that the responsibility for the wakeup is passed on to the peer. Instead, peers may be asked to help waking up a client, because a peer may have information about the client's network location that the original server responsible for the job lacks. To recruit the help from a peer in waking up a client the [illegible] message is used.

Note:

While go-susi and gosa-si-server both follow the general principles outlined above for the forwarding of jobs, the timing and the message details are different for the two.

new_server

Purpose:

Server➔Server. A server announces its presence to another server so that the receiving server will inform the sending server about job updates etc.

The receiving server must react by sending a [illegible] message to the sender (via a new connection). The sender will keep using the old encryption key if it does not receive a [illegible] message with the new key or, if there is no old key, communication will fail altogether.

Example message (encrypted with [ServerPackages] key):

```
[illegible]
    [illegible]  [illegible]     </header>
    [illegible]        /new_server>
    [illegible]  [illegible]            </source>
    [illegible]  [illegible]            </target>
    [illegible] [illegible]  [illegible]          </key>
    [illegible]        es>[illegible]        </loaded_modules>
    [illegible]        es>[illegible]       </loaded_modules>
    [illegible]        es>[illegible]      </loaded_modules>
    [illegible]        es>[illegible]           </loaded_modules>
    [illegible]        es>[illegible]     </loaded_modules>
    [illegible]        es>[illegible]       </loaded_modules>
    [illegible]        es>[illegible]   </loaded_modules>
    [illegible]  [illegible]                     </client>
    [illegible]  [illegible]                    </client>
    [illegible]      [illegible]          </macaddress>
[illegible]
```

The message elements have the following meaning:

<new_server> Always empty.

<key>

A randomly generated string of letters and digits to be used for server-server communication between sender and receiver. Replaces the most recent key exchanged between the 2 servers via [illegible] /[illegible] [illegible]. The sender won't actually use the new key until it receives a [illegible] .

<loaded_modules> (*0 or more times*) Modules supported by the sending server.

A server that advertises "goSusi" in this list (as go-susi does) promises the following:

- All [illegible] [illegible] it sends will be synchronous (see **<sync>** description there)
- It will send out a [illegible] message whenever a job it is responsible for changes, even if that change is caused by a [illegible] . Note the restriction of this rule to jobs the "goSusi" server is responsible for (i.e. those that have it as **<siserver>**). Without this restriction there would be infinite series of update messages.
- When it establishes a connection with the receiving peer for the first time or re-establishes it after an interruption, it will send a [illegible] with **<sync>**[illegible] **</sync>**.
- All jobs have a unique **<id>** which is contained in [illegible] replies as well as [illegible] messages and can be used to address jobs with specificity in **<where>** clauses.
- When a "goSusi" server sends out a [illegible] message that affects a job whose **<siserver>** is another server, it will use that server's original **<id>** for the job. IOW, the **<id>** of a job is always from the jobdb of the job's **<siserver>**.
- An **<id>** value is never re-used for a different job once it has been used.

- When a **<periodic>** job has finished, the next repetition of the job gets a new **<id>**.
- When a **<periodic>** job has finished and the next repetition gets scheduled, both facts are communicated by [illegible] (either in two separate messages or one combined message).
- Two jobs with different **<id>** are never treated as the same. In particular [illegible] messages do not use the combination **<headertag>+<macaddress>** to identify jobs. This applies only to communication with other servers advertising "goSusi". When communicating with non-"goSusi" servers this behaviour is not required.
- A corollary of the previous point is that a server that advertises "goSusi" is capable of managing multiple jobs with identical properties.

<client>
Listening port and MAC address of a client registered at this server.
<macaddress> The sending server's MAC address.

gosa-si-server notes:
gosa-si-server sends this message in [illegible]
gosa-si-server re-registers at all known other servers (not just from DNS but also those known from incoming messages) in regular intervals. go-susi doesn't do that at this time.

go-susi notes:
When go-susi starts, it will send this message to all servers listed in DNS for the service tcp/gosa-si. When go-susi receives a [illegible] message it will send a [illegible] message to the server listed in <source> (usually the sender itself) and following [illegible] it will send its complete jobdb as a [illegible] message. The latter behaviour is different from gosa-si which only sends [illegible]
When go-susi is contacting a peer server for the first time after being started, go-susi will consider the key it sends in the new_server message valid and will use it even before the peer replies with [illegible]. IOW the [illegible] message is not required in the case of go-susi.

confirm_new_server

Purpose:
Server➔Server. Same format as [illegible], except that the **<header>** and the empty tag are [illegible]. When server A sends server B a [illegible] message containing server A's information, server B sends back (via a new connection) a [illegible] message containing server B's information. After this exchange both servers have each other's data. The [illegible] message usually contains the same **<key>** as the new_server message. In any case the most recent [illegible] *received* (not *sent*!) determines the key used for those messages that are encrypted with a server-key (such as [illegible]).

foreign_job_updates

Purpose:

Server➔Server. Inform another server about changes made to one or more jobs. This message is also used to cancel other servers' jobs by telling them the status is "done" (and periodic is "none"). The server that receives this message replaces the data of the job(s) with the new data.

Example message:

```
                          </header>
                        </source>
                      </target>
             </sync>

          e>          </plainname>
          >           </progress>
                    </status>
          >           </periodic>
          >                  </siserver>
          >1</modified>
          g>                         </targettag>
          ss>                          </macaddress>
          p>                     </timestamp>
      4   d>
          g>                              </headertag>
                    </result>
          ge>




                           </xmlmessage>
```

The message elements have the following meaning:

<sync> *(optional, unless the sender has "goSusi" in <loaded_modules>)*

[illegible]

The order in which [illegible] messages are received and the order in which jobs are listed within a message may not reflect the order in which the changes were performed.

[illegible]

The sender ensures that [illegible] messages are received and jobs within them are listed in the same order in which the changes were performed. This synchronization is implemented by sending all [illegible] messages properly ordered over a dedicated permanent connection.

[illegible]

The [illegible] message contains a complete list of all jobs the sender is responsible for (i.e. where **<siserver>** is the sender). The receiver is expected to discard all old information about the sender's jobs it has and replace it with the new data.

<answerX>

Each job in the list has its own number X, counting from 1 with no numbers left out. Note that X is

not identical to the **<id>** number.

<id>

If the sending server does *not* advertise "goSusi" in its message, this number is the ID of the job in the *sending* server's jobdb.
If the sending server *does* advertise "goSusi" in its message, this number is the ID of the job in the *responsible* server's jobdb (i.e. the jobdb of the job's **<siserver>**).

<siserver>

The listen address (IP:port) of the server responsible for processing the job. When gosa-si-server sends this message, this can also be the word "localhost" (and go-susi treats this as alias for the address from **<source>**).

<periodic>

If the was sent as a result of , then **<periodic>** is always "none" or not present, even if the job started out as periodic.

<modified> always 1.

<macaddress>+<headertag>

gosa-si-server uses this pair as the key to uniquely identify a job. This means that the job database of a gosa-si-server can only contain one job type per MAC address. If the jobdb already contains an entry for the **<macaddress>+<headertag>** pair from the message, that entry will be updated. Otherwise a new entry will be added to the jobdb.
gosa-si-server treats **<macaddress>** as case-sensitive and will not process correctly if the case in the message does not match the case in its database.
go-susi uses **<id>** to uniquely identify a job and permits multiple jobs with the same **<macaddress>+<headertag>** to coexist. When go-susi evaluates from a gosa-si-server, however, it will fall back to gosa-si-server's behaviour.

<xmlmessage>

Base64-encoded <xml>...</xml> message that was used to create the job originally.

gosa-si-server notes:

This message is handled in
<xmlmessage> may include whitespace characters in the base64-string that need to be removed before the string will decode properly.
gosa-si treats **<macaddress>** as case-sensitive and will not process correctly if the case in the message does not match the case in its database.

go-susi notes:

<sync> is a go-susi extension.
gosa-si-server sends **<xmlmessage>** to itself (replacing "job_" with "gosa_" in the <header>) when the job's time has come. This means that in case of an inconsistency between **<xmlmessage>** and the job's data, **<xmlmessage>** wins. go-susi does not use **<xmlmessage>**, so in case of an inconsistency go-susi will go by the job's data.

new_foreign_client

Purpose:

Server➔Server. When a client registers at a server using , that server notifies its peers of the new client by sending a message.

Example message:

```
</header>
</source>
</target>
</client>
</macaddress>
</key>
</key>
client></new_foreign_client>
```

The message elements have the following meaning:

The **<macaddress>** and **<client>** elements identify the client. The **<key>** elements, if present, specify the most recent and the previous encryption key sent by the client via or . The **<key>** elements are go-susi specific and not sent by gosa-si-server.

Note that even with the client's key a foreign server can not contact that client successfully, because gosa-si-client discards messages whose origin is not the si-server it is registered at.

trigger_wake

Purpose:

Server➔Server. Ask a peer server for help in waking up a machine.

Example message:

```
<trigger_wake>
    <header>
    <source>
    <target>
    <macaddress>
    ></trigger_wake>
</trigger_wake>
```

The message elements have the following meaning:

The **<macaddress>** identifies the machine to be woken up. The other elements have their usual meanings. The empty **<trigger_wake>** element is always empty.

Client - Server

When an si-client starts up, it checks its configuration as well as DNS and assembles a list of available si-servers. It will then send a [illegible] message to the first server on the list. If it receives a [illegible] message from the server within a certain amount of time (typically 10s), the client and that server are paired and have a shared encryption key that they will use for all future communications.
If no [illegible] message is received from the server within the time window, the client will send a [illegible] to the next server on the list and will keep on going through the server list until a server manages to answer within the time window. This behaviour can lead to the absurd situation that all servers on the list are available and do answer but registration fails anyway because delays cause all of the answers to miss the time window.
From time to time the client will send a [illegible] message to the server to establish a new encryption key to be used for future messages between the two parties. While this does not make the stupid homegrown encryption protocol used by gosa-si secure, it does cause race conditions because the client simply starts using the new key right away, even though the server may not have processed the [illegible], yet.
What happens after the server has sent the [illegible] message to the client depends on whether an LDAP object exists for the client's system or not. If there is an LDAP object, the server will read its data and data from [illegible] it is a member of and will send corresponding [illegible] and [illegible] messages to the client.
If there is no LDAP object for the client, the server sends a [illegible] message and creates an installation job with [illegible] for the new system. The client then performs hardware detection and sends the results back to the server in a [illegible] message. The server uses that information to create an LDAP object. If a matching template object exists (go-susi extension, see [illegible] for details), the result is a complete system object. Otherwise it is an incomplete object in [illegible] that needs to be completed in GOsa.
Once the object has been completed, the server sends the available information to the client in [illegible] and [illegible] messages. And when the [illegible] becomes [illegible], the server sends [illegible] It is only after the client has received both [illegible] and [illegible] that it can begin the actual installation process. A variety of message types may occur during an installation or softupdate. These are explained in their own chapter of this manual.
Whenever a server starts executing a job that affects a client, it will send that client a [illegible] message. Depending on the kind of job the client may react to this by e.g. presenting a popup to a logged in user asking the user to log out.
Jobs of type [illegible] have their own associated client-server messages but go-susi does not support these. See the chapter on deprecated messages later in this manual.

here_i_am

Purpose:

Client➔Server. Used by an si-client to register at the si-server responsible for it. The si-server reacts by sending a message possibly followed by

- a message to tell the client which LDAP parameters it should use.
- a message to instruct the client to perform hardware detection.
- a message to tell the client which NTP server(s) to use.
- a message to tell the client which syslog server to use.

A client to be (re)installed will not start the installation until it has received the return message(s) to its message from the server from which it is installing. In other situations (e.g. normal booting) the client will, if it does not receive the messages after a few seconds, try to register at each si-server listed in DNS in turn to find one that answers quickly enough.
The message is also responsible for setting the encryption key the si-server should use to encrypt messages sent to the client (such as
When a client has successfully registered at a server, the server sends out messages to its peers.
The new client will also appear in future ( ) messages sent by the server.
If the server can find the client in LDAP by its MAC address, it will also update the ipHostNumber field. go-susi also updates the cn.
If the sending client does not exist in LDAP, a reinstall job will automatically be created with **<progress>**hardware-detection**</progress>** and **<status>**processing**</status>** and the message will be sent to the client.

Example message:

```
          </header>
                    </source>
                    </target>
      here_i_am>
      </mac_address>
                              </new_passwd>
      >6  </key_lifetime>
      ion>0    </client_revision>
      s>h     </client_status>
      Checksum>                    g</gotoHardwareChecksum>
```

The message elements have the following meaning:

<mac_address>

The sending client's MAC address. **Note:** The tag has an underscore in its name unlike the MAC address elements in other messages.

<new_passwd>

The client equivalent to the **<key>** element of messages. The receiving server will use this key to encrypt messages to the client.

<key_lifetime>

Number of seconds the key is valid. The client will send a message in intervals of this many seconds (Some versions of gosa-si-client don't do this because of bugs. They keep using the same key indefinitely).

<client_revision>, <client_status> Version information about the client.

<gotoHardwareChecksum> A digest generated based on the machine hardware.

<events> The set of messages the client understands.

go-susi notes:

go-susi updates the system's field when it receives this message. If the system's does not match the long or short name of the reverse lookup of its IP, the object will be renamed to match the DNS name.

gosa-si-client notes:

gosa-si-client internally manages a list that it will contact in turn with until it receives a response within a certain reply time. If gosa-si-client receives a message that it cannot decode, it will re-register, however, it will do so at the next server in the list instead of starting at the beginning again.

new_key

Purpose:

Client➔Server. Informs the server about a new encryption key to be used when communicating with the sending client.

Example message:

```
<new_key>
    <header>
    <source>
    <target>
    <key>
</new_key>
```

registered

Purpose:

Server➔Client. Sent in reaction to a client's [illegible] message.

Example message:

```
[illegible]
    [illegible]    [illegible]    </header>
    [illegible]    [illegible]            </source>
    [illegible]    [illegible]            </target>
    [illegible]    le>[illegible]  </ldap_available>
    [illegible]    /registered>
[illegible]
```

The message elements have the following meaning:

<ldap_available>

If this is present and [illegible]e it tells the client that it will get a [illegible] message. If the element is absent or [illegible]e , this means that no LDAP information is available and the client will not receive a [illegible] message until its LDAP object has been created (usually after hardware detection).

LHM note:

If this element is sent as [illegible] and not followed up with a [illegible] [illegible] , the installation will not properly display the message "System is locked. Waiting for activation."

deregistered

Purpose:

Server➔Client. Sent to a client that has been registered at this server to inform it that the registration needs to be renewed if the client wants to stay registered at this server. This go-susi specific message is sent when the server receives a [illegible] message that informs it that one of its clients is now registered at a different server. When a client receives a [illegible] message from a server the client still believes is the one it should be registered at, it will re-register at that server. Otherwise the client will simply ignore the message. This corrects the effects of erroneous [illegible] messages which may result from race conditions during the registration process or network problems.

Example message:

```
[illegible]
    [illegible]    [illegible]      </header>
    [illegible]    [illegible]            </source>
    [illegible]    [illegible]            </target>
[illegible]
```

go-susi note:

This message is specific to post-TLS versions of go-susi. However both pre-TLS versions of go-susi and gosa-si-clients will react properly because they see the "STARTTLS" as undecryptable gibberish and will re-register based on the assumption that the decryption keys of the pre-TLS protocol have gone out of sync between client and server.

new_ldap_config

Purpose:

Server➔Client. Sends various LDAP-related information to the client. This message is not sent until the client object has been properly created.

Note:

When go-susi *receives* this message, it will call . If go-susi is in client-only mode, it will also update its internal LDAP parameters.

Example message:

```
                    </header>
                       </source>
                       </target>
                                              </admin_base>
                    </department>
                                             </ldap_base>
                                  </ldap_uri>
                                  </ldap_uri>
               </release>
                           </unit_tag>
          fig></new_ldap_config>
```

The message elements have the following meaning:

(optional)

The release the client should have according to LDAP if known.

(1 or more)

The LDAP server(s) from which the client should read its data. All listed LDAP servers contain the same data with respect to the client system's object but may differ in other ways. In general the client should use the first **<ldap_uri>** and only fall back to trying the others if the first is not available.

The base under which the client should look for its data. There is only one **<ldap_base>** element even if there are multiple **<ldap_uri>** elements.

(only present when unit tags are used) The unit tag for the client.

(only present when unit tags are used)

The dn of the admin base object. This is the first object under the **<ldap_base>** that matches Typically this is the same as the **<ldap_base>**.

(only present when unit tags are used) The attribute of the admin base object.

gosa-si-server notes:

contains the code that constructs this message. When the system is activated for installation, the message will always be followed by . This seems like a race condition. I think it would be better to send the beforehand. In practice this is not a problem with gosa-si-server because it seems to send the LDAP config once before and once after the however I believe that the data sent before reflects only a default choice and may not be current if the LDAP server is changed in GOsa for a new system.

GOsa/go-susi note:

The format of the [illegible] attribute that is the source for this message's information has changed over time. go-susi only supports the version 3 format.

Version 3 format: [illegible]
Version 2 format: [illegible]
Version 1 format: [illegible]

where

- [illegible] is a number used to sort multiple [illegible] values.
- [illegible] is unused in version 3 format.
- [illegible] is the ldap:// or ldaps:// URL of the server.
- [illegible] is the DN of the subtree to which searches should be restricted.

new_ntp_config

Purpose:

Server➔Client. Tells the client which NTP server(s) to use.

Note:

When go-susi *receives* this message, it will call [illegible] .

Example message:

```
[illegible]
    [illegible]  [illegible]          </header>
    [illegible]  [illegible]              </source>
    [illegible]  [illegible]              </target>
    [illegible]  [illegible]          </server>
    [illegible]  [illegible]            </server>
    [illegible]          ig></new_ntp_config>
[illegible]
```

The message elements have the following meaning:

<server> *(1 or more)* NTP server(s) to use.

detect_hardware

Purpose:

Server➔Client. When the server receives a [illegible] message with a MAC address for which no [illegible] object can be found in LDAP, the server sends a [illegible] message. The client reacts to this message by performing hardware detection and sends a [illegible] message to the server. The server then creates the LDAP object with that data.

Example message:

```
[illegible]
    [illegible]  [illegible]          </header>
    [illegible]  [illegible]             </source>
    [illegible]  [illegible]             </target>
    [illegible]       are></detect_hardware>
[illegible]
```

detected_hardware

Purpose:

Client➔Server. When the server receives a [illegible] message with a MAC address for which no GOhard object exists in LDAP , the server sends a [illegible] message to the client. The client reacts to this message by performing hardware detection and sends a [illegible] message to the server. The server then creates the LDAP object with that data. If a [illegible] message is received for a system whose MAC address is found in LDAP, then that system's LDAP object will be updated with the new info.

go-susi extension:

go-susi extends the usage of [illegible] to permit any kind of creation, modification and even moving of system objects within the LDAP tree, even without prior use of [illegible] . See the go-susi note further below for more information.

Template objects:

The data from the [illegible] message is not enough to create a complete system object that is ready for installation. This means that usually further action is necessary in GOsa to provide the remaining data before the installation can begin. To allow for fully automatic installation, go-susi has the ability to complement the [illegible] data with data copied from a template object.

Whenever go-susi receives a [illegible] message for a system that does not yet exist in LDAP, go-susi will search for template objects which are objects that have [illegible] and a [illegible] attribute that starts with "[illegible] ". The remainder of the [illegible] is taken as a matching rule as described below. If the rule matches against the attributes of the new system that are known so far, then the other attributes are copied from the template object.

Template objects that are group members:

If the template object is [illegible] of any object groups ([illegible]), then the new system will be added to the same groups.

Auto-activation for immediate installation:

If the template object has [illegible] , then the new system will be activated and installation will commence immediately.

Object creation not in new-systems-base:

The new system's LDAP object will be placed in the same LDAP path as the template object from which it copies attributes. Template objects are allowed to be but don't need to be in [illegible]

Matching rule language:

A matching rule consists of one or more parts, optionally separated by whitespace. Each part can have one of the following 2 forms:

[illegible]

matches if the system has at least an attribute named [illegible] with at least one value that matches the given regex. This is a non-anchored match. If you want the match to be anchored you need to specifically include ^ and/or $ in the regex.

As a special case the regex [illegible] matches if the system has no attribute [illegible] .

[illegible]

matches if none of the system's values for the attribute named [illegible] match the given regex. If the system has no attribute of that name this will be treated as if it had such an attribute with an empty string as value.

The regex syntax is described here: https://code.google.com/p/re2/wiki/Syntax

Whitespace note: Every sequence of whitespace characters in regexes is converted to "[illegible] ", so a simple space represents any kind of whitespace. Note that this breaks some things like "[illegible] ". Use [illegible] if you need to explicitly match a space.

Matching process:

The matching rule is divided into groups of subsequent parts of the same type (i.e. ≃ or ∔).
From each ≃ group the system must match at least one part.
From each ∔ group the system must match all parts (i.e. it must not match any of the regexes).

Example:

matches
matches
does not match (the " " from the 2nd ≃ group is missing)
does not match (forbidden by)
does not match (forbidden by)
does not match (needs " " or " " from the 1st ≃ group)
matches (multiple matches from each group are okay)

Matching rule precedence:

If multiple template objects have matching rules that match the new system, go-susi will choose that template object whose rule contains the greatest number of matching (or non-matching in the case of "!~") regular expressions. If multiple template objects have the same score, one of them will be picked in an unspecified manner.

Example: In the example from **Matching process** above, has a matching score of 5, because it matches all three ≃ parts and does not match the two ∔ parts.

Attributes available for matching:

The following attributes may be used in matching rules:

- all attributes from the **<detected_hardware>** element of the message.
- the system's . If the **<detected_hardware>** element does not contain a valid MAC, it will be determined from the most recent or message concerning the client matching 's **<source>** element.
- the system's extracted from 's **<source>** element (unless there is an attribute or sub-element in **<detected_hardware>**).
- which is the system's DNS name in plain or fully qualified form as determined by a reverse lookup of (see above).
- which evaluates to the go-susi server doing the matching. You can match by IP address as well as DNS name (is a multi-value attribute)

Example message:

```
                    </header>
                     </source>
                     </target>
          dware
            ="GenuineIntel / Intel(R) Celeron(R) CPU E3300 @
             2493.734"
            ter="VMWare VMWARE0405"
            ="1025692"
            "AMD PCnet - Fast 79C971"
            apter="Ensoniq Creative Sound Blaster AudioPCI64V,
            28"
            ort="false"
            areChecksum="qQvNNtPqaV5MG82UElNS7g"
            dule="snd_ens1371"
            er=""
```

```
                c="31-48" gotoXMonitor="Generic Monitor"
                eType="explorerps/2" gotoXMouseport="/dev/input/mice"
                lution="800x600" gotoXVsync="50-90"
                del="pc104" macAddress="00:00:00:00:00:00"
        >
                v>                          </ghScsiDev>
                v>                 </ghScsiDev>
                les>     </gotoModules>
                les>        </gotoModules>
                les>       </gotoModules>
                les>      </gotoModules>
                les>          </gotoModules>
                ...
                les>         </gotoModules>
                les>   </gotoModules>
                les>       </gotoModules>
        >              rdware>
```

gosa-si-client note:

The **macAddress** attribute of **<detected_hardware>** may be 00:00:00:00:00:00 when this message is sent by gosa-si-client. I don't know if this happens all the time with all versions of gosa-si-client.

go-susi note:

go-susi permits multiple **<detected_hardware>** elements and will take the union of all their values. As an alternative to attributes on the **<detected_hardware>** element go-susi also accepts child elements inside **<detected_hardware>**, e.g. **<detected_hardware><gotoXHsync>**31-48**</gotoXHsync></detected_hardware>**.

go-susi writes all attributes and sub-elements of **<detected_hardware>** into the LDAP object, except . If there is an attribute or element, it will override the IP address extracted from **<source>**. If both and are present, the **<source>** element does not need to identify a known client. This means that with go-susi you can use a message to create a self-contained ready-for-installation object in LDAP and you can use messages to update all aspects of a system object. It is even possible to rename a system by including a attribute and to move a system by including a attribute.

Note that when updating an existing object, all attributes from the old object that are completely missing from **<detected_hardware>** will be copied, but if **<detected_hardware>** has an attribute with the same name, it will replace the old attribute completely. IOW, old and new attribute values are never mixed for the same attribute. E.g. if you want to add a single entry, you need to repeat all the old entries in the message. You can't just list the new module and expect the others to be copied.

To delete an attribute, include an empty sub-element of that name in **<detected_hardware>**.

go-susi accepts messages encrypted with any key, not just client keys. This allows a 3rd party to create system objects by sending encrypted e.g. with the [GOsaPackages] key.

trigger_action_*

Purpose:

Server➔Client. Sent by the server to the client when the server starts executing a job that affects the client. The client will then react appropriately. Most of the time this means presenting a message to logged in users to log out.

Example message:

```
                                    </header>
                                </source>
                                </target>
                 on_localboot></trigger_action_localboot>
                  </session_id>
```

The following actions are supported:

Installation and Softupdate

When an installation is pending and the client's LDAP object has the client waits for the si-server where it is registered to send the client a message. During an installation as well as during a softupdate the client sends information about the FAI progress to the server.
The message contains a number between 1 and 100 that gives a rough indication of the percentage of the process that has been completed.
The message is a strange kind of progress message that sets the job's <progress> (which otherwise is a number) to the string
The and messages inform the server about specific steps in the FAI process.
The most important message during installation/update is which the client uses to transmit the log files from the process to the server. The server stores these log files and allows access to them via and
The has an unclear purpose and is ignored by the si-server.

set_activated_for_installation

Purpose:

Server➔Client. Tell the client that it has been activated. This message is always sent when a job is executed, regardless of whether the client has a pending installation or not. It’s even sent if the client is already activated.

Example message:

```
                              </header>
                   </source>
                   </target>
            </faistate>
         d_for_installation></set_activated_for_installation>
             </session_id>
```

The message elements have the following meaning:

<faistate>

This reflects the value of the target machine's attribute at the time of the activation job that triggered the sending of the message.

gosa-si-client notes:

When the client receives this message in it creates a file . waits for the existence of this file before starting the installation.

CLMSG_PROGRESS

Purpose:

Client→Server. During installation or softupdate the client uses this message to send progress information in percent to the server.
When the server receives this message it will update its job data and will forward the new information via [illegible] to its peers. Subsequent [illegible] requests will also receive the new progress number in the answer's **<progress>** element.
Note that [illegible] is technically also a progress indication and reflected in the **<progress>** element.

Example message:

```
[illegible]
    [illegible]  [illegible]          </header>
    [illegible]  [illegible]              </source>
    [illegible]  [illegible]            </target>
    [illegible]        SS>7 </CLMSG_PROGRESS>
    [illegible]          [illegible]              </macaddress>
    [illegible]        [illegible]           </timestamp>
[illegible]
```

The message elements have the following meaning:

[illegible]

An integer between between 1 and 100 (inclusive) giving the percentage of the installation/softupdate that has been completed.

[illegible] see [illegible]

CLMSG_GOTOACTIVATION

Purpose:

Client➔Server. The client sends this message when it receives via its FIFO interface. I don't know who sends this to the FIFO. In any case it happens when the client is waiting for activation because it is in It does not matter if the client already existed in LDAP or was just created.

When the server receives this message it will update its job data to have

<progress> **</progress>**

and will forward the new information via to its peers. Subsequent requests will also receive the new progress **<progress>** element.

Other than updating the progress, this message has no effect on the server.

Example message:

```
                    </header>
                  </source>
                 </target>
       TIVATION></CLMSG_GOTOACTIVATION>
                     </macaddress>
                 </timestamp>
```

The message elements have the following meaning:

see

CLMSG_save_fai_log

Purpose:

Client➔Server. Use case 1: Transmits log files from a finished installation or softupdate to the server.
Use case 2: Transmits audit data to the server. See [illegible]

Example message:

```
[illegible]
    [illegible]    [illegible]                </header>
    [illegible]    [illegible]                </source>
    [illegible]    [illegible]               </target>
    [illegible]        [illegible]                </macaddress>
    [illegible]        [illegible]      </fai_action>
    [illegible]         ai_log>
    [illegible]
    [illegible]
    .
    [illegible]         fai_log>
[illegible]
```

The message elements have the following meaning:

[illegible]

The MAC of the client whose log files are being sent (usually the same machine as identified by **<source>**.

[illegible]

The log files to be transmitted. Every log file starts with the literal string [illegible], followed by the file name, followed by [illegible]. This is followed by the base64-encoded file contents and terminated by a space.

[illegible]

The FAI action for which log files are being transmitted. Possible values are [illegible], [illegible] and [illegible] [illegible]

There is no **<timestamp>**. Unlike most other [illegible] messages, this one does not include a **<timestamp>** element.

gosa-si-client notes:

gosa-si-client inserts spaces into the base64-encoded data which break some base64 decoders. Remove all whitespace before decoding.
I have observed broken base64 strings coming from gosa-si-client.

CLMSG_<FAI MONITOR EVENT>

Purpose:

Client➔Server. During installation gosa-si client receives events from FAI via its FIFO interface. These events can be seen in the log file. gosa-si-client passes these events on to the server.

The CLMSG_* messages are used to communicate the status of an installation from the client to the Server. These messages look almost the same, except the element with the value and the header. Furthermore, the message has the additional element

Example message:

```
                                    </header>
                                       </source>
                                     </target>
                GIN>                </CLMSG_TASKBEGIN>
                                         </macaddress>
                                     </timestamp>
```

Possible <FAI MONITOR EVENT>s:

CLMSG_TASKBEGIN / CLMSG_TASKERROR / CLMSG_TASKEND

Purpose:

(See also [illegible] [illegible])

Client➔Server. During installation or softupdate the client sends these messages to the server to inform it about FAI tasks that are being started ([illegible] [illegible]), errors that occur during task execution ([illegible]) and the exit status when a task finishes ([illegible]).

Example messages for a successfully completed task:

Task “confdir” begins:

```
                    </header>
                       </source>
                      </target>
          GIN>         </CLMSG_TASKBEGIN>
                           </macaddress>
                        </timestamp>
```

Task “confdir” has terminated successfully with exit code 0:

```
                  </header>
                       </source>
                      </target>
          D>          </CLMSG_TASKEND>
                           </macaddress>
                        </timestamp>
```

Example messages for a failed task:
Task “instsoft” begins:

```
</header>
</source>
</target>
GIN>6 </CLMSG_TASKBEGIN>
</macaddress>
</timestamp>
```

An error code 472 occurs during execution of task “instsoft”:

```
</header>
</source>
</target>
GIN>e </CLMSG_TASKBEGIN>
</macaddress>
</timestamp>

</header>
</source>
</target>
ROR>

RROR>
</macaddress>
</timestamp>

</header>
</source>
</target>
D># </CLMSG_TASKEND>
</macaddress>
</timestamp>
```

The CLMSG_TASKEND for the “instsoft” task that failed:

```
</header>
</source>
</target>
D># </CLMSG_TASKEND>
</macaddress>
</timestamp>
```

The message elements have the following meaning:

TASKEND>

These messages always come in pairs. Every has a matching .

_TASKERROR>,<CLMSG_TASKEND>

A is surrounded by and for an task. This means that if an error occurs during a task there will be (at least) 5 messages: . Other messages, in particular and may occur between these messages.

The following is (an incomplete) list of possible errors:

see

CLMSG_TASKDIE

Purpose:

(See also [illegible])

Client➔Server. Signals a fatal error that causes FAI to abort. The system to be installed is probably unusable. Log files are transmitted using [illegible] despite [illegible] .

Example message:

```
[illegible]
    [illegible]  [illegible]          </header>
    [illegible]  [illegible]               </source>
    [illegible]  [illegible]              </target>
    [illegible]       E>[illegible]                                  </CLMSG_TASKDIE>
    [illegible]      [illegible]              </macaddress>
    [illegible]      [illegible]  [illegible]      </timestamp>
[illegible]
```

go-susi notes:

When go-susi receives this message, it removes the running job and sets the system's [illegible] to [illegible] .

CLMSG_check

Purpose:

(See also)

Client➔Server.Sent by gosa-si-client when it receives "check <foo>" via its FIFO. This is caused by

from FAI's . Has no effect on the server.

Example message:

```
                                    </header>
                                      </source>
                                    </target>
                                        </macaddress>
                                  </CLMSG_check>
                                    </timestamp>
```

The message elements have the following meaning:

The plain name of the client.

see

Query various information

The si-server manages various databases on behalf of GOsa.
The message [illegible] returns a list of all known Debian software repositories as well as the available releases and their sections.
The message [illegible] performs a parameterized search over the database of FAI classes. It allows selection by a variety of criteria.
The message [illegible] performs a parameterized search over the database of Debian packages. It allows selection by a variety of criteria.
The message [illegible] returns a list of all available kernels for a certain release.
The messages [illegible] and [illegible] are used to access the log files from installations/updates stored on the server. Note that server-server-communication does not extend to log files, so these messages only return those log files sent to the particular server via [illegible]
The messages [illegible] and [illegible] are used to access audit data stored on the server. Note that server-server-communication does not extend to audit files, so these messages only return those log files sent to the particular server via [illegible]

Note:

The transition of GOsa from accessing LDAP directly to using the si-server is very incomplete. In many places GOsa does its own LDAP reading and writing and maintains its own state. Whenever an operation in GOsa takes very long, that is usually the reason.

gosa_query_fai_server

Purpose:

GOsa↔Server. Return a list of all known Debian software repositories as well as the available releases and their sections.

Example message:

```
[illegible]
    [illegible]                              </header>
    [illegible]       </source>
    [illegible]       </target>
[illegible]
```

Example reply:

```
[illegible]
    [illegible]                       </header>
    [illegible]                          </source>
    [illegible]       </target>
    [illegible]
        [illegible]    p>[illegible]               </timestamp>
        [illegible]    ase>[illegible]           </fai_release>
        [illegible]    >[illegible]            </repopath>
        [illegible]  5 [illegible]                </tag>
        [illegible]                                    </server>
        [illegible]    >[illegible]                         f </sections>
    [illegible]
        .
    [illegible]    . </answerN>
    [illegible]          [illegible]   </session_id>
[illegible]
```

The reply elements have the following meaning:

<server> The repository URL of the directory that contains the directories [illegible] and [illegible].

<repopath>

The path relative to the d[illegible] directory of the directory that contains the [illegible] file.
If the same server has multiple repository paths, each one has its own **<answerX>** element.

<fai_release>

The FAI release to which the repository path belongs. Typically there is one repository path with the same name as the FAI release and additional repository paths that can optionally be used via additional sources.list lines (typical examples are [illegible] and [illegible]

<timestamp> The time when go-susi last checked this entry.

<tag> *(optional)*

The gosaUnitTag of the repository server (if it has one). The reply may contain servers with different unit tags.

gosa_query_fai_release

Purpose:

GOsa↔Server. Query the FAI classes database.

Example message:

```
</header>
t>
e>
i_release>             </fai_release>
```

The message elements have the following meaning:

<where>

The query syntax is the same as for . The available column names are **<timestamp>**, **<fai_release>**, **<type>**, **<class>, <tag>** and **<state>**. Their meanings are described below in the explanation of the reply elements.

Note: <tag> is a go-susi extension. See the explanation of the corresponding reply element below.

Example reply:

```
</header>
</source>
t>
</timestamp>
</fai_release>
</type>
</class>
e>
</timestamp>
</fai_release>
</type>
</class>
tag>
e>
.
N>
session_id>
```

The reply elements have the following meaning:

<timestamp>

The time (local server time) when go-susi last checked this entry.

<fai_release>

The release the answer belongs to, i.e. the release this class is available in. Only one release is listed per answer. If the query requested information for multiple releases that offer the same FAI class, each is listed in its own answer.

<type>

The type of the class. Possible values are

and .

Only one type is listed per answer element, even if the query returns multiple types with the same class name.

<class>

The name of the FAI class described in the answer.

<tag> *(optional)*

The gosaUnitTag of the FAI class (if it has one). The reply may contain classes with different unit tags.

Note: This is a go-susi extension. gosa-si-server does not include **<tag>** in its reply and does not support filtering by tag in the **<phrase>** element. gosa-si-server always reports all FAI classes regardless of their gosaUnitTag.

<state> The following values are possible:

empty string: This class has no special properties.

modifications to this class via GOsa are not permitted.

deprecated. go-susi never reports this state.

go-susi note:

In addition to FAI classes actually present in LDAP, go-susi reports pseudo-classes for all attributes found in the output from Refer to the section describing that hook for more information.

GOsa note:

GOsa 2.7 presents all types of FAI class with the same name in one integrated entry. If at least one type has FAIstate , GOsa will present a lock icon with the entry. However the individual parts remain independent and if e.g. a of name FOO has state freeze but a of name FOO doesn't, then GOsa permits editing of the , even though it prevents editing of the hook.

gosa_query_packages_list

Purpose:

GOsa↔Server. Query the Debian packages database.

Example message:

```
</header>
t>
e>
</select>
ect>
ect>
ect>
</select>
</select>
elect>
stribution>       </distribution>
r> </connector>
erator>    </operator>
ckage>                                                    </package>
erator>    </operator>
ckage>                                          </package>
```

The message elements have the following meaning:

<where> *(exactly 1)*

The query syntax is the same as for . The available column names are **<distribution>**, **<package>**, **<version>**, **<section>, <description>**, **<timestamp>** and **<template>**. Their meanings are described below in the explanation of the reply elements.

<select> *(0 or more)*

Each **<select>** element contains the name of a column that should be returned for each answer matching the query. If there is no **<select>** element, all columns are to be returned.

Note:

At this time go-susi ignores **<select>** and will always return all columns.

Example reply:

```
                                </header>
                              </source>
                  t>

                                  </timestamp>
                              </distribution>
                                                </package>
                    rsion>
                              </section>
                                              </description>
                                        </template>


                                  </timestamp>
                              </distribution>
                                        </package>
                    rsion>
                              </section>
                                            </description>
                    plate>

     .
          .         N>
                    ession_id>
```

The reply elements have the following meaning:

<timestamp>The time (local server time) when go-susi last updated the packages database.

<distribution>

The release the answer belongs to, i.e. the release the package is available in. Only one release is listed per answer. If the query requested information for multiple releases that include the same package, each is listed in its own answer.

<package> The name of the package described by the **<answerX>** element.

<version>

The version of the package. A package may exist in multiple versions even within the same distribution. In that case each version will have its own **<answerX>** element.

<section>Guess what.

<description>

The base64 encoding of the short description of the package, i.e. the text from the Debian file that starts after and extends to the end of the line. Neither nor the newline at the end of the line are included in the **<description>** element.

<template>

If the Debian package has a file describing debconf-parameters, the **<template>** element contains this file in its entirety encoded in base64.

gosa-si-server note:

As always gosa-si-server inserts spurious whitespace into the base64 encodings.

gosa_get_available_kernel

Purpose:

GOsa↔Server. Return a list of all available kernels for a certain release.
go-susi generates this list from the output of an external program specified by the configuration option [illegible] The default is [illegible]

Example message:

```
[illegible]
      [illegible]      [illegible]                          </header>
      [illegible]      [illegible]   </source>
      [illegible]      [illegible]   </target>
      [illegible]           [illegible]            </fai_release>
[illegible]
```

Example reply:

```
[illegible]
      [illegible]      [illegible]                    </header>
      [illegible]      [illegible]                   </source>
      [illegible]      [illegible]   </target>
      [illegible]           [illegible]  </session_id>
      [illegible]            e_kernel></get_available_kernel>
      [illegible]       [illegible]  [illegible]                     </answer1>
      [illegible]       [illegible]  [illegible]                    </answer2>
      .
      [illegible]       [illegible]  [illegible]  </answerN>
[illegible]
```

gosa_show_log_by_mac

Purpose:

GOsa↔Server. Returns all subdirectories of log files within the log file directory of the machine selected by the **<mac>** element. The MAC address is case-insensitive.

Example message:

```
</header>
</mac>
```

Example reply:

```
</header>
</source>
t>
_3f>                                  </mac_00_16_36_7c_db_3f>
_3f>                                     </mac_00_16_36_7c_db_3f>
.
show_log_by_mac>
 sion_id>
```

The reply elements have the following meaning:

<mac_...>

This element corresponds to the **<mac>** element in the request, but is always lowercase, even if the MAC address in **<mac>** was not.

For each subdirectory in the system's log directory there is one element.

If no log files are available, the answer will not contain any **<mac...>** elements.

gosa_show_log_files_by_date_and_mac

Purpose:

GOsa↔Server. Get the list of log files contained in a specific subdirectory of a machine's log file directory.

Example message:

```
</header>
t>
e>
</date>
</mac>
```

The message elements have the following meaning:

<mac> The MAC address (case-insensitive) of the machine for which to list log files.
<date> The subdirectory name within the machine's log file directory.

Example reply:

```
</header>
</source>
t>
date_and_mac>
_date_and_mac>
date_and_mac>
_date_and_mac>
ession_id>
date_and_mac></show_log_files_by_date_and_mac>
```

The reply elements have the following meaning:

<show_log_files_by_date_and_mac>

Each of these elements contains the name of one of the log files present in the requested subdirectory.

Note:

One of the returned **<show_log_files_by_date_and_mac>** is always empty.

gosa_get_log_file_by_date_and_mac

Purpose:

GOsa↔Server. Get the contents of a specific log file.

Example message:

```
                                                    </header>
                          t>
                          e>
                                        </date>
                             </mac>
                           og_file>
```

The message elements have the following meaning:

<mac> The MAC address (case-insensitive) of the machine for which to read a log file.

<date> The subdirectory name within the machine's log file directory where the log file is found.

<log_file> The name of the log file to return.

Example reply:

```
                                               </header>
                                     </source>
                          t>
                                    </foo.log>
                          te_and_mac></get_log_file_by_date_and_mac>
                           sion_id>
```

The reply elements have the following meaning:

<foo.log>

The name of the element corresponds to the contents of the **<log_file>** element in the request. The contents are the base64-encoded file contents.

GOsa note:

When the tab "installation logs" for a specific machine is accessed for the first time, no log file is selected and GOsa creates an incorrect request where **<log_file>** has the value "0". gosa-si-server answers this invalid request with a likewise broken reply.

gosa_query_audit

Purpose:

GOsa↔Server. Query the audit logs (see fai-audit-hook).

Example message:

```
                          </header>
                  dit>
                         </tstart>
                       </tend>
                  cludeothers>
                     ect>
                  ect>
                  ect>
                    </select>
                   </select>
                  elect>
                   </select>


                  y>        </key>




```

The message elements have the following meaning:

<audit> *(exactly 1)*

This name with an appended is the file name of the audit files to be scanned.

<tstart> *(optional)*

Timestamp of the beginning (inclusive) of the audit period to be scanned. Note that this does not refer to the modification/creation times in the filesystem but the names of the directories created when receives the audit files. Underscores _ within the timestamp are ignored. Default if omitted is "00000101000000".

<tend> *(optional)*

Timestamp of the end (inclusive) of the audit period to be scanned. See **<tstart>.** Default if omitted is "99991231235959".

<includeothers> *(optional)*

The presence of this tag causes the reply to contain **<nonmatching>** elements with information about systems that contain audit data within the requested time period that does match the **<where>** query.

In addition to this the reply will also contain **<noaudit>** elements with information about systems that do not have audit data in the requested time period.

<where> *(optional)*

The query syntax is the same as for . The available column names are **<macaddress>**, **<key>** and all custom child elements used in **<entry>** elements in the audit files selected by **<audit>.** If **<where>** is omitted, all entries are considered matching.

<select> *(0 or more)*

Each **<select>** element contains the name of a column that should be returned for each answer

matching the query. If there is no **<select>** element, it is unspecified which (if any) columns will be returned. You may request any of the custom columns as well as "macaddress", "ipaddress", "hostname" and "lastaudit". Note however that among these only "macaddress" and "lastaudit" are always available.

Note: Duplicate answers are **not** coalesced. For instance if you select "macaddress" and no other columns, you will receive as many answers with the same <macaddress> as there are <entry> elements in the audit data for the system with that MAC address. If you need coalescing of duplicate answers, may be more appropriate.

Example reply:

```
                      </header>
                                 </source>
                      t>
            2
              3       n>

                                        </macaddress>
                        3            </ipaddress>

    .

                                        </macaddress>
                        2                 </lastaudit>

    .

                        </lastaudit>
                                      </macaddress>
                      2         3   </ipaddress>
                                    </version>
               i      tus>
                      e>


    .

    .
           .          N>

```

The reply elements have the following meaning:

<known>

Number of systems that with <audit> information within the requested timestamp range. This includes those that do not have an <entry> that matches the <where> condition. However <audit> files that don't contain any <entry> at all do not count.

<unknown>

Number of systems that do not have <audit> information within the requested timestamp range. This includes systems that have sent corrupt or empty <audit> files.

<noaudit>

These elements contain (usually incomplete) information about systems that do not have the requested <audit> file within the requested timestamp range. May contain any of the subelements

<lastaudit>, <macaddress>, <ipaddress> and/or <hostname> but usually only <macaddress> will be present.

<nonmatching>

These elements contain (sometimes incomplete) information about systems that do have the requested <audit> file within the requested timestamp range, but that have no <entry> that matches the <where> condition. May contain any of the subelements <macaddress>, <ipaddress>,<lastaudit> and/or <hostname>.

<lastaudit>

Timestamp of the most recent audit (if any) of the system, regardless of <tstart>, <tend>, <audit>.

gosa_query_audit_aggregate

Purpose:

GOsa↔Server. Query the audit logs (see fai-audit-hook) with aggregate functions applied.

Example message:

```
                                          </header>
                    dit>
                           </tstart>
                         </tend>
                    cludeothers>
                        </select>

                           </as>
                          </unique>


                   rase>
                        <key>     </key>
                   hrase>




```

The message elements have the following meaning:

<audit> *(exactly 1)*

This name with an appended is the file name of the audit files to be scanned.

<tstart> *(optional)*

Timestamp of the beginning (inclusive) of the audit period to be scanned. Note that this does not refer to the modification/creation times in the filesystem but the names of the directories created when receives the audit files. Underscores _ within the timestamp are ignored. Default if omitted is "00000101000000".

<tend> *(optional)*

Timestamp of the end (inclusive) of the audit period to be scanned. See **<tstart>.** Default if omitted is "99991231235959".

<includeothers> *(optional)*

The presence of this tag causes the reply to contain **<noaudit>** elements with information about systems that do not have audit data in the requested time period. There is **no <nonmatching>** in the reply from as opposed to

<select> *(0 or more)*

Each **<select>** element contains the name of a column that should be returned for each answer matching the query. If there is no **<select>** element, there will be no **<answerX>** in the the reply, but **<aggregate>** will still be there.

Note: Unlike this function coalesces results, so there will not be any 2 **<answerX>** in the reply that have the exact same set of values for the whole set of **<select>** columns.

<count> *(0 or more)*

Report the number of matching entries.

<as> *(required within **<count>**)*

The name of the subelements in **<aggregate>** and **<answerX>** under which to report the count.

<unique> *(0 or more, only permitted within* ***<count>****)*

For each combination of **<select>** columns (each of which yields one **<answerX>**) only increment the count once for each unique value of the column specified inside **<unique>.**

If multiple **<unique>** are specified, the combination of all these columns has to be unique (which is not the same as all of them being unique individually).

The same applies to the **<aggregate>** result.

<where> *(optional, only permitted within* ***<count>****)*

The query syntax is the same as for . The available column names are **<macaddress>**, **<key>** and all custom child elements used in **<entry>** elements in the audit files selected by **<audit>.** Only matching entries will be counted. If **<where>** is omitted, all entries are counted (taking **<unique>** into account if present).

Example reply:

```
</header>
</source>
t>
n>
</macaddress>
</lastaudit>
.
4</bashversions>
</macaddress>
0</bashversions>
.
.
N>
```

The reply elements have the following meaning:

<known>

Number of systems that with <audit> information within the requested timestamp range. This includes those that do not have an <entry> that matches the <where> condition. However <audit> files that don't contain any <entry> at all do not count.

<unknown>

Number of systems that do not have <audit> information within the requested timestamp range. This includes systems that have sent corrupt or empty <audit> files.

<noaudit>

These elements contain (usually incomplete) information about systems that do not have the requested <audit> file within the requested timestamp range. May contain any of the subelements <lastaudit>, <macaddress>, <ipaddress> and/or <hostname> but usually only <macaddress> will be

present.

<lastaudit>

Timestamp of the most recent audit (if any) of the system, regardless of <tstart>, <tend>, <audit>.

<aggregate>

The numbers requested by **<count>** aggregated over all entries, regardless of the **<select>** set.

<answerX>

The numbers requested by **<count>,** counted for a unique combination of **<select>** values. E.g. if the query has **<select>**macaddress**</select>** and no other **<select>,** then each **<answerX>** will correspond to exactly one MAC address. If there is a **<select>** for both "macaddress" and "key", then there will be one **<answerX>** for each unique macaddress+key combination.

Miscellaneous

The messages in this section do not fit nicely in any of the other categories.

gosa_ping

Purpose:

GOsa↔Server. GOsa uses this message to determine if a specific client is on or off. GOsa uses this information e.g. to present the action "Wake up" only for clients that are off.

Example message:

```
                        </header>
               </source>
                              </target>
```

Example reply if client is ON:

```
                           </header>
                ></got_new_ping>
                              </source>
                           </target>
```

Reply if client is OFF:

If the client is off, the si-server closes the connection without reply.

GOsa notes:

GOsa only checks if there is a reply but ignores the contents. This means even an error reply will be interpreted as the client being on.

go-susi note:

go-susi does not actually ping the client and wait for a pong. go-susi only checks if it can connect to the si-client port. This eliminates the need for server-server communication in case the client is registered at a peer and improves GOsa performance compared to the use of gosa-si-server.

panic

Purpose:

Admin➔Server. Causes go-susi to abort with a full stack trace of all running goroutines. This function is go-susi specific and useful only for debugging.

Example message:

```
                </header>
```

sistats

Purpose:

Admin➔Server. Query various statistics about go-susi's operation. This function is go-susi specific.

Example message:


```
</header>
```


Example reply:


```
</header>
</source>
</target>
c> </Alloc>
itecture> </Architecture>
HashSys>0</BuckHashSys>
iler>g </Compiler>
gGC> </DebugGC>
leGC> </EnableGC>
s>3 </Frees>
ersion> </Go-Version>
Alloc> </HeapAlloc>
Idle> </HeapIdle>
Inuse> </HeapInuse>
Objects> </HeapObjects>
Released>0</HeapReleased>
Sys> </HeapSys>
GC> </LastGC>
ups> </Lookups>
heInuse> </MCacheInuse>
heSys> </MCacheSys>
nInuse> </MSpanInuse>
nSys> </MSpanSys>
ocs>2 </Mallocs>
GC> </NextGC>
PU>2</NumCPU>
C>2</NumGC>
oroutine>2 </NumGoroutine>
</OS>
eTotalNs> </PauseTotalNs>
sion> </Revision>
kInuse> </StackInuse>
kSys> </StackSys>
</Sys>
lAlloc> </TotalAlloc>
ion> </Version>
info_arena> </mallinfo_arena>
info_fordblks> </mallinfo_fordblks>
```

```
            info_hblkhd>0</mallinfo_hblkhd>
            info_hblks>0</mallinfo_hblks>
            info_keepcost>       </mallinfo_keepcost>
            info_ordblks>  </mallinfo_ordblks>
            info_uordblks>     </mallinfo_uordblks>
```

The reply elements have the following meaning:

The reply elements are subject to change. They are purely for testing and debugging. Most of them are either self-explanatory or correspond directly to values from the Go runtime (http://golang.org/pkg/runtime/).

<Version>

The go-susi version.

<SusiPeersUp>/<SusiPeersDown>

Number of working (Up) and non-working (Down) peers known to support the go-susi protocol.

<NonSusiPeersUp>/<NonSusiPeersDown>

Number of working (Up) and non-working (Down) peers that do not use the go-susi protocol.

<KnownClients>

Number of clients the server is aware of, including those registered at peers.

<MyClientsUp>

Number of clients registered at this server and currently reachable.

<MyClientsDown>

Number of clients registered at this server and currently unreachable (probably off).

<TotalRegistrations>

Total number of messages received.

<MissedRegistrations>

Number of messages to which go-susi did not manage to reply in less than 8s.

<AvgRequestTime>

Time in nanoseconds go-susi took to process requests averaged over the last 100. The time for reading the request from the network and writing the reply is not included.

<mallinfo_arena>

The number of non-mmap'ed bytes currently reserved by malloc-based memory management. This includes memory that has already been free()d but not returned to the OS, yet.

Note: malloc-based memory is not managed by the Go runtime and will not be garbage collected.

<mallinfo_fordblks>

The number of bytes within **<mallinfo_arena>** that have been free()d. These bytes have not been returned to the OS (i.e. they're reported as in-use in the output of top and ps) even though they are not actually used by the program. The memory management keeps these bytes reserved either because of memory fragmentation or to improve performance.

Note: If **<mallinfo_arena>** and **<mallinfo_fordblks>** keep growing together over the lifetime of the program this means that the program has problems with memory fragmentation. It's unlikely you'll ever see this.

If the difference **<mallinfo_arena> - <mallinfo_fordblks>** keeps growing over the lifetime of the program this means that the program has a memory leak. Report this as a bug.

<mallinfo_hblkhd>

Number of bytes allocated by malloc-based memory management using mmap(). This kind of memory, unlike **<mallinfo_arena>** is always returned to the OS immediately when free()d, so all of these bytes are actually in use by the program.

Note: If this number keeps growing over the lifetime of the program this means that the program has a memory leak. Report this as a bug.

<mallinfo_keepcost>

Number of bytes in **<mallinfo_arena>** that could be returned to the OS but are kept to make future allocations faster.

Note: This number should always be close to **<mallinfo_fordblks>**. If there's a large difference between the two it means there's an issue with memory fragmentation.

<mallinfo_hblks>

The number of mmap'ed blocks that hold the memory reported as **<mallinfo_hblkhd>**.

Note: This number should be fairly constant and small over the lifetime of the program. If it keeps growing this means the program has a memory leak. Report this as a bug.

gosa_trigger_reload_ldap_config

Purpose:

GOsa➔Server. Causes the server to send a [illegible] message with up-to-date data. GOsa sends this after operations on the client's LDAP object that have potentially changed the relevant attributes.

Example message:

```
[illegible]
    [illegible]                                          </header>
    [illegible]    </source>
    [illegible]                       </target>
    [illegible]                            </macaddress>
[illegible]
```

gosa_recreate_fai_release_db

Purpose:

GOsa➔Server. GOsa sends this message whenever a FAI class has been added, modified or removed.

Example message:

```
[illegible]
    [illegible]                                     </header>
    [illegible]    </source>
    [illegible]    </target>
    [illegible]       </macaddress>
    [illegible]    </periodic>
[illegible]
```

go-susi note:

Because gosa-si-server does not forward this message to peer servers, it is not a good idea to rely on it. Therefore go-susi does not use this message and instead uses a cache with a very short lifetime.
If this message is ever implemented in go-susi, go-susi will probably forward it to peers (depending on whether **<source>** is GOSA or a peer, to avoid forwarding an already forwarded message).

Deprecated

The messages in this section are concerned with user presence tracking and delivering messages to individual users. Like the homegrown encryption protocol this message delivery service is badly designed and go-susi does not support it. The hook can be used to interface go-susi with an external messaging service such as Jabber or email.

CLMSG_CURRENTLY_LOGGED_IN

Purpose:

Client➔Server. After the client has registered at the server it will send this message to tell the server the login names of all users currently logged in (including ssh logins). This corresponds to the users listed by the command.

Updates to the list of logged in users are transmitted via and messages.

When the server receives this message, it sends an message with **<user_db>** elements to all peers.

Example message:

```
        </header>
        </source>
        </target>
        TLY_LOGGED_IN>

        NTLY_LOGGED_IN>
        </macaddress>
        </timestamp>
```

The message elements have the following meaning:

N>

a space-separated list of login names, including users with graphical as well as ssh sessions. The list may contain duplicate entries if a user has multiple simultaneous sessions.

The **UTC** time at which the message is being sent.

MAC address of the sending client.

go-susi note:

go-susi currently does not send messages.

CLMSG_LOGIN

Purpose:

Client➔Server. When a user opens a new session the client sends this message to the server. When the server receives this message, it sends an [illegible] message with **<new_user>** elements to all peers.

Example message:

```
[illegible]
    [illegible]    [illegible]    </header>
    [illegible]    [illegible]      </source>
    [illegible]    [illegible]     </target>
    [illegible]      [illegible]  </CLMSG_LOGIN>
    [illegible]      [illegible]      </macaddress>
    [illegible]      [illegible]     </timestamp>
[illegible]
```

The message elements have the following meaning:

[illegible]

Login name of the user who has just opened a new session.

[illegible] see[illegible]

gosa-si-client note:

gosa-si-client does not seem to notice new ssh sessions, even though existing ssh sessions are included in the list from [illegible]

go-susi note:

go-susi currently does not send [illegible] messages.

CLMSG_LOGOUT

Purpose:

Client➔Server. When a user terminates a login session the client sends this message to the server. When the server receives this message, it does *not* send an [illegible] message to its peers.

Example message:

```
[illegible]
    [illegible]    [illegible]    </header>
    [illegible]    [illegible]      </source>
    [illegible]    [illegible]    </target>
    [illegible]      >[illegible]    </CLMSG_LOGOUT>
    [illegible]    [illegible]      </macaddress>
    [illegible]    [illegible]    </timestamp>
[illegible]
```

The message elements have the following meaning:

[illegible]

Login name of the user who has just closed a session. Note that the same user may still have other sessions open.

[illegible] see [illegible]

gosa-si-client note:

gosa-si-client does not seem to notice when ssh sessions are terminated, even though existing ssh sessions are included in the list from [illegible]

information_sharing

Purpose:

Server➔Server. Informs the receiving server about users currently logged in at clients registered at the sending server. Whenever a client sends a [illegible] message to its server, that server sends [illegible] messages to its peers with **<new_user>** information.
Whenever a client sends a [illegible] message to its server, that server sends [illegible] messages to its peers with **<user_db>** information.
When the client sends [illegible] to its server, gosa-si-server does not seem to forward that information to its peers.

Example messages:

```
[illegible]
    [illegible]  [illegible]  </header>
    [illegible]  [illegible]  </source>
    [illegible]  [illegible]  </target>
    [illegible]  [illegible]  </user_db>
    [illegible]  [illegible]  </user_db>
    [illegible]  [illegible]  </user_db>
    [illegible]  sharing></information_sharing>
[illegible]

[illegible]
    [illegible]  [illegible]  </header>
    [illegible]  [illegible]  </source>
    [illegible]  [illegible]  </target>
    [illegible]  [illegible]  </new_user>
    [illegible]  sharing></information_sharing>
[illegible]
```

The message elements have the following meaning:

<user_db>

When a **<user_db>** element is present it means that the [illegible] message lists *all* users currently logged in at clients registered at the **<source>** server. Each such user is listed in its own **<user_db>** element. The receiving server should discard all old user information from the **<source>** server and replace it with the new information.

<new_user>

The **<new_user>** element is like **<user_db>** with the difference that the [illegible] message does not list all users. This means that the receiving server should add the new user information to the old information from **<source>** rather than discard all old information as it would do in the case of **<user_db>**.

go-susi notes:

Currently go-susi understands but does not send these messages.

usr_msg

Purpose:

Server➔Client. Tell a client to present a popup message to the user.

Example message:

```
</header>
                </source>
                </target>
        </message>
        </subject>
usr>
r_msg>
```

go-susi note:

go-susi renames all **<usr>** elements to **<user>**, then calls See for more information.

confirm_usr_msg

Purpose:

Client➔Server. Tells the server that a message sent to the client via has been presented to the user.

Example message:

```
</header>
                </source>
                </target>
        </message>
        </subject>
usr>
msg></confirm_usr_msg>
```

go-susi note:

go-susi does not support this message.

Appendix

sibridge

Remote control for an si-server.

SYNOPSIS

DESCRIPTION

Remote control for an si-server at

OPTIONS

print usage and exit.

print version and exit.

read configuration from <file> instead of the default location (which is). The format of the configuration file is the same as for GOsa² (*not* go-susi!) but only the sections matter. There has to be a section for

listen for socket connections on <port>. Always uses TLS without STARTTLS (unlike go-susi). TLS client authentication is required. GOsa extensions with appropriate access control bits need to be present in the client certificate.
If the attribute of in the config file lists multiple paths, client certificates may be signed by any of these CAs.

execute commands from <string>.

execute commands from <file>. If <file> is not an ordinary file, it will be processed concurrently with other special files and data from other -e and -f arguments. This permits using FIFOs and other special files for input.

Read from stdin even if -l, -e or -f is used. Normally these switches suppress interactive mode.

log INFO level log messages (by default only ERRORs are logged). INFO level messages may aid the administrator in debugging problems.

log INFO and DEBUG level log messages. DEBUG level messages are useful only for developers and may produce so much data that it affects performance. They also contain cleartext passwords.

TLS CERTIFICATES

In order to connect to an sibridge put into listening mode with the switch a client must present a certificate. GOsa extensions are required and most access controls apply to sibridge in an obvious way. E.g. the bit GosaAccessControl.GosaAccessJobs.install controls whether sibridge will accept commands that create new install jobs. Note that the effective permissions a user connecting to sibridge are the intersection of the permissions granted by the client certificate and the certificate sibridge uses to connect

to go-susi, which is always the certificate configured in sibridge's configuration file, regardless of what certificate is used to connect to sibridge.
The following list contains aspects in which sibridge's handling of access controls differs from go-susi's or is non-obvious:

- sibridge ignores GosaAccessLDAPIncoming. Adding an empty GosaAccessLDAPIncoming list in combination with not setting the GosaAccessMisc.peer bit is an easy way to make sure that a certificate can not be used to connect to go-susi directly.
- sibridge *does* check GosaConnectionLimits.communicateWith. It applies to the machine and port sibridge is listening at, not the go-susi target. In other words you can use communicateWith to specify which sibridge instances a certificate can be used to talk to.
- In order to use the [illegible] command, the client needs the GosaAccessMisc.debug permission.
- Commands [illegible] and [illegible] require the GosaAccessLDAPUpdate.dh permission. Commands [illegible] and [illegible] additionally require GosaAccessLDAPDetectedHardware.dn.
- The commands [illegible] and [illegible] require GosaAccessQuery.queryAll.

generate_package_list

Debian repository scanner to serve as [illegible]

SYNOPSIS

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

DESCRIPTION

generate_package_list scans a list of Debian repositories, extracts package information and prints it to stdout in LDIF format as required for use as package-list-hook. For details about this format, see the section about package-list-hook in the documentation on go-susi's configuration options.

WARNING

If requested (see ENVIRONMENT section below) generate_package_list will scan *all* Debian packages in the target repositories for debconf-templates. This requires that *all* packages be transferred at least partially and will cause significant network traffic. It will also take a lot of time. Because generate_package_list has an internal timeout of 1h after which it will stop scanning, it may be necessary to call it multiple times to complete the scan. Packages that were scanned in a previous run will not be rescanned. Their data will be used from the cache instead.

ENVIRONMENT

The parameters to control generate_package_list are passed via environment variables. Remember that the names of environment variables are case-sensitive!

[illegible]
: Path of a directory where generate_package_list should store its cache file.

[illegible]
: Controls which .deb packages will be (partially) downloaded for extracting debconf-templates. The following values are possible:
[illegible] Do not scan any packages for templates. Templates information already in the cache will be used.
[illegible] Only packages that have a Depends or Pre-Depends line that includes the word "debconf" will be scanned.
everything else: Scan all packages whose debconf-templates (or lack thereof) are not yet registered in the cache. See WARNING section above.

[illegible]
: A space-separated list of strings in the same format as the FAIrepository LDAP attribute. All of the corresponding repositories will be scanned.

[illegible]
: If set to 1, 2, 3 or 4 generate_package_list will output informative messages to stderr. Higher numbers produce more output.

initrd_autopack

A godsend for people developing/debugging initrd.img

SYNOPSIS

DESCRIPTION

/usr/lib/go-susi/initrd_autopack is a TFTP hook that allows you to maintain your initrd.img as an extracted directory tree. When the file is requested, initrd_autopack will automatically compress it. The compressed file will be kept so that recompression happens only when there have been changes in the extracted directory tree.

OPTIONS

-f

A TFTP request for /initrd.img addresses the path **tftp_dir**/initrd.img. If this file exists and is up-to-date with respect to the corresponding extracted directory tree, it will be served directly. If this file does not exist or the extracted directory tree has been changed since the last recompression, the extracted directory tree will be compressed and stored at this location and then served.

-d

A TFTP request for /initrd.img corresponds to an extracted directory **extract_dir**/initrd.img. If this directory does not exist, but the corresponding **tftp_dir**/initrd.img does, then **tftp_dir**/initrd.img will be extracted. This means that to start working on the insides of an initrd.img, you can just throw it into **tftp_dir**/ and boot it once. Afterwards you will find an extracted directory corresponding to the initrd.img that you can make your modifications to.

TLS Certificates

Details on X.509 certificates for establishing TLS connections with go-susi

DESCRIPTION

go-susi requires that all parties connecting via TLS present a valid certificate signed by a trusted CA (i.e. it requires TLS client authentication). In addition to the basic validity and trust checks, go-susi evaluates several certificate extensions, if present in the connecting party's certificate, to determine what messages go-susi will accept from that party and what actions go-susi will take in reaction to these messages.

subjectAltName

The certificate must contain the subjectAltName extension and the connecting party's IP address has to match one of the entries in subjectAltName. The following types of subjectAltName entries are supported:

- If the address in the certificate ends in one or more 0-bytes (e.g. 1.2.3.0 or 1.2.0.0), it is interpreted as a wildcard address that will match all IP addresses from the corresponding /24, /16 or /8 subnet. The address 0.0.0.0 means that *every* IP address will be accepted. The address 127.0.0.1 may be used to match parties connecting from the same machine that go-susi is running on.
- If the dNSName can be resolved to an IP, this IP must match the connecting party's IP. If it can not be resolved, the connecting party's IP will be converted to a name by reverse DNS and that name must match the certificate's dNSName. In this case the dNSName may start with a "*" which is interpreted as a wildcard. E.g. dNSName "*.example.com" matches name "foo.bar.example.com".
- The following predefined OBJECT IDENTIFIERs are supported:

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
id-msb OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 45753 }
id-msb-gosa OBJECT IDENTIFIER ::= { id-msb 1 }

gosa-gn-my-server OBJECT IDENTIFIER ::= { id-msb-gosa 1 }
  -- corresponds to the IP address of the server to which the
  -- receiving go-susi has sent its most recent [illegible] message.
  -- Note: go-susi updates this IP address before sending the
  -- [illegible] to make sure that it will accept a reply from a
  -- server that only has a gosa-gn-my-server certificate.

gosa-gn-config-file OBJECT IDENTIFIER ::= { id-msb-gosa 2 }
  -- corresponds to the set of all servers explicitly listed in the
  -- [illegible] and [illegible] configuration
  -- settings. All these entries will be treated as if they were dNSName
  -- or iPAddress entries in the certificate (depending on whether they
  -- are names or numeric addresses).

gosa-gn-srv-record OBJECT IDENTIFIER ::= { id-msb-gosa 3 }
  -- corresponds to the set of all servers listed in DNS SRV records for
  -- the service "gosa-si" with protocol "tcp". All these records will be
```

```
  -- treated as if they were dNSName entries in the certificate.
  -- If [illegible] or [illegible] is set
  -- to "false" in the configuration file, gosa-gn-srv-record has no effect.

gosa-gn-my-peer OBJECT IDENTIFIER ::= { id-msb-gosa 4 }
  -- corresponds to the set of all servers with whom the receiving go-susi
  -- has established server-server-communication.
  -- Note: A server whose certificate is bound to gosa-gn-my-peer can
  -- not establish server-server-communication on its own initiative.
  -- However, once a server with a more powerful certificate has
  -- contacted it with [illegible] it may respond and this will lead
  -- to a peer relationship.

END
```

Connection and data transfer limits

The following certificate extension may be used to impose or relax limits on the party presenting this certificate. Except for the communicateWith field which has some general usefulness, the fields in this extension are rarely useful. The default limits are appropriate for most cases. You should not include this extension in a certificate without a good reason to do so.

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

gosa-ce-connectionLimits OBJECT IDENTIFIER ::= { id-msb-gosa 5 }

GosaConnectionLimits ::= SEQUENCE {

  totalTime [0] INTEGER OPTIONAL,
  -- Number of milliseconds after which the connection will be terminated
  -- regardless of the amount of data transferred. In particular this will
  -- terminate transfers that take too long due to actual or simulated
  -- transmission errors.
  -- A number <= 0 means no time limit.
  -- Note: A non-0 setting in a certificate used for
  -- server-server-communication may cause job database inconsistencies
  -- between the respective servers.
  -- Do not use it in certificates used for server-server-communication.

  totalBytes [1] INTEGER OPTIONAL,
  -- Maximum number of bytes allowed over a single connection.
  -- Both bytes read and written count towards this limit.
  -- (The same connection may be used for multiple messages.)
  -- A number <= 0 means no limit.
  -- Note: A non-0 setting in a certificate used for
  -- server-server-communication may cause job database inconsistencies
  -- between the respective servers.
  -- Do not use it in certificates used for server-server-communication.

  messageBytes [2] INTEGER OPTIONAL,
```

-- Maximum number of bytes allowed for a single message.
-- Reply bytes do not count against this limit.
-- (The same connection may be used for multiple messages.)
-- A number <= 0 means no limit.

connPerHour [3] INTEGER OPTIONAL,
-- Maximum number of connection attempts per hour from the same IP
-- address. If the number is exceeded the connection will be rejected.
-- Repeated connection attempts after the limit has been exceeded
-- may prolong the lockout indefinitely.
-- Tracking of connections per time per IP is approximate only.
-- It is guaranteed that if a cool IP connects and then makes
-- connPerHour further connections before 1h has elapsed since
-- the first connection, this will trigger lockout. It is also guaranteed
-- that making no connections for 1h will completely cool down an IP.
-- The limit cannot be evaded by opening parallel connections.
-- A number <= 0 means no limit.
-- Note: This counts successful and unsuccessful connection attempts.
-- Once established a connection may be held open indefinitely without
-- incurring any further strikes against this limit.
-- Note 2: To make rejection of IPs that have exceeded their limits
-- as fast as possible, limits are stored in a DB after TLS handshake
-- but consulted before the TLS handshake. Connections from the
-- same IP with differing certificate limits will cause issues such as
-- locking out legitimate clients and permitting illegitimate clients to
-- exceed their limits.

connParallel [4] INTEGER OPTIONAL,
-- Maximum number of parallel connections from the same IP address.
-- A number <= 0 means no limit.
-- Note: To make rejection of IPs that have exceeded their limits
-- as fast as possible, limits are stored in a DB after TLS handshake
-- but consulted before the TLS handshake. Connections from the
-- same IP with differing certificate limits will cause issues such as
-- locking out legitimate clients and permitting illegitimate clients to
-- exceed their limits.

maxLogFiles [5] INTEGER OPTIONAL,
-- Maximum number of files allowed in a CLMSG_SAVE_FAI_LOG message.
-- A number <= 0 means no limit.

maxAnswers [6] INTEGER OPTIONAL,
-- Maximum number of <answerX> elements that will be returned in a query
-- reply. This is particularly relevant to prevent DOS by requesting all
-- packages from the package database.
-- A number <= 0 means no limit.
-- Note: A non-0 setting in a certificate used for
-- server-server-communication may cause job database inconsistencies
-- between the respective servers.
-- Do not use it in certificates used for server-server-communication.

```
  communicateWith [7] SEQUENCE OF UTF8String OPTIONAL
  -- List of DNS names and/or IP addresses with or without port number.
  -- DNS names may start with a "*" as wildcard character
  -- (e.g. "*.example.com:20081").
  -- A connection is only permitted with another system if that system's
  -- IP/Name/Port matches one of the entries in this list.
  -- When system A and system B verify each
  -- other's certificates at the beginning of a connection, system A will
  -- terminate theconnection immediately if system B's certificate contains
  -- a communicateWiththat does not match system A. It does not matter
  -- which system initiated the connection.
  -- An empty list prevents all incoming and outgoing connections.
}

END
```

Access control by feature

By default go-susi will refuse to answer queries or take any kind of action that has some persistent effect. The following certificate extension may be used to selectively permit queries and actions in reaction to messages from the owner of a certificate with this extension. This extension is optional for client certificates if you can live without automatic creation of LDAP objects for new systems and without automatic updating of cn/ipHostNumber in reaction to DNS changes. For server certificates however (including the one used by GOsa) this extension is required to allow essential functions.

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

gosa-ce-accessControl OBJECT IDENTIFIER ::= { id-msb-gosa 6 }

GosaAccessControl ::= SEQUENCE {
  misc  [0] GosaAccessMisc  OPTIONAL,
  query [1] GosaAccessQuery OPTIONAL,
  jobs  [2] GosaAccessJobs  OPTIONAL,
  incoming   [3] GosaAccessLDAPIncoming OPTIONAL,
  ldapUpdate [4] GosaAccessLDAPUpdate   OPTIONAL,
  detectedHw [5] GosaAccessLDAPDetectedHardware OPTIONAL
}

GosaAccessMisc ::= BIT STRING {

  debug(0),
  -- This flag permits access to functions that are only meant for
  -- debugging. These functions pose a security risk. This flag
  -- should not be set in a certificate used for production systems.

  wake(1),
  -- This flag enables the server-to-server function trigger_wake.

  peer(2)
  -- This flag enables the messages new_server,
  -- confirm_new_server and foreign_job_updates, that are the backbone
  -- of server-to-server communication.
  -- For servers listed in [illegible] in the config file
  -- this flag is always assumed to be set, even if they present a
  -- certificate without it. This allows setting up server-to-server
  -- communication via configuration files without issuing certificates with
  -- the peer flag.
  -- NOTE: This exception does NOT extend to servers listed in DNS!
}

GosaAccessQuery ::= BIT STRING {

  queryAll(0),
  -- This flag enables all messages that query information. This includes
  -- database queries such as [illegible] and messages like
```

-- s and g .
-- If this flag is set, other flags of g have no effect.
-- If it is 0, the other flags may be used to enable queries on an
-- individual basis.

queryJobs(1)
-- This flag only enables the message gosa_query_jobdb.
-- This flag is always assumed to be set when the message comes from
-- a peer server (see g) because server-to-server
-- communication uses these queries to synchronize job databases.
}

GosaAccessJobs ::= BIT STRING {

jobsAll(0),
-- This flag enables creating, removing and modifying of jobs. If this
-- flag is set, the other GA flags have no effect. If this
-- flag is 0, the other flags may be used to enable individual job
-- messages.
-- NOTE: Even though this flag enables g ,
-- the creation of new system objects in LDAP requires additional rights
-- from a .

lock(1),
-- This flag enables g and
-- g .

unlock(2),
-- This flag enables g ,
-- g , g and
-- g .

shutdown(3),
-- This flag enables g ,
-- g , g and
-- g .

wake(4),
-- This flag enables g and
-- g .

abort(5),
-- This flag enables g ,
-- g , g and
-- g . These messages abort running FAI jobs.

install(6),
-- This flag enables g and
-- g .

update(7),

-- This flag enables and
-- .

modifyJobs(8),
-- This flag enables and
-- .

newSys(9)
-- This flag enables and
-- .

userMsg(10)
-- This flag enables .

audit(11)
-- This flag enables and
-- .
}

GosaAccessLDAPIncoming ::= SEQUENCE OF UTF8String
-- This is a list of URIs of LDAP servers, each optionally followed by
-- a DN (e.g. ldaps://ldap.foo.de:389/ou=incoming,o=foo,c=de).
-- The server component of the URI may start with "*" as
-- a wildcard and the DN may either start or end with a "*". Both
-- parts are permitted to be just "*" (e.g. ldap://*/*).
-- When a client sends its to a server and the client's
-- certificate contains a G list, the
-- registration will be denied if none of the LDAP servers in the
-- list matches the one used by the contacted server.
-- If present, an entry's optional DN has to match the contacted
-- server's configuration option
-- and registration will succeed even if there is no LDAP object for
-- the client yet.
-- If the optional DN is missing from the list entry, then registration
-- will succeed only if LDAP already contains an object for
-- the client.
-- An empty list prevents registration
-- with any server.
-- If the list is not present at all in the
-- certificate, this is equivalent to a list ["ldap://*", "ldaps://*"]
-- which permits registration at any server but only if there is an
-- existing LDAP object for the client.
-- NOTE: The protocol part of list entries is significant. An entry that
-- starts with "ldaps:" will only allow registration at servers that talk to
-- LDAP via a secure connection.

GosaAccessLDAPUpdate ::= BIT STRING {

cn(0),
-- When a client registers at a server and the cn of the

-- client's LDAP entry (as found by its MAC address)
-- does not match the reverse DNS of its IP, the server's
-- behaviour depends on this flag in the client's certificate:
-- If the flag is set, the registration succeeds and the LDAP
-- entry is updated with the new name.
-- If the flag is unset or [illegible] is not present
-- in the certificate, registration will fail.

ip(1),
-- When a client registers at a server and the ipHostNumber of the
-- client's LDAP entry (as found by its MAC address)
-- does not match the client's IP, the server's
-- behaviour depends on this flag in the client's certificate:
-- If the flag is set, the registration succeeds and the LDAP
-- entry is updated with the new IP.
-- If the flag is unset or [illegible] is not present
-- in the certificate, registration will fail.

mac(2),
-- When a client registers at a server and there is no LDAP entry for its
-- MAC address, but there is an LDAP entry whose ipHostNumber
-- matches the client's IP, then the server's behaviour depends on this
-- flag:
-- If the flag is unset (or [illegible] is not present)
-- registration will fail even if [illegible] would permit
-- creation of a new LDAP entry.
-- If the flag is set and the cn of the existing LDAP entry matches the
-- reverse DNS of the client's IP, or the **cn** flag is set, then the
-- macaddress attribute of the existing entry will be updated and
-- registration succeeds.
-- If the flag is set but the client's reverse DNS does not match the cn
-- and the **cn** flag is not set, then registration will fail even if
-- [illegible] would permit creating a new LDAP entry.

dh(3)
-- This flag enables using the [illegible] message to modify
-- existing LDAP entries. If this flag is unset or [illegible]
-- is not present, the server will only accept [illegible] if there
-- is no existing LDAP entry. This is not necessary for normal FAI/GOsa
-- operation.
-- See also [illegible] .
}

GosaAccessLDAPDetectedHardware ::= BIT STRING {
-- ATTENTION! The following flags enable extended functionality of
-- the [illegible] message that permits a variety of modifications
-- of LDAP objects.
-- The **template** flag has some use cases in a certificate used in a FAIROOT
-- for installing new clients. All of the other flags are useful only for
-- special applications such as interfacing with other software distribution
-- systems or batch importing systems into LDAP.

-- Do not use any of the below flags in a certificate unless you have a
-- specific reason to do so.

unprompted(0),
-- This flag allows the sending of the [illegible] message to a
-- server even if that server has not requested it by sending a
-- [illegible] message.

template(1),
-- This flag tells the server that it should use template objects (see
-- description of detected_hardware message) when processing a
-- [illegible]e message.

dn(2),
-- This flag permits the sender of [illegible] to specify the DN
-- of the LDAP object to create in the [illegible] message, even
-- if that DN is not in the tree configured as [illegible].
-- In combination with the **dh** flag from [illegible]e
-- this even permits moving of systems within the LDAP directory.

cn(3),
-- This flag permits the sender of [illegible] to specify the CN
-- of the LDAP object to create in the [illegible] message,
-- even if it does not match the name obtained from reverse DNS of
-- the sender's IP.

ipHostNumber(4),
-- This flag permits the sending of a detected_hardware message containing
-- a <ipHostNumber> element with a different IP than that of the sender.

macAddress(5)
-- This flag permits specifying the MAC address in a [illegible]
-- message. Without this flag the MAC address from the most recent
-- [illegible] or [illegible] is used that matches the <source>
-- of the detected_hardware message.
-- ATTENTION! This flag allows the creation of multiple LDAP objects
-- with the same [illegible].
}

END

License