GLPI Agent

Release 1.0-git*

GLPI Project, Teclib'

Dec 10, 2021

CONTENTS

CHAPTER

ONE

INSTALLATION

The latest release is available from our github releases page.

Note: In the case you're replacing FusionInventory agent with GLPI Agent, you should first uninstall **FusionInventory agent** before installing **GLPI Agent**. You can use the same dedicated configuration if you placed it in *conf.d* configuration subfolder.

Hint: Nightly builds are also available from our dedicated GLPI-Agent Nightly Builds page.

1.1 Windows

1.1.1 Windows installer

By default, the installer will bring you to the graphical user interface unless you use the */i /quiet* options, calling it from command line.

```
C:                  > GLPI-Agent-1.0-git*-x64.msi /i /quiet SERVER=<URL>
```

or:

```
C:              > msiexec /i GLPI-Agent-1.0-git*-x64.msi /quiet SERVER=<URL>
```

All options can be defined in several places; the last taking precedence on all others:

- default values,
- values from the current GLPI Agent installation,
- values from the command line,
- values from the graphical installer.

Note: When using command line parameters, you should keep in mind:

- parameters beggining with a slash are indeed `msiexec.exe` options,
- an equal sign is always required for other parameters: `TAG=prod`,
- options names are case-sensitive,
- options values are *not* case-sensitive, unless specified,
- the equal sign must not be preceded or followed with a space character: `LOCAL = C:\temp` is incorrect,

- if a value contains a space, it must be surrounded with single ' or double quotes ",
- if you want to set a empty value, put an empty string (LOCAL= or LOCAL="").

1.1.2 Command line parameters

/i Specify this is a normal installation. This is indeed a `msiexec.exe` option.

When used with `msiexec.exe`, it must be used just before the MSI package path.

/quiet Silent installation. This is indeed a `msiexec.exe` option. (By default: No)

ADD_FIREWALL_EXCEPTION=1 Adds GLPI Agent to the Windows Firewall exception list. (By default: `0` for No)

ADDLOCAL=feature[,feature[...]] This parameter permits to select features to install. (By default: "feat_DEPLOY,feat_COLLECT")

The *feature* can take the following values:

- `ALL`: All included tasks are selected for installation
- `feat_NETINV`: to select NetDiscovery and NetInventory tasks for installation
- `feat_DEPLOY`: to select Deploy task for installation
- `feat_COLLECT`: to select Collect task for installation
- `feat_WOL`: to select WakeOnLan task for installation

The base feature is feat_AGENT which is always selected and includes Inventory task. By default, Deploy and Collect tasks are also selected.

BACKEND_COLLECT_TIMEOUT=180 Timeout for task `Inventory` modules execution. (By default: `180` seconds)

CA_CERT_DIR=pathname Absolute path to the standard certificate directory of certificate authorities (CA). (By default: empty)

The use of this parameter is incompatible with the use of the `CA_CERT_FILE` parameter. The `CA_CERT_DIR` and `CA_CERT_FILE` parameters are mutually exclusive.

A *standard certificate directory* must contain the certificate files corresponding to different certificate authorities in Privacy Enhanced Mail (PEM) format, and their names must correspond to the hash value of the certificate's *subject* field, with extension `.0`.

For example, if you want to include the certificate file *FICert_Class1.crt* in the directory *pathname*, you must calculate the hash value of the certificate's *subject* field using, for example, OpenSSL

```
C:        > openssl.exe x509 -in C:_Class1.crt -subject_hash -noout b760f1ce
```

and afterwards, move or copy the certificate file to the directory *pathname* with the name `b760f1ce.0`.

```
C:                    > move.exe C:_Class1.crt pathname760f1ce.0
```

CA_CERT_FILE=filename Full path to the certificates file of certification authorities (CA). (By default: empty)

The use of this parameter is incompatible with the use of the `CA_CERT_DIR` parameter. The `CA_CERT_DIR` and `CA_CERT_FILE` parameters are mutually exclusive.

filename must have extension `.pem` (Privacy Enhanced Mail) and can contain one or more certificates of certificate authorities. To concatenate multiple certificate files into one file you can make use, for example, of the command *copy*.

```
C:            > copy.exe FICert_Class1.crt+FICert_Class2.crt FICerts.pem
```

DEBUG=level Sets the debug level of the agent. (By default: `0`)

level can take one of the following values:

- `0`: Debug off
- `1`: Normal debug
- `2`: Full debug

DELAYTIME=limit Sets an initial delay before first contact with a remote destination (see `SERVER`). This delay is calculated at random between *limit/2* and *limit* seconds. (Default: `3600` seconds)

This parameter is ignored for remote destinations after the first contact with each one, in favor of the specific server parameter (PROLOG_FREQ or Contact expiration).

The `DELAYTIME` parameter comes into play only if GLPI Agent runs in *server mode* (see `EXECMODE`).

EXECMODE=value Sets the agent execution mode. (By default: `1`)

mode can take one of the following values:

- `1` for `Service`: The agent runs as a Windows Service (always running)
- `2` for `Task`: The agent runs as a Windows Task (runs at intervals)
- `3` for `Manual`: The agent doesn't run automatically (no `Service`, no `Task`)

The mode `Service` is known also as *server mode*.

The mode `Task` is only available on Windows XP (or higher) and Windows Server 2003 (or higher) operative systems.

HTML=value Save the inventory as HTML instead of XML. (By default: `0` for No)

The `HTML` parameter comes into play only if you have also indicated a value for the `LOCAL` parameter.

HTTPD_IP=ip IP address by which the embedded web server should listen. (By default: `0.0.0.0`)

HTTPD_PORT=port IP port by which the embedded web server should listen. (By default: `62354`)

HTTPD_TRUST={ip|range|hostname}[,{ip|range|hostname}[...]] Trusted IP addresses that do not require authentication token by the integrated web server. (By default: 127.0.0.1/32)

ip is an IP address in dot-decimal notation (ex. "127.0.0.1") or in CIDR notation (ex. "127.0.0.1/32")

range is an IP address range in dot-decimal notation (ex. "192.168.0.0 - 192.168.0.255" or "192.168.0.0 + 255") or in CIDR notation (ex. "192.168.0.0/24")

hostname is the name of a host (ex. "itms.acme.org")

Keep in mind that `HTTPD_TRUST` does not have to include the hostname part of those URIs that are set up in `SERVER` because they are tacitly included. The following is an example, both configurations are equal:

```
... HTTPD_TRUST="127.0.0.1/32,itms.acme.org" \
    SERVER="http://itms.acme.org/glpi/front/inventory.php"
```

```
... HTTPD_TRUST="127.0.0.1/32" \
    SERVER="http://itms.acme.org/glpi/front/inventory.php"
```

INSTALLDIR=pathname Sets the installation base directory of the agent. (By default: `C:\Program Files\GLPI-Agent`)

pathname must be an absolute path.

LAZY=1 Contact server only if the server expiration delay has been reached. (By default: `0`)

This option is only used if you set `EXECMODE=2` to use Windows Task scheduling.

LOCAL=pathname Writes the results of tasks execution into the given directory. (By default: empty)

You must indicate an absolute pathname or an empty string (""). If you indicate an empty string, the results of tasks execution will not be written locally.

You can use the `LOCAL` and `SERVER` options simultaneously.

LOGFILE=filename Writes log messages into the file *filename*. (By default: `C:\Program Files\GLPI-Agent\logs\glpi-agent.log`)

You must indicate a full path in *filename*. The `LOGFILE` parameter comes into play only if you have also indicated `file` as a value of the `LOGGER` parameter, which is the default.

LOGFILE_MAXSIZE=size Sets the maximum size of logfile (see `LOGFILE`) to *size* in MBytes. (By default: 4 MBytes)

LOGGER=backend[,backend] Sets the logger backends. (By default: `file`)

backend can take any of the following values:

- `file`: Sends the log messages to a file (see `LOGFILE`)
- `stderr`: Sends the log messages to the console

NO_CATEGORY=category[,category[...]] Do not inventory the indicated categories of elements. (By default: empty)

category can take any value listed by the following command:

```
C:                    > "C: Files-Agent-agent" --list-categories
```

NO_HTTPD=1 Disables the embedded web server. (By default: `0`)

NO_P2P=1 Do not use peer to peer to download files. (By default: `0`)

NO_SSL_CHECK=1 Do not check server certificate. (By default: `0`)

NO_TASK=task[,task[...]] Disables the given tasks. (By default: empty)

task can take any of the following values:

- `Deploy`: Task Deploy
- `ESX`: Task ESX
- `Inventory`: Task Inventory
- `NetDiscovery`: Task NetDiscovery
- `NetInventory`: Task NetInventory
- `WakeOnLan`: Task WakeOnLan

If you indicate an empty string (""), all tasks will be executed.

PASSWORD=password Uses *password* as password for server authentication. (By default: empty)

The `PASSWORD` comes into play only if you have also indicated a value for the `SERVER` parameter.

PROXY=URI Uses *URI* as HTTP/S proxy server. (By default: empty)

QUICKINSTALL=1 Don't ask for detailed configurations during graphical install. (By default: `0`)

RUNNOW=1 Launches the agent immediately after its installation. (By default: `0`)

SCAN_HOMEDIRS=1 Allows the agent to scan home directories for virtual machines. (By default: `0`)

SERVER=URI[,URI[...]] Sends results of tasks execution to given servers. (By default: empty)

If you indicate an empty string (""), the results of tasks execution will not be written remotely.

You can use the SERVER and LOCAL parameters simultaneously.

TAG=tag Marks the computer with the tag *tag* . (By default: empty)

TASKS=task[,task[,...]] Plan tasks in the given order. (By default: empty)

Not listed tasks won't be planned during a run, unless `,...` is specified at the end.

task can take any of the following values:

- `Deploy`: Task Deploy
- `ESX`: Task ESX
- `Inventory`: Task Inventory
- `NetDiscovery`: Task NetDiscovery
- `NetInventory`: Task NetInventory
- `WakeOnLan`: Task WakeOnLan

If you indicate an empty string (""), all tasks will be executed. If you indicate `,...` at the end, all not listed tasks will be added in any order. You can indicate a task more than one time if this makes sens.

TASK_DAILY_MODIFIER=modifier Daily task schedule modifier. (By default: 1 day)

modifier can take values between 1 and 365, both included.

The TASK_DAILY_MODIFIER parameter comes into play only if you have also indicated `daily` as value of the TASK_FREQUENCY option.

TASK_FREQUENCY=frequency Frequency for task schedule. (By default: `hourly`)

frequency can take any of the following values:

- `minute`: At minute intervals (see TASK_MINUTE_MODIFIER parameter)
- `hourly`: At hour intervals (see TASK_HOURLY_MODIFIER parameter)
- `daily`: At day intervals (see TASK_DAILY_MODIFIER parameter)

TASK_HOURLY_MODIFIER=modifier Hourly task schedule modifier. (By default: 1 hour)

modifier can take values between 1 and 23, both included.

The TASK_HOURLY_MODIFIER parameter comes into play only if you have also indicated `hourly` as value of the TASK_FREQUENCY parameter.

TASK_MINUTE_MODIFIER=modifier Minute task schedule modifier. (By default: 15 minutes)

modifier can take the any value from 1 to 1439.

The TASK_MINUTE_MODIFIER parameter comes into play only if you have also indicated `minute` as value of the TASK_FREQUENCY parameter.

TIMEOUT=180 Sets the limit time (in seconds) to connect with the server. (By default: 180 seconds)

The TIMEOUT parameter comes into play only if you have also indicated a value for the SERVER parameter.

USER=user Uses *user* as user for server authentication. (By default: empty)

The USER parameter comes into play only if you have also indicated a value for the SERVER parameter.

VARDIR=pathname Sets the vardir base directory of the agent. (By default: `C:\Program Files\GLPI-Agent\var`)

This parameter can be used when the agent is installed in a shared storage.

pathname must be an absolute path.

The installer integrates its native, although reduced but recent, version of Strawberry Perl including recent OpenSSL support.

You can download the lastest GLPI Agent installer or current nightly build. It is available for both 32 and 64 bits systems and provides a graphical interface as well as command line facilities.

By default, it will perform a graphical installation, unless you use the msiexec */i* and */quiet* options. All installer parameters are described in *Windows installer* dedicated page.

Note: All graphical installer options are related to a command line one. Check *Windows installer* if you need help.

1.1.3 Large Installations

A VBScript (Visual Basic Script) is provided to deploy the installer on a network: `glpi-agent-deployment.vbs`.

1.2 MacOS

The installer integrates its native, although reduced but recent, version of Perl including recent OpenSSL support.

Get the latest `.pkg` package from our releases page or the nightly build page. After installing it, you'll have to configure the agent to your needs by creating a dedicated `.cfg` file under the `/Applications/GLPI-Agent/etc/conf.d` folder.

You can for example create a `local.cfg` file and :

- add the `server = GLPI_URL` line to point to your GLPI server,
- eventually set `debug = 1` to generate some debug in logs,
- set a `tag` like `tag = MyLovelyTag`.

1.3 GNU/Linux

We support major distros as we provides generic packages for **RPM** and **DEB** based distros as well if they supports **Snap** packaging. You can install required packages after getting them from our github releases page or the nightly build page.

Hint: When possible, prefer to use our *linux installer* as it supports **RPM** and **DEB** based distros and there's a version also including the **Snap** package. The linux installer accepts few options to configure the agent so it can simplify manual or automatic installation. It also can be handy for tools like Puppet or Ansible.

1.3.1 Snap

The Snapcraft **Snap** package integrates its native, although reduced but recent, version of Perl including recent OpenSSL support.

If your system support **Snap**, you can simply install the agent with the `snap` command after getting the **Snap** package from our releases page or the nightly build page. Then, you just have to run:

```
$ snap install --classic --dangerous GLPI-Agent-1.0-git*_amd64.snap
```

After installation, you can easily configure the agent with the **set** `snap` sub-command:

```
$ snap set glpi-agent server=http://myserver/front/inventory.php
```

Any supported glpi-agent option can be set this way. If you need to unset a configuration parameter, just set it empty:

```
$ snap set glpi-agent tag=
```

Note: You won't find the package in the Snapcraft store as their standard policies are too restrictive for GLPI Agent features and requirements.

1.3.2 Linux Installer

Attention: The **linux installer** main requirement is the **perl** command.

It also requires one of the following command, depending on the targeted system:

- **dnf** for recent **RPM** based systems
- **yum** for previous generation of **RPM** based systems
- **apt** for **DEB** based systems like Debian & Ubuntu
- **snap** for other systems supporting Snapcraft **Snap** packages

We also provide a dedicated linux installer which includes all the packages we build (**RPM** & **DEB**) and eventually the *snap* one. On supported distros (**DEB** & **RPM** based), the installer will also eventually try to enable third party repositories, like EPEL on CentOS if they are required.

The installer is a simple perl script. It supports few options to configure the agent during installation. You can check all supported options by running:

```
$ perl glpi-agent-1.0-git*-linux-installer.pl --help
```

or if you use the installer embedding **snap** package:

```
$ perl glpi-agent-1.0-git*-with-snap-linux-installer.pl --help
```

If your GNU/Linux distro is not supported, you still can *install it from sources*.

1.4 From sources

Note: We strongly recommend the use of *GNU tar* because some file path length are greater than 100 characters. Some tar version will silently ignore those files.

First, you need to extract the source and change the current directory.

```
$ tar xfz GLPI-Agent-1.0-git*.tar.gz
$ cd GLPI-Agent-1.0-git*
```

Executing `Makefile.PL` will verify all the required dependencies are available and prepare the build tree.

```
$ perl Makefile.PL
```

If you don't want to use the default directory (`/usr/local`), you can use the `PREFIX` parameter:

```
$ perl Makefile.PL PREFIX=/opt/glpi-agent
```

Note: At this point, you may have some missing required modules. See *PERL Dependencies* section for installing them. Once this is done, run the same command again.

You now can finish the installation. Here again we recommend *GNU make* (*gmake*):

```
$ make
$ make install
```

1.4.1 Tests

Note: The tests suite requires some additional dependencies like Test::More.

GLPI agent comes with a test-suite. You can run it with this command:

```
$ make test
```

1.4.2 Perl dependencies

The easiest way to install perl dependencies is to use cpanminus script, running:

```
$ cpanm .
```

You can use the `--notest` flag if you are brave and want to skip the tests suite execution for each install perl module.

Offline installations

Note: This requires the cpanminus script to be installed.

First grab the tarball from the website and extract it:

```
$ tar xzf GLPI-Agent-1.0-git*.tar.gz
$ cd GLPI-Agent-1.0-git*
```

We use `cpanm` to fetch and extract the dependencies in the extlib directory:

```
$ cpanm --pureperl --installdeps -L extlib --notest .
```

If this command fails with an error related to `Params::Validate`, then just run this last command:

```
$ cpanm --installdeps -L extlib --notest .
```

Now you can copy the directory to another machine and run the agent this way:

```
$ perl -Iextlib/lib/perl5 -Ilib glpi-agent
```

1.4.3 Other dependencies

On Solaris/SPARC, you must install sneep and record the Serial Number with it.

On Windows, we use an additional `dmidecode` binary shipped in the windows MSI package to retrieve many information not available otherwise, including fine-grained multi-cores CPUs identification. Unfortunately, this binary is not reliable enough to be used on Windows 2003, leading to less precise inventories.

On Linux, `lspci` will be used to collect PCI, AGP, PCI-X, ... information.

CHAPTER

TWO

CONFIGURATION

2.1 System location

On Unix, the agent reads its configuration from a configuration file named `agent.cfg`, whose location depends of the installation method:

- `/etc/glpi-agent/agent.cfg` on FHS (File System Hierarchy) compliant systems
- `/Applications/GLPI-Agent/etc/agent.cfg` on MacOS X pkg

More globally, you'll find that file in the GLPI Agent installation directory.

It is strongly discouraged to change this file, as you will probably loose your configuration on update (especially if you use a linux or MacOS X package).

Just ensure the `include conf.d/` is not commented (does not starts with a `#`). Your specific configuration should then go to any `conf.d/*.cfg` file.

On Windows, the agent read its configuration from a registry key, whose location may depends on architecture:

- `HKEY_LOCAL_MACHINE\SOFTWARE\GLPI-Agent` is the default,
- `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\GLPI-Agent` for 32bits agent installed on a 64bits system.

But windows portable version will use `agent.cfg` located under `etc` folder unless `--config=registry` option is used.

2.2 Parameter syntax

Most of the configuration options are single-valued; you can use a comma (`,`) as separator for multi-valued ones:

In configuration file:

```
logger = stderr,file
```

On command-line:

```
$ glpi-agent --logger=stderr,file
$ glpi-agent --logger stderr,file
```

2.3 Available parameters

Note: Most configuration options can be specified on command line ; this will override configuration file values in that case.

The only required configuration parameter is an execution target, which depends on the mode you will use:

- `server`: a server URL, such as `https://glpi/front/inventory.php`,
- `local`: full path for local directory, like `/tmp/inventory`.

Warning: Using multiple targets implies multiple executions of the same inventory ; this is not just a matter of targets. This can lead to different results, see *Multiple execution targets*.

server Specifies the server to use both as a controller for the agent, and as a recipient for task execution output.

If the given value start with `http://` or `https://`, it is assumed to be an URL, and used directly. Otherwise, it is assumed to be an hostname, and interpreted as `http://hostname/inventory`.

Multiple values can be specified, using a comma as a separator.

include This directive can only be used from a configuration file and permits to specify a file or a path from where to load any `*.cfg` files. The default is `conf.d` to load any `<INSTALLDIR>/etc/conf.d/*.cfg` file.

conf-reload-interval Automatically reload agent configuration after the given delay in seconds. The default is 0 which value just disables the feature.

delaytime Specifies the upper limit, in seconds, for the initial delay before contacting the control server. The default is 3600.

The actual delay is computed randomly between TIME / 2 and TIME seconds.

This directive is used for initial contact only, and ignored thereafter in favor of server-provided value in response from prolog or Contact request.

lazy Do not contact the control server before next scheduled time.

This directive is used when the agent is run in the foreground (not as a daemon) only.

no-task Disables given task.

Multiple values can be specified, using a comma as a separator.

proxy Specifies the URL of the HTTP proxy to use. By default, the agent uses HTTP_PROXY environment variable.

user Specifies the user to use for HTTP authentication on the server.

password Specifies the password to use for HTTP authentication on the server.

ca-cert-dir Specifies the directory containing indexed Certification Authority (CA) certificates.

ca-cert-file Specifies the file containing aggregated Certification Authority (CA) certificates.

no-ssl-check Disables server SSL certificate validation. The default is 0 (false).

timeout Specifies a timeout, in seconds, for server connections.

no-httpd Disables the embedded web interface, used to receive execution requests from the GLPI server or serve httpd plugins. The default is 0 (false).

httpd-ip Specifies the network interface to use for the embedded web interface. The default is to use all available ones.

httpd-port Specifies the network port to use for the embedded web interface. The default is 62354.

httpd-trust Specifies which IP address should be trusted, for execution requests. The default is to only accept requests from the control servers.

All formats supported by Net::IP can be used (IP addresses, IP addresses ranges, networks in CIDR notatation), as well as hostnames.

Multiple values can be specified, using a comma as a separator.

logger Specifies the logger backend to use. The possible values are:

- file: log messages in a file.
- stderr: log messages directly in the console.
- syslog: log messages through the local syslog server.

Multiple values can be specified, using a comma as a separator.

logfile Specifies the file to use for the file logger backend.

logfile-maxsize Specifies the maximum size for the log file, in MB. When the max size is reached, the file is truncated. The default is unlimited.

logfacility Specifies the syslog facility to use for the syslog logger backend. The default is LOG_USER.

color Enables color display for the stderr logger backend.

This directive is used on Unix only.

debug Specifies the level of verbosity for log content. The possible values are:

- 0: basic agent processing
- 1: extended agent processing
- 2: messages exchanged with the server and activates traces from Net::SSLeay if used

no-compression Disable compression when exchanging informations with GLPI Server. The default is to compress data.

This directive is only supported when server option is set.

listen Force agent to always listen for requests on httpd interface, even when no target is defined with server or local option.

This directive does nothing if server or local option is set.

vardir Set dedicated `vardir` path as agent storage. The default is `<INSTALLDIR>/var` on MacOSX, win32 or source install and generally `/var/lib/glpi-agent` on linux/unix when installed with a package.

2.4 Task-specific parameters

tag Specifies an arbitrary string to add to output. This can be used as an additional decision criteria on server side.

This directive is only for inventory or esx task only.

no-category Disables given category in output. The possible values can be listed running `glpi-agent --list-categories`. Some available categories:

- printer
- software
- environment
- process
- user

Multiple values can be specified, using a comma as a separator.

This directive is used for inventory task only.

additional-content Specifies an XML file whose content will be automatically merged with output. If inventory format is JSON, you can also specify a JSON file from which `content` base node will be merged.

This directive is used for inventory task only.

scan-homedirs Enables scanning user home directories for virtual machines (Any OS) or licenses (MacOS X only) . The default is 0 (false).

This directive is used for inventory task only.

scan-profiles Enables scanning profiles for softwares installation (Win32). The default is 0 (false).

This directive is used for inventory task only.

force Execute the task, even if not required by the server.

This directive is used for inventory task only.

collect-timeout Specifies the timeout for task modules execution.

This directive is used for inventory task only.

no-p2p Disables peer to peer for downloading files.

This directive is used for deploy task only.

html Output inventory in HTML format.

This directive is used for inventory task and for local target only.

json Use JSON as inventory format.

This directive is used for inventory task.

CHAPTER

THREE

USAGE

3.1 Running the agent

3.1.1 HTTP interface

If the agent is running as a service or a daemon, its web interface should be accessible at `http://hostname:62354`.

If the machine connecting to this interface is trusted (see *httpd-trust configuration directive*), a link will be available to force immediate execution.

3.1.2 Command line

Agent can also be executed from command line, using one of available executables.

Windows

Open a command interpreter windows (`cmd.exe`), with administrator privileges (*right click*, *Run as Administrator*).

Go to `C:\Program files\GLPI-Agent` (adapt path depending on your configuration) folder to run it:

```
C:                                  > cd "C: files-Agent"
C:                 Program files                 GLPI-Agent> glpi-agent
```

OS X

Ensure you have rights to run the executable with `sudo` command:

```
$ sudo /Applications/GLPI-Agent.app/bin/glpi-agent
```

Others

In most of the cases, you should just run (as an administrator):

```
$ glpi-agent
```

3.2 Execution modes

"How to run the agent" is not limited to a choice between running it either as a cron task (a scheduled task under Windows) or as a daemon (a service under Windows), because this only makes a difference in the control of the execution schedule, ie when the agent runs: in both cases, the server controls the execution plan, ie what the agent does. In order to control this execution plan on agent side, it is also possible to use a different set of executables. The different possibilities, designed as *execution modes*, are the following:

- **managed mode**: the agent runs continuously in the background, wakes up periodically to execute the tasks required by the server, and may eventually execute out of schedule on server request.
- **half-managed mode**: the agent only runs when launched by a local mechanism (usually an automated scheduling system, such as cron or task scheduler), contacts the servers to ask for a task list, executes them immediately, and stops.
- **autonomous mode**: the agent only runs when launched by a local mechanism, as in previous mode, executes a fixed list of tasks, sends the results to the server, and stops.

This table summarizes who controls what in each mode:

Execution mode	Execution schedule	Execution plan
Managed	Server-side control	Server-side control
Half-managed	Agent-side control	Server-side control
Autonomous	Agent-side control	Agent-side control

The correct mode to use for your needs mainly depends on the role assigned to your GLPI server:

- Is it a trusted management platform, or just a passive information database ? In the first case, you'd rather need server-side control, whereas on the second case, you'd rather need agent-side control

But it also depends on your technical expertise: if everything so far sounds has no meaning for you, no need to go further, just select managed mode. Otherwise, the following presents each mode with additional details.

3.2.1 Managed mode

This mode requires the agent to be run as a server process (daemon under Unix, service under Windows). The agent wake-up schedule is controlled from GLPI server, using PROLOG_FREQ or Contact expiration setting. For example, the agent wakes up at a random time between 90% and 100% of this value, ie for 24H, it will executes at sometime between 23 and 24H. Additionaly, the server may also initiate additional out-of-schedule executions by sending HTTP requests to the agent.

Example:

```
$ glpi-agent --server http://glpi/front/inventory.php --daemon
```

That's the easiest mode to set up, offering (almost) complete control from a centralized point, fully compatible with all available agent features, and the most flexible in terms of usage.

On the downside, this mode involves having a Perl interpreter loaded in memory permanently, which is insignificant on any modern desktop, but may eventually be a concern in specific memory-constraints scenario, such as IoT or minimal virtual machines. It also involves having a privileged process listening on a network port, unless run with `no-httpd` configuration directive (see *no-httpd configuration*).

And the more important: who controls the GLPI servers also controls all assets where an agent is installed, with ability to execute code at anytime, which may involve running arbitrary command with full privileges if related tasks (currently: deploy) are installed AND enabled on agent side. That's the exact purpose of this mode: everything the GLPI server wants, when it wants.

3.2.2 Half-managed mode

This mode requires a local triggering mechanism to launch the agent. It may be a scheduling system (cron, task scheduler) to run it automatically and regularly, but it may as well be a user session start script, for instance.

Example:

```
$ glpi-agent --server http://glpi/front/inventory.php
```

This mode doesn't consume memory permanently, only during agent execution. However, it is also less flexible, as scheduling can't get changed without reconfiguration. But the server still retains control over execution plan, as the agent asks for a task list when run.

This mode is a compromise between the other modes, with the advantages of the autonomous mode in term of resources usage, and the advantages of the managed mode in term of simplicity and flexibility. Its purpose can get summarized as: everything the GLPI server wants, but only when the agent wants.

Note: As a counter-part, if the system scheduling is planned too often, this may involve an overloading on the GLPI server if a lot of GLPI agents starts to submit requests at the same time. A way to avoid this inconvenient is to enable the *lazy configuration* so the GLPI server still decide the time before which the agent doesn't have to run tasks.

```
$ glpi-agent --lazy --server http://glpi/front/inventory.php
```

See also *Concurrent executions* to use `--wait` option.

3.2.3 Autonomous mode

This mode requires a local triggering mechanism to launch the agent, as the half-managed mode. It also has the same benefits for memory usage and reduced security concerns. However, the agent only executes a fixed list of tasks, and the server only receives the execution results, without any control. As sending those results may be done separately, this mode may also be used offline. This is achieved by using specific task-dedicated launchers, instead of the GLPI Agent one.

Deferred upload example:

```
$ glpi-inventory --json > inventory.json
$ glpi-injector --file inventory.json --url http://glpi/front/inventory.php
```

Immediate upload example:

```
$ glpi-inventory | curl --data @- http://glpi/front/inventory.php
```

This mode is the most complex to set-up, as you have to script the execution of multiple programs, this is not just a matter of configuration. It is also restricted to a limited set of agent tasks, for which a dedicated launcher exists (currently: local inventory, network discovery, network inventory). However, you have a full local control of agent execution.

If you don't trust the GLPI server for any reason (for instance, because it is run by another organization), of if your use case is just to report an inventory regularly, this mode is perfectly suited. It can get summarized as: only what the agent wants, only when the agent wants.

3.3 Offline usage

3.3.1 Agent execution

Most tasks handled by the agent can be executed directly without server, when it is not available, or for easier troubleshooting.

Most tasks have a dedicated launcher for this purpose. For instance, to execute a local inventory:

```
$ glpi-inventory
```

See man pages for details.

3.3.2 Result import

GLPI Interface

Go to the Administration > Inventory menu, choose the Import tab and upload the inventory file.

glpi-injector

The agent also has a dedicated executable for result injection:

```
$ glpi-injector --file inventory.json --url http://glpi/front/inventory.php
```

See glpi-injector man page for details.

curl

You can also use curl to push an inventory. This can be useful if your Perl installation has no SSL support, for instance:

```
$ curl --header "Content-Type: Application/x-compress" --cacert your-ca.pem -u username:password --data
```

With no SSL check and no authentication:

```
$ curl --header "Content-Type: Application/x-compress" -k --data @/tmp/inventory-file.json https://glpi/
```

3.4 Usage caution

3.4.1 Concurrent executions

When using managed mode, the server automatically distributes agent executions over time, using random scheduling. However, with other modes, the server doesn't have any control of agent execution schedule, and if they all try to communicate with it simultaneously, for instance because of a cron task executed on all hosts at the same time, the server may get flooded, and become unable to manage the load.

In order to avoid the issue, either distribute automated task execution over time, or use `--wait` command-line option for glpi-agent executable, introducing a random delay before effective execution. For instance:

```
# execute agent daily at random time between 0h00 and 0h30
0 0 * * * /usr/bin/glpi-agent --wait=1800
```

3.4.2 Multiple execution targets

Using multiple execution targets (servers or local directories) doesn't mean "execute once, upload the result multiple times", but "tries to execute every available task once for each given target". As a result, there is no guarantee that running an inventory for two different servers, or for one server and for local directory, will produce the exact same result.

The only reliable way currently to produce a single inventory, and transmit the result to multiple targets, is to execute the agent in autonomous mode once, and then upload the results multiple times:

```
$ glpi-inventory --json > inventory.json
$ glpi-injector --file inventory.json --url http://my.first.glpi/front/inventory.php
$ glpi-injector --file inventory.json --url http://my.second.glpi/front/inventory.php
```

CHAPTER

FOUR

TASKS

4.1 Remote inventory

GLPI Agent supports to run computer inventory remotely, this feature can also be referenced as **Agent-less inventory** for targeted computers.

4.1.1 Overview

This task can inventory remote computers via:

- **ssh**: for unix/linux computers
- **WinRM**: for win32 computers

Pre-requisite

To remotely inventory unix/linux computers supporting ssh, GLPI Agent needs to make network requests on ssh port. The remote ssh user **must** have administration privileges.

For windows computers, WinRM **must** be enabled on targeted computers. You can follow Microsoft official documentation to enable WinRM. But the short way to enable it with minimal security is to run from an administion console:

```
C:                                  > winrm quickconfig
C:                     > winrm set winrm/config/service/auth @Basic="true"
C:                  > winrm set winrm/config/service @AllowUnencrypted="true"
```

You may probably need to also create a dedicated user with administrative privileges and enable the `windows remote management` firewall rule.

For WinRM, GLPI Agent must be able to make network http requests on WinRM dedicated ports, by default 5985 for HTTP and 5986 for HTTPS. The remote WinRM user **must** have administration privileges.

Hint: **WinRM** remote inventory can be run from an agent running on unix/linux platform.

4.1.2 Setup

Remote computers will be known as **remotes** on GLPI agent side.

Remote registration

So before running inventory, you'll have to register in GLPI Agent environment remote computers with dedicated credentials. This step can be done by *glpi-remote* script using the **add** sub-command.

This is as simple as running:

```
$ glpi-remote add ssh://admin:pass@192.168.43.237
```

or

```
$ glpi-remote add winrm://admin:pass@192.168.48.250 --target server0
```

Note: When add a remote supporting WinRM, the agent will test provided credential and will fail if something goes wrong. You can avoid this check using `-C` or `--no-check` option.

Managing remotes

After remotes has been registered, you can list them with the following command:

```
$ glpi-remote list
```

This will provides the locally known remotes:

```
index  deviceid                url                                        target   Next␣
↪run date
    1  WIN-2020-09-23-15-37-52  winrm://glpi-agent:****@192.168.100.138  server0  Tue␣
↪Nov  9 15:46:51 2021
    2  XPS-2021-11-10-15-10-16  winrm://glpi-agent:****@192.168.100.139  server0  on␣
↪next agent run
```

You can delete a remote giving its listing index:

```
$ glpi-remote delete 1
```

You can update credential by simply *register again* the remote as the script will recognize your are updating an existing **remote**.

Attention: As of this writing, no solution has still been implemented in GLPI to manage remotes. So everything has to be done from the console.

4.1.3 Running

Automatic execution

When run as a service or a daemon and once remotes are registered against GLPI Agent and associated to a target, the agent will run RemoteInventory task when expected, generate an inventory and submit it to the related server or store it to a local path.

The selected target must be a known target:

- if selected target is `server0`, `server` must be defined in configuration,
- if selected target is `server1`, `server` must be defined with at least 2 URLs as `server1` means to use the second URL,
- if selected target is `local0`, `local` must be set in configuration with an existing path.

Command-line execution

When GLPI Agent is run from the commandline, it will try to run RemoteInventory task if at least one **remote** is known. It will then select one **remote** and only one to run an inventory but only if its `Next run date` has been set to `on next agent run`.

You can try to run only RemoteInventory task with:

```
$ glpi-agent --logger=stderr --tasks remoteinventory
```

You may have to run again the agent if another **remote** is expected to be inventoried. Just run `glpi-remote list` to verify if a **remote** has to be inventoried.

4.2 Network discovery

A network discovery task aims to scan the the network and reports devices found to the GLPI server, so they can be added to the list of known assets.

Once part of the list of known assets, further information can be retrieved from SNMP enabled devices using *Network inventory*.

4.2.1 Overview

This task uses the following protocols to scan IP networks:

- Arp local table request with `arp -a` command or `ip neighbor show` command
- ICMP Echo (aka ping) with fallback on ICMP Timestamp scan[1]
- NetBIOS scan (if `Net::NBName` is available and proper credits provided)
- SNMP scan (if `Net::SNMP` is available, and proper credits provided)

[1] For ICMP Echo & ICMP timestamp definition, see RFC 792:
– ICMP Echo messages have type 8 for requests and 0 for answers.
– ICMP Timestamp messages have type 13 for requests and 14 for answers.

Any device replying to at least one of those protocols will be *discovered*, with minimal information, such as mac address and hostname.

Additionally, if the device replies to SNMP, the agent will attempt to identify it using various methods.

The primary method relies on retrieving the value of the dedicated SNMP variable (`SNMPv2-MIB::sysObjectID.0`), which is a constructor-specific OID identifying a given device model, and comparing it to the agent internal database (the `sysobject.ids` file, described in *SNMP device IDS database*).

If a match is found, model, type and manufacturer are added to the information reported to the GLPI server, allowing simple identification. If no match is found, various heuristics are performed in order to identify the device, with lower reliability.

A secondary method relies on GLPI Agent `MibSupport` feature which permits to implement dedicated perl module for any kind of device. As an example, when the sysObjectID is reported as **linux** with the `8072.3.2.10` model-specific suffix, the `LinuxAppliance` **MibSupport** module usage is triggered and permits to inventory Synology or Ubiquiti devices.

Discovered devices are then reported to the GLPI servers, where import rules are applied. Devices not matching any import criteria will be kept in a server list of ignored devices.

4.2.2 Running

Pre-requisite

The agent performing the task needs to have the **netdiscovery** module installed.

The agent performing the task needs network access the target networks, with forementioned protocols, as well as control access, for SNMP: just being able to send UDP packets to a device is not enough, if this device is configured to ignore them. It is even best to use an agent from the same network than the scanned devices so it can access the local system arp table.

As for any other server controlled task, the agent should use either managed or half-managed mode, as explained in *Execution modes*. If the task is server-triggered, the agent must run in managed mode, and its HTTP port should be reachable from the server.

Command-line execution

A network discovery task can be also performed without a GLPI server, allowing easier control and troubleshooting, with the `glpi-netdiscovery` command-line tool.

However, this command generates files which will have to be injected in GLPI server using `glpi-injector` command.

Efficiency concerns

Credentials

Unfortunately, there is no way to distinguish a failed SNMP authentication attempt on a device from the absence of a device. It means the agent will try each available credential against each IP address, in given order, and wait for timeout each time. The most efficient way to address this issue if to only use the relevant set of credentials, and reduce the specific SNMP timeout.

Threads number

In order to scan multiple addresses simultaneously, the agent can use multiple discovery threads. This allow multiple simultaneous request, but also increase resource usage on agent host.

4.2.3 Troubleshooting

1. **The task doesn't run at all**
   a) The agent may be lacking the NetDiscovery module: run `glpi-agent --list-tasks` to check installed modules.
   b) There may be a server/agent communication issue: check you can reach the agent port (62354 by default) from the server host.
   c) The agent may be ignoring server requests, due to a a trust issue: check the agent logs for `[http server] invalid request (untrusted address)` message.
2. **The task runs, but agent logs show that SNMP is not used**
   a) The agent may be lacking the required Net::SNMP perl module: run `perl -MNet::SNMP` on agent host to check, it should blocks.
   b) There may be no SNMP credentials associated to the network scanned.
3. **The task runs, but no devices get added to my inventory**

   The reported items are insufficiently identified to be imported, according to your current import rules, check the list of ignored devices and the list of import rules on server side.
4. **The task runs, but my SNMP devices are not properly identified**

   The agent probably lacks the device SNMP identifier in its internal database.

   Use `glpi-netdiscovery` executable with debug option on the device, get the value from its output, and add it to the `sysobject.ids` file, as described in *SNMP device IDS database* to fix the issue.

   ```
   $ glpi-netdiscovery --first 192.168.0.1 --last 192.168.0.1 --credentials version:2c,community:publi
   ```

   Output:

   ```
   ...
   [debug] partial match for sysobjectID .1.3.6.1.4.1.311.1.1.3.1.1 in database:␣
   ↪unknown device ID
                                                  ^^^^^^^^^^^^^
   ```
5. **Agent crashes**

   This is likely to be a TLS multithreading issue. They are multiple ways to reduce the probability of such crash:
   a) make sure you only have one TLS perl stack installed on the agent host, preferably `IO::Socket::SSL + Net::SSLeay`. Having both stacks at once (`IO::Socket::SSL + Net::SSLeay` vs `Net::SSL + Crypt::SSLeay`) usually leads to unexpected results, even without thread usage
   b) use latest upstream release of `IO::Socket::SSL`, even if your distribution doesn't provide it
   c) reduce threads number during network discovery tasks

   However, the only actual solution currently is to disable SSL completely, using plain HTTP for agent/server communication. If the agent run on server host, that's usually not really a problem.

4.3 Network inventory

A network inventory task aims to retrieve exhaustive information from SNMP compatible devices, such as network devices or printers, already part of the list of known assets.

This task can only be performed on devices already part of the list of known assets, either as a result of a previous *Network discovery task*, or manually created, with proper SNMP credentials.

4.3.1 Overview

This task uses SNMP to retrieve various information from a device, so as to update it in GLPI:

- consumable levels and print counter on printers
- virtual LANs definition, network topology, network ports status on network devices

4.3.2 Running

Pre-requisite

The agent performing the task needs to have the NetInventory module installed.

The agent performing the task needs network access the target devices, with proper access control: just being able to send UDP packets to a device is not enough, if this device is configured to ignore them.

The target device should be associated with proper SNMP credentials in GLPI.

As for any other server controlled task, the agent should use either managed or half-managed mode, as explained in *Execution modes*. If the task is server-triggered, the agent must run in managed mode, and its HTTP port should be reachable from the server.

Command-line execution

A network inventory task can be also performed without a GLPI server, allowing easier control and troubleshooting, with the `glpi-netinventory` command-line tool.

However, this command generates files which will have to be injected in GLPI server using `glpi-injector` command.

4.3.3 Troubleshooting

See *NetDiscovery troubleshooting*.

CHAPTER

FIVE

PLUGINS

5.1 SSL Server Plugin

5.1.1 Context

When run as a daemon or a service, by default, GLPI Agent can be reached on an HTTP interface. By default, GLPI Agent listens on port 62354, but this can be disabled by setting *no-httpd configuration* or changed to listen on a different port using *httpd-port configuration*.

By default, the agent embedded HTTP interface doesn't support message encryption using SSL.

5.1.2 Purpose

The purpose of this plugin is to enable SSL on the embedded HTTP interface to secure all exchanges with external clients.

5.1.3 Setup

By default, this plugin is disabled. The first step is to enable it creating a dedicated configuration:

1. Locate the `ssl-server-plugin.cfg` file under the GLPI agent *configuration folder*[1],
2. Make a copy of this file in the same folder by just changing the file extension from `.cfg` to `.local`.
3. Edit the `ssl-server-plugin.local` and set `disabled` to `no`

This way, the agent will start to only accept client supporting SSL. For instance, if you accessed before the agent interface on local machine using `http://127.0.0.1:62354`, you'll now have to use `https://127.0.0.1:62354`.

5.1.4 Configuration

The default configuration is self-explanatory:

```
# By default, a plugin is always disabled unless "disabled" is set to "no" or "0".
# You can uncomment the following line or set it in included configuration file
# at the end of this configuration
#disabled = no

# Comma separated list of ports like in: ports = 62355,62356
```

(continues on next page)

[1] On windows, the configuration is also a file and it located under the `etc` sub-folder of the GLPI Agent installation folder.

(continued from previous page)

```
#ports = 0

# Example command to generate key/certificate files pair
# openssl req -x509 -newkey rsa:2048 -keyout etc/key.pem -out etc/cert.pem -days 3650 -
↪sha256 -nodes -subj "/CN=127.0.0.1"
#ssl_cert_file = cert.pem
#ssl_key_file  = key.pem

# You should create and define you specific parameter in the following
# included configuration file to override any default.
# For example just set "disabled = no" in it to enable the plugin
include "ssl-server-plugin.local"
```

disabled Can be set to "no" to enable the plugin. (By default: yes)

ports Can be set to a list of ports on which you need to enable SSL support. (By default: 0)

You can for example keep simple http support on the default port and just enable SSL on the port used by one or more agent server plugins.

ssl_cert_file The path to the server certificate to use with SSL support. (By default: not defined)

The path can be relative to the configuration folder or an absolute path.

ssl_key_file The path to the server private key certificate to use with SSL support. (By default: not defined)

The path can be relative to the configuration folder or an absolute path. This path should be a secured location, not readable by simple local system users.

5.2 Proxy Server Plugin

5.2.1 Context

When run as a daemon or a service, by default, GLPI Agent can be reached on an HTTP interface. By default, GLPI Agent listens on port 62354, but this can be disabled by setting *no-httpd configuration* or changed to listen on a different port using *httpd-port configuration*.

5.2.2 Purpose

The purpose of this plugin is to enable a proxy mode on the embedded HTTP interface.

It can replace a proxy pass configuration on a http server or a complex proxy setup. It offers few other advantages and features that can be used in advanced setup.

5.2.3 Setup

By default, this plugin is disabled. The first step is to enable it creating a dedicated configuration:

1. Locate the `proxy-server-plugin.cfg` file under the GLPI agent *configuration folder*[1],
2. Make a copy of this file in the same folder by just changing the file extension from `.cfg` to `.local`.
3. Edit the `proxy-server-plugin.local` and set `disabled` to `no`

This way, the agent will start to accept inventory submission on its current port and on `/proxy/glpi` as base url. If a GLPI server is setup, any submitted inventory will be forwarded to that GLPI server.

Note: You can also enable a **secondary proxy** by creating the `proxy2-server-plugin.local` configuration. The purpose here is to enable another proxy configuration on the same agent, mostly in the case you need it on a dedicated port with eventually SSL support plugin enabled.

5.2.4 Configuration

The default configuration is self-explanatory:

```
# By default, a plugin is always disabled unless "disabled" is set to "no" or "0".
# You can uncomment the following line or set it in included configuration file
# at the end of this configuration
#disabled = no

# Set base url matching for API
#url_path = /proxy
# Note: the server URL to set on client would have to be http[s]://[host]:[port][url_
↪path]/glpi
# By default, this should be: http://[agent-ip-or-dns]:62354/proxy/glpi

# Port on which to listen for inventory requests, default to legacy port
#port = 62354

# The delay the proxy should return as contact timeout to agents (in hours)
#prolog_freq = 24

# Option to handle proxy local storing. Set a folder full path as local_store to
# also store received XML locally. Set only_local_store to not immediatly send
# received XML to known server(s).
#only_local_store = no
#local_store =

# To limit any abuse we expect a maximum of 30 requests by hour and by ip (3600 seconds)
# You can adjust the rate limitation by updating the requests number limit
# or the period on which apply the requests number limit
#maxrate        = 30
#maxrate_period = 3600

# The maximum number of forked handled request
```

(continues on next page)

[1] on windows the configuration is also a file under the `etc` sub-folder of the GLPI Agent installation folder.

(continued from previous page)

```
#max_proxy_threads = 10

# The maximum number of proxy a request can pass-through
#max_pass_through = 5

# By default, if a GLPI server is set, we consider it supports GLPI protocol
# otherwise this proxy should only support legacy XML based protocol
#glpi_protocol = yes

# no-category config returned to agent when using CONTACT protocol without GLPI server
# or if only_local_store is set to yes
# no_category =
# Example: no_category = process,environment

# You should create and define you specific parameter in the following
# included configuration file to override any default.
# For example just set "disabled = no" in it to enable the plugin
include "proxy-server-plugin.local"
```

disabled Can be set to "no" to enable the plugin. (By default: yes)

port Can be set to a port on which the agent will listen too. (By default: 0, meaning use agent port)

You can dedicate a port for the proxy usage. You can even enable the *SSL Server Plugin*, and set the port in its `ports` list to force using SSL with the proxy mode.

prolog_freq In the case the proxy agent is not in touch with a GLPI server, this is the delay time in hours an inventory should be sent by contacting agents. (By default: 24)

only_local_store Set to `yes` to only store locally submitted inventories. (By default: no)

This can be handy if the only purpose is to collect inventories and no GLPI server is reachable. Stored inventories could then be passed later to `glpi-injector`.

local_store This is a full path where to store submitted inventories. If set, the proxy agent will always stored submitted inventories. (By default: empty)

You must manage by yourself the stored inventories or you may face a storage outage after a while if many agents submits inventories. But as inventories are stored with the deviceid as file basename, new inventory for a known agent will just replace the existing one.

This storage may be used as an inventory backup solution but keep in mind this storage should be keep secured as it will contains a lot of sensible datas.

maxrate and maxrate_period Limit requests for a given ip to **maxrate** other the **maxrate_period** time (in seconds). (By default: **30** requests by **3600** seconds for a single ip)

This 2 parameters could be used to limit abuse if the agent proxy if listening on a public network.

By default the average request for a given ip should be lower than 2. But this can be greater if the other ip is a chained proxy. In that last case, you may need to grow that values.

max_proxy_threads This set the maximum number of single requests the proxy agent can handle at the same time. (By default: 10)

max_pass_through This set the maximum number of proxy agents a single inventory submission can pass. (By default: 5)

Each time a inventory is submitted, a HTTP header value is set or increased. If that value reachs the `max_pass_through` of a proxy agent, the inventory won't be submitted to the next proxy agent.

Changing this parameter is only needed when you're using a chained proxy agents configuration and when you have at least 5 proxy agents in the chain. This parameter is a security to block loops in the case of chained proxy agents misconfiguration pointing to each others.

glpi_protocol Set to "no" if you don't want to use new GLPI Agent Protocol needed for GLPI native inventory. (By default: yes)

When set to "no", the proxy agent will just act as a legacy GLPI server supporting XML inventory format only. Otherwise, it will tell remote agent to use new protocol which involves to send inventory in JSON format.

no_category This is a comma separated list of category to not include in inventories. It has the same purpose than *no-category configuration* but set on server-side. This only works with new GLPI Agent Protocol and JSON format.

5.2.5 Use cases

Private network inventory storage

In the case, you have a private network from where no device can access the GLPI server, you can:

First, store submitted inventory using a proxy agent with such `proxy-server-plugin.local` configuration:

```
disabled = no
only_local_store = yes
local_store = /var/glpi-agent/proxy
```

Then, from GLPI server or a dedicated platform with GLPI server access possible, get a copy of stored inventories into a dedicated folder:

- using a command like `scp`, `ftp` or `rsync`
- using an USB key

Finally, inject inventories into GLPI with `glpi-injector` script:

```
$ glpi-injector -d /var/glpi-agent -r -R -u http://glpi-server/front/inventory.php
```

Proxy with HTTP and HTTPS support

In the case you need to secure a private network with eventually old FusionInventory agent not supporting SSL, but you still need SSL for newer GLPI Agents, you can create a proxy agent listening on port 80 and port 443.

First, enable main proxy mode on port 443 with the following `proxy-server-plugin.local` configuration:

```
disabled = no
port = 443
```

Secondly, enable the secondary proxy mode on port 80 with the following `proxy2-server-plugin.local` configuration:

```
disabled = no
port = 80
```

Also enable the *SSL Server Plugin* with the following `ssl-server-plugin.local` configuration:

```
disabled = no
ports = 443
```

Now, in agent configuration having access to the proxy agent, you can use any one of the 2 following URL as *server configuration*:

```
server = http://proxy-agent/proxy/glpi
server = https://proxy-agent/proxy/glpi
```

Here you don't have to specify the port as standard http and https ports are used. The only requirement for the proxy agent is to run on a dedicated server with that ports not used by any other service.

Chained proxies

Imagine you want to inventory the devices from one factory of your company in a given town having its dedicated and private network. This factory network is only visible via a vpn at the town level through a network being able to see other factories in the same town. Now your GLPI server is located in your head quarter in another town linked through a dedicated network link.

You can first create a proxy agent at the factory level just by enabling the proxy plugin on one computer, let'say the one with `10.77.200.55` ip. You just need to enable the plugin with the following `proxy-server-plugin.local` configuration:

```
disabled = no
```

Do the same on other agent where you need the plugin, the one at the town level, let's say it has the `10.77.0.2` ip. And do also the same with the agent in the head quarter network with let's say the `10.1.0.120` ip.

On all other devices in the factory, setup agents with the following URL as *server configuration*:

```
server = http://10.77.200.55:62354/proxy/glpi
```

On the agent with the proxy plugin enabled, set the server URL to the proxy agent enabled at the town level:

```
server = http://10.77.0.2:62354/proxy/glpi
```

On the agent at the town level with the proxy plugin enabled, set the server URL to the proxy agent enabled at the head quarter level:

```
server = http://10.1.0.120:62354/proxy/glpi
```

On the agent at the head quarter level, just set the normal GLPI server URL as *server configuration*.

Then you just have to secure your network to permit each proxy agent to communicate on port 62354 with its chained one.

Note: Remember only 5 proxy agents can be chained by default (see *max_pass_through parameter*). If you want to chain more proxy agents, set this parameter value to a higher value starting from the 6th proxy agent.

5.3 Inventory Server Plugin

5.3.1 Context

When run as a daemon or a service, by default, GLPI Agent can be reached on an HTTP interface. By default, GLPI Agent listens on port 62354, but this can be disabled by setting *no-httpd configuration* or changed to listen on a different port using *httpd-port configuration*.

5.3.2 Purpose

This plugin purpose is to permit requesting an inventory from a trusted remote computer knowing a shared secret. This is handy in the case the agent has no way to contact the GLPI server by itself.

5.3.3 Setup

By default, this plugin is disabled. The first step is to enable it creating a dedicated configuration:

1. Locate the `inventory-server-plugin.cfg` file under the GLPI agent *configuration folder*[1],
2. Make a copy of this file in the same folder by just changing the file extension from `.cfg` to `.local`.
3. Edit the `inventory-server-plugin.local` and set `disabled` to `no`
4. Also set a private token setting the `token` parameter to any strong secret

This way, the agent will start to accept inventory requets from a computer knowing the shared secret. To make such inventory requests, you'll just have to use the **agent** sub-command of *glpi-remote* script.

Hint: You can also enable the *SSL Server Plugin* to also encrypt the communication.

Note: This HTTP server plugin feature as nothing to do with the glpi-agent remote inventory feature as here, you still need to install a GLPI Agent on the targeted systems. With the glpi-agent remote inventory feature, you don't need any agent on the targeted systems.

5.3.4 Configuration

The default configuration is self-explanatory:

```
# By default, a plugin is always disabled unless "disabled" is set to "no" or "0".
# You can uncomment the following line or set it in included configuration file
# at the end of this configuration
#disabled = no

# Set base url matching for API
#url_path = /inventory

# Port on which to listen for inventory requests, default to legacy port
#port = 62354
```

(continues on next page)

[1] on windows the configuration is also a file under the `etc` sub-folder of the GLPI Agent installation folder.

(continued from previous page)

```
# token as private secret used to verify authorization token, not defined by
# default, but mandatory to access API
#token = [secret-string]

# A timeout for session to no more trust a client payload
#session_timeout = 60

# Set this to 'yes' if XML compression is not required
#no_compress = no

# To limit any abuse we expect a maximum of 50 requests by hour (3600 seconds)
# You can adjust the rate limitation by updating the requests number limit
# or the period on which apply the requests number limit
#maxrate        = 30
#maxrate_period = 3600

# You should create and define you specific parameter in the following
# included configuration file to override any default.
# For example just set "disabled = no" in it to enable the plugin
include "inventory-server-plugin.local"
```

disabled Can be set to "no" to enable the plugin. (By default: yes)

url_path The path to the server certificate to use with SSL support. (By default: **/inventory**)

The path can be relative to the configuration folder or an absolute path.

port Can be set to a port on which the agent will listen too. (By default: **0**, meaning use agent port)

You can dedicate a port for the inventory usage. You can even enable the *SSL Server Plugin*, and set the port in its `ports` list to force using SSL with the inventory plugin.

token **MUST** be set to a strong secret or no inventory will be generated. (By default: not defined)

session_timeout The session timeout is a time in seconds and defines the maximum time the agent will wait for the remote client to authentify itself with the shared secret. (By default: **60**)

no_compress Can be set to **yes** to avoid inventory compression when sent back. (By default: **no**)

maxrate and maxrate_period Limit requests for a given ip to **maxrate** other the **maxrate_period** time (in seconds). (By default: **30** requests by **3600** seconds for a single ip)

This 2 parameters could be used to limit even more any brute force attack attempt.

5.3.5 Inventory request

Inventory request have to be done using **agent** sub-command of the `glpi-remote` script.

See the *glpi-remote* dedicated man page for all possible options.

5.3.6 Use cases

DMZ server inventory

In the case you have a server in DMZ which cannot access the GLPI server, but the GLPI server is authorized to reach it. You still can install an agent on it.

Then first, enable the plugin with such `inventory-server-plugin.local` configuration:

```
disabled = no
token = 5c9898f9-e619-4bdb-8e29-6a20766ab760
```

In the agent conf, don't set `server` nor `local` but set `listen` to `yes` and set `httpd-trust` with the GLPI server one. For example create the `/etc/glpi-agent/conf.d/local.cfg` file with:

```
listen = yes
httpd-trust = <glpi-server-ip>
```

On the GLPI server, create a script you would want to put in `/etc/cron.daily`:

```
#!/bin/bash
sleep $((RANDOM/100))
glpi-remote -T 5c9898f9-e619-4bdb-8e29-6a20766ab760 agent <dmz-server-ip> | \
   glpi-injector -url http://127.0.0.1/front/inventory.php >/var/tmp/server-inventory.
↪log 2>&1
```

Adapt this shell script to your needs.

Internet server

In the case you have an internet server hosted anywhere and you want to inventory it in your GLPI being in your intranet.

Make sure server and intranet firewalls will permits communications between them, the GLPI server being the HTTP client and let's say via the `54443` port.

Then first, after installed the agent on the internet server, enable the plugin with such `inventory-server-plugin.local` configuration:

```
disabled = no
token = 2b0a48a2-6eb1-4e8f-bf8c-41f461b58ef1
base = /2cd3a12ac1c4
port = 54443
```

Also enable the *SSL Server Plugin* with such `ssl-server-plugin.local` configuration:

```
disabled = no
ports = 54443
```

In the agent conf, don't set `server` nor `local` but set `listen` to `yes` and set `httpd-trust` with your intranet public one. For example create the `/etc/glpi-agent/conf.d/glpi.cfg` file with:

```
listen = yes
httpd-trust = <intranet-public-ip>
```

On the GLPI server, create a script you would want to put in `/etc/cron.daily`:

```
#!/bin/bash
sleep $((RANDOM/100))
glpi-remote -T 2b0a48a2-6eb1-4e8f-bf8c-41f461b58ef1 -p 54443 --ssl --no-ssl-check -b /
↪2cd3a12ac1c4 agent <internet-server-ip> | \
   glpi-injector -url http://127.0.0.1/front/inventory.php >/var/tmp/internet-server-
↪inventory.log 2>&1
```

Adapt this shell script to your needs.

5.4 Test Server Plugin

5.4.1 Context

When run as a daemon or a service, by default, GLPI Agent can be reached on an HTTP interface. By default, GLPI Agent listens on port 62354, but this can be disabled by setting *no-httpd configuration* or changed to listen on a different port using *httpd-port configuration*.

5.4.2 Purpose

This plugin has indeed no functional purpose but it can be used as a template if you need to develop your own GLPI Agent HTTP server plugin.

This plugin when enabled only log the request and just answer to it with a HTTP 200 return code if its URL matched the `^/test/([\w\d/-]+)?$` regular expression.

See the code for more details.

5.4.3 Setup

By default, this plugin is disabled. The first step is to enable it creating a dedicated configuration:

1. Locate the `server-test-plugin.cfg` file under the GLPI agent *configuration folder*[1],
2. Make a copy of this file in the same folder by just changing the file extension from `.cfg` to `.local`.
3. Edit the `server-test-plugin.local` and set `disabled` to `no`

5.4.4 Configuration

The default configuration is:

```
# By default, a plugin is always disabled unless "disabled" is set to "no" or "0".
# You can uncomment the following line or set it in included configuration file
# at the end of this configuration
#disabled = no

#configtest = test

#port = 62355
```

(continues on next page)

[1] on windows the configuration is also a file under the `etc` sub-folder of the GLPI Agent installation folder.

(continued from previous page)

```
# You should create and define you specific parameter in the following
# included configuration file to override any default.
# For example just set "disabled = no" in it to enable the plugin
include "server-test-plugin.local"
```

disabled Can be set to "no" to enable the plugin. (By default: yes)

port Can be set to a port on which the agent will listen too. (By default: 0, meaning use agent port)

You can dedicate a port and even enable the *SSL Server Plugin* setting this port in its `ports` list to force using SSL with this Test plugin.

configtest Just a value that would be logged with the request log line.

5.5 ToolBox

5.5.1 Context

When run as a daemon or a service, by default, GLPI Agent can be reached on an HTTP interface. By default, GLPI Agent listens on port 62354, but this can be disabled by setting *no-httpd configuration* or changed to listen on a different port using *httpd-port configuration*.

5.5.2 Purpose

TO BE DOCUMENTED

CHAPTER

SIX

IDS DATABASES

The agent integrates IDS databases which are located in agent data directory. It consists of the following files:

- `pci.ids` is a database of PCI devices, used by local inventory module,
- `usb.ids` is a database of USB devices, used by local inventory module,
- `edid.ids` is a database of Screen manufacturer, used by local inventory module,
- `sysobject.ids` is a database of SNMP devices, used by network discovery and network inventory tasks modules.

Those files can easily be customized if needed, as their format is self-documented. However, local modifications will get lost on upgrade.

6.1 SNMP device IDS database

The `sysobject.ids` file is a database of known SNMP devices, indexed by the discriminant part of their sysObjectID[1] value:

```
9.1.1111    Cisco   NETWORKING      Catalyst 3500
+           +       +               +
|           |       |               |
|           |       |               +-> device model
|           |       |
|           |       +-----------------> device type
|           |
|           +-------------------------> device manufacturer
|
+-----------------------------------> sysObjectID model-specific suffix
```

The **sysObjectID model-specific suffix** is the last part of the full sysObjectID value, ie:

```
.1.3.6.1.4.1.9.1.111
+            +
|            |
|            +-----> model-specific suffix
|
+------------------> shared prefix
```

[1] See sysObjectID definition in RFC 3418:
sysObjectID is the vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining *what kind of box* is being managed.

The sysObjectID value for any SNMP device can be retrieved by any SNMP client, using its OID (`.1.3.6.1.2.1.1.2.0`), or with either *Network inventory* or *Network discovery* command-line tools, with `--debug` flag.

Hint: You can contribute your modifications on the related github sysobject.ids project.

CHAPTER

SEVEN

BUG REPORTING

Attention: Problems related to this documentation should be reported on its dedicated project issue tracker. You can also just edit the project in your own github environment and submit a pull request.

For any GLPI agent problem or issue, you need to use GitHub issue tracker, and report your issue with all the requested informations. This requires an GitHub user account.

Hint: Before reporting your problem, always take a look at the opened and closed GitHub issues by updating the *Filters* field and check if your problem has still not been addressed.

7.1 Problem description

You have to use english, for the sake of internationalization.

Always describe your problem clearly and precisely, for someone without any prior knowledge of your environment.

To do so, always describe:

- the operating system on which the GLPI agent is running,
- the GLPI agent version you're using and the way you installed it,
- the problem itself, eventually including log extract or agent run output,
- what you're expecting

Hint: You can obtain a more verbose output when running the agent from the console by using `--debug` option to set debug level 1, or by using `--debug --debug` options to set debug level 2. You can also set the *debug parameter* in configuration to `1` or `2` and restart the daemon or service.

7.2 External data

When the issue is related to some missing or invalid value, it can be caused by a parsing error while processing external data (a command output, a file, whatever...). In order to reproduce the problem and find a way to fix it, we need a sample of those data.

You can then attach the relevant file or command output as an attachement to the issue report. If there is any privacy concern, try to obfuscate sensible data or ask for an email where to send it.

On unix/linux, when running any command for such purpose, you always must unset locales first to avoid localized output:

```
$ export LC_ALL=C
```

Here are other command-specific advices:

7.2.1 WMI queries

Windows WMI queries can be exported with `wmic` tool:

```
C:                > wmic path <somequery> get /Format:list > <somefile.txt>
```

7.2.2 Registry extract

Windows registry content can be exported with `regedit` tool:

```
C:                           > regedit /e <somefile.reg> <somekey>
```

7.2.3 dmidedoce output

`dmidecode` output can be generated on any system, including windows, as we ship a `dmidecode` executable in the agent windows packaging, under the `perl\bin` subdirectory.

```
C:                      > "C: Files-Agent-agent" > dmidecode-output.txt
```

On unix/linux, just run the command in a `root` console:

```
$ dmidecode >dmidecode-output.txt
```

7.2.4 snmpwalk output

Snmpwalk output can be generated with the following command under unix/linux:

```
$ snmpwalk -v <version> -c <community> -t 15 -Cc -On -Ox <somehost> .1 > walk-filename.walk
```

Using an explicit root OID (.1 here), a non-default timeout (`15` seconds, the same one as the agent default), and disabling internal consistency checks (`-Cc`) are required to extract all possible data.

Option `-Ox` is not mandatory but can help to enhance debugging discovery and inventory tasks as we may not know anything about the related MIB. So just having full numeric values can help.

Option `-On` is required to keep OID numerically to be sure to have a fully compliant snmp walk.

Hint: Joining known private MIBs related to your device could be really useful. You always can share them privately if you don't have the right to make them public.

Hint: When reporting a snmp walk, list all possible expected data you can know by another way, and at least the missing ones, like:

- serial number
- accurate model name
- manufacturer
- device name
- device mac address
- ...

CHAPTER

EIGHT

MAN PAGES

8.1 glpi-agent

Hint: Configuration options are also documented on the *Configuration* page.

8.1.1 NAME

glpi-agent - GLPI perl agent For Linux/UNIX, Windows and MacOSX

8.1.2 SYNOPSIS

glpi-agent [options] [--server server|--local path]

```
Target definition options:
  -s --server=URI                send tasks result to a server
  -l --local=PATH                write tasks results locally

Target scheduling options:
  --delaytime=LIMIT              maximum delay before first target,
                                   in seconds (3600). It also defines the
                                   maximum delay on network error. Delay on
                                   network error starts from 60, is doubled at
                                   each new failed attempt until reaching max
  --lazy                         do not contact the target before
                                 next scheduled time
  --set-forcerun                 set persistent state 'forcerun' option so a run
                                 will be started immediately during a start or init

Task selection options:
  --list-tasks                   list available tasks and exit
  --no-task=TASK[,TASK]...       do not run given task
  --tasks=TASK1[,TASK]...[,...]  run given tasks in given order

Inventory task specific options:
  --no-category=CATEGORY         do not list given category items
  --list-categories              list supported categories
  --scan-homedirs                scan user home directories (false)
```

(continues on next page)

(continued from previous page)

```
  --scan-profiles                scan user profiles (false)
  --html                         save the inventory as HTML (false)
  --json                         save the inventory as JSON (false)
  -f --force                     always send data to server (false)
  --backend-collect-timeout=TIME timeout for inventory modules
                                   execution (30)
  --additional-content=FILE      additional inventory content file
  --partial=CATEGORY             make a partial inventory of given category
                                   items, this option implies --json
  --credentials                  set credentials to support database inventory

Package deployment task specific options:
  --no-p2p                       do not use peer to peer to download
                                   files (false)

Network options:
  -P --proxy=PROXY               proxy address
  -u --user=USER                 user name for server authentication
  -p --password=PASSWORD         password for server authentication
  --ca-cert-dir=DIRECTORY        CA certificates directory
  --ca-cert-file=FILE            CA certificates file
  --no-ssl-check                 do not check server SSL certificate
                                   (false)
  -C --no-compression            do not compress communication with server
                                   (false)
  --timeout=TIME                 connection timeout, in seconds (180)

Web interface options:
  --no-httpd                     disable embedded web server (false)
  --httpd-ip=IP                  network interface to listen to (all)
  --httpd-port=PORT              network port to listen to (62354)
  --httpd-trust=IP               trust requests without authentication
                                   token (false)
  --listen                       enable listener target if no local or
                                 server target is defined

Logging options:
  --logger=BACKEND               logger backend (stderr)
  --logfile=FILE                 log file
  --logfile-maxsize=SIZE         maximum size of the log file in MB (0)
  --logfacility=FACILITY         syslog facility (LOG_USER)
  --color                        use color in the console (false)

Configuration options:
  --config=BACKEND                   configuration backend
  --conf-file=FILE                   configuration file
  --conf-reload-interval=<SECONDS>   number of seconds between two
                                       configuration reloadings

Execution mode options:
  -w --wait=LIMIT                maximum delay before execution,
                                   in seconds
```

(continues on next page)

(continued from previous page)

```
  -d --daemon                    run the agent as a daemon (false)
  --no-fork                      don't fork in background (false)
  -t --tag=TAG                   add given tag to inventory results
  --debug                        debug mode (false)
  --setup                        print the agent setup directories
                                   and exit
  --vardir=PATH                  use specified path as storage folder for agent
                                   persistent datas

  --version                      print the version and exit
  --no-win32-ole-workaround      [win32 only] disable win32 work-around
                                   used to better handle Win32::OLE apis.
                                   !!! Use it at your own risk as you may
                                   experiment perl crash under win32 !!!
```

8.1.3 DESCRIPTION

The *glpi-agent* agent is a generic multi-platform agent. It can perform a large array of management tasks, such as local inventory, software deployment or network discovery. It can be used either standalone, or in combination with a compatible server acting as a centralized control point.

8.1.4 OPTIONS

Most of the options are available in a *short* form and a *long* form. For example, the two lines below are all equivalent:

```
% glpi-agent -s localhost
% glpi-agent --server localhost
```

Target definition options

-s, --server=URI Send the results of tasks execution to given server.

If *URI* doesn't start with http:// or https://, the agent assume the parameter is a hostname and rewrite it as:

```
% --server=http://example/front/inventory.php
```

In general, GLPI server URL have this format:

```
http://example/front/inventory.php
```

and FusionInventory for GLPI this one:

```
http://example/glpi/plugins/fusioninventory
```

Multiple values can be specified, using comma as a separator.

-l, --local=PATH Write the results of tasks execution locally.

Exact behaviour according to given path:

- if *PATH* is a directory, a file will be created therein
- if *PATH* is a file, it will be used directly

- if *PATH* is '-', STDOUT will be used

Multiple values can be specified, using comma as a separator.

Target scheduling options

--delaytime=LIMIT Set an initial delay before the first target, whose value is computed randomly between LIMIT / 2 and LIMIT seconds. This setting is ignored for server targets after the initial contact, in favor of server-specified parameter (PROLOG_FREQ).

--lazy Do not contact the target before next scheduled time.

This option is only available when the agent is not run as a server.

Task selection options

--list-tasks List all available tasks, tasks planned for execution and exit

--no-task=TASK Do not run given task.

Multiple values can be specified, using comma as a separator. See option *--list-tasks* for the list of available tasks.

--tasks=TASK Run given tasks in given order.

Multiple tasks can be specified, using comma as a separator. A task can be specified several times. if '...' is given as last element, all other available tasks are executed.

See option *--list-tasks* for the list of available tasks.

Examples :

* --tasks=inventory,deploy,inventory First task executed is 'inventory', second task is 'deploy', third and last task is 'inventory'.

* --tasks=inventory,deploy,... First executed task is 'inventory', second task is 'deploy' and then all other available tasks are executed.

Inventory task specific options

--no-category=CATEGORY Do not list given category items in inventory.

Multiple values can be specified, using comma as a separator. The available categories are:

- accesslog
- battery
- bios
- controller
- cpu
- database
- drive
- environment
- hardware
- input

- licenseinfo
- local_group
- local_user
- lvm
- memory
- modem
- monitor
- network
- os
- port
- printer
- process
- provider
- psu
- registry
- remote_mgmt
- rudder
- slot
- software
- sound
- storage
- usb
- user
- video
- virtualmachine

--list-categories List all supported categories by scanning all available inventory modules

--credentials=CREDENTIALS Setup credentials for database inventory

CREDENTIALS should be a list of "key:value" separated by commas like in: For example: --credentials="type:login_password,login:root,password:****,use:postgresql,params_id:0"

--scan-homedirs Allow the agent to scan home directories for virtual machines.

--scan-profiles Allow the agent to scan user profiles for software.

--html|--json Save the inventory as HTML or JSON.

This is only used for local inventories.

-f, --force Send an inventory to the server, even if this last one doesn't ask for it.

--backend-collect-timeout=TIME Timeout for inventory modules execution.

--additional-content=FILE Additional inventory content file.

This file should be an XML file, using same syntax as the one produced by the agent.

Package deployment task specific options

--no-p2p Do not use peer to peer to download files.

Server target specific options

-P, --proxy=PROXY Use *PROXY* as HTTP proxy.

By default, the agent uses HTTP_PROXY environment variable.

-u USER, --user=USER Use *USER* for server authentication.

-p, --password=PASSWORD Use *PASSWORD* for server authentication.

--ca-cert-dir=DIRECTORY CA certificates directory.

--ca-cert-file=FILE CA certificates file.

--no-ssl-check Do not check server SSL certificate.

--timeout=TIME Timeout for server connections.

Web interface options

--no-httpd Disable the embedded web server.

--httpd-ip=IP The network interface to use for the embedded web server (all).

--httpd-port=PORT The network port to use for the embedded web server (62354).

--httpd-trust=IP Trust requests from given addresses without authentication token (false).

For example: "192.168.0.0/24", "192.168.168.0.5" or an IP range like "20.34.101.207 - 201.3.9.99". Hostnames are also accepted. See Net::IP documentation to get more example.

Multiple values can be specified, using comma as a separator.

--listen This option should be used if no local or server target is defined and the agent still needs to answer http requests. **--no-httpd** should not be set and **--httpd-trust** should be set to enable trusted remote clients.

Logging options

--logger=BACKEND Logger backend to use.

Multiple values can be specified, using comma as a separator. The available backends are:

- stderr: log messages directly in the console.
- file: log messages in a file.
- syslog: log messages through the local syslog server.

Multiple values can be specified, using comma as a separator.

--logfile=FILE Log message in *FILE* (implies File logger backend).

--logfile-maxsize=SIZE Max logfile size in MB, default is unlimited. When the max size is reached, the file is truncated. This is only useful if there is no log rotation mechanism on the system.

--logfacility=FACILITY Syslog facility to use (default LOG_USER).

--color Display color on the terminal, when the Stderr backend is used.

This options is ignored on Windows.

Configuration options

--config=BACKEND Configuration backend to use.

The available backends are:

- file: read configuration from a file (default anywhere else as Windows).
- registry: read configuration from the registry (default on Windows).
- none: don't read any configuration.

--conf-file=FILE Use *FILE* as configuration file (implies file configuration backend).

--conf-reload-interval=SECONDS SECONDS is the number of seconds between two configuration reloadings. Default value is 0, which means that configuration is never reloaded. Minimum value is 60. If given value is less than this minimum, it is set to this minimum. If given value is less than 0, it is set to 0.

Execution mode options

-w LIMIT, --wait=LIMIT Wait a random delay whose value is computed randomly between 0 and LIMIT seconds, before execution. This is useful when execution is triggered from some kind of system scheduling on multiple clients, to spread the server load.

-d, --daemon Run the agent as a daemon.

--no-fork Don't fork in background.

This is only useful when running as a daemon.

--pidfile[=FILE] Store pid in *FILE* or in default PID file.

This is only useful when running as a daemon and still not managed with a system service manager like systemd.

--tag=TAG Add the given tag to every inventory results.

--debug Turn the debug mode on. You can use the parameter up to 3 times in a row to increase the verbosity (e.g: **--debug --debug**).

Level 3 turns on the debug mode of some external libraries like Net::SSLeay. These messages will only be be printed on STDERR.

--setup Print the agent setup directories and exit.

--version Print the version and exit.

8.2 glpi-inventory

8.2.1 NAME

glpi-inventory - Standalone inventory

8.2.2 SYNOPSIS

glpi-inventory [options]

```
Options:
  --scan-homedirs               scan use home directories (false)
  --scan-profiles               scan user profiles (false)
  --html                        save the inventory as HTML (false)
  --json                        save the inventory as JSON (false)
  --no-category=CATEGORY        do not list given category items
  --partial=CATEGORY            make a partial inventory of given category
                                  items, this option implies --json
  --credentials                 set credentials to support database inventory
  -t --tag=TAG                  mark the machine with given tag
  --backend-collect-timeout=TIME timeout for inventory modules
                                  execution (30)
  --additional-content=FILE     additional inventory content file
  --verbose                     verbose output (control messages)
  --debug                       debug output (execution traces)
  -h --help                     print this message and exit
  --version                     print the task version and exit
```

8.2.3 DESCRIPTION

glpi-inventory can be used to run an inventory task without a GLPI server.

8.3 glpi-netdiscovery

8.3.1 NAME

glpi-netdiscovery - Standalone network discovery

8.3.2 SYNOPSIS

glpi-netdiscovery [options] --first <address> --last <address>

```
Options:
  --host <ADDRESS>       Host IP address to scan or IP range first address
  --first <ADDRESS>      IP range first address
  --last <ADDRESS>       IP range last address
  --port <PORT[,PORT2]>  SNMP port (161)
  --protocol <PROT[,P2]> SNMP protocol/domain (udp/ipv4)
```

(continues on next page)

(continued from previous page)

```
--community <STRING>   SNMP community string (public)
--credentials <STRING> SNMP credentials (version:1,community:public)
--timeout <TIME        SNMP timeout, in seconds (1)
--entity <ENTITY>      GLPI entity
--threads <COUNT>      number of discovery threads (1)
--control              output control messages
--file <FILE>          snmpwalk input file
-i --inventory         chain with netinventory task for discovered devices
-s --save <FOLDER>     base folder where to save discovery and inventory xmls
                        - netdiscovery xmls will go in <FOLDER>/netdiscovery
                        - netinventory xmls will go in <FOLDER>/netinventory
--debug                debug output
-h --help              print this message and exit
--version              print the task version and exit
```

8.3.3 DESCRIPTION

glpi-netdiscovery can be used to run a network discovery task without a GLPI server.

8.3.4 OPTIONS

--first|--host ADDRESS Set the first IP address of the network range to scan.

--last ADDRESS Set the last IP address of the network range to scan.

If not set, it is set with the value of the --first or --host option.

--port PORT[,PORT2] List of ports to try, defaults to: 161

Set it to 161,16100 to first try on default port and then on 16100.

--protocol PROTOCOL[,PROTOCOL2] List of protocols to try, defaults to: udp/ipv4

Possible values are: udp/ipv4,udp/ipv6,tcp/ipv4,tcp/ipv6

--file FILE Run an offline discovery against snmpwalk output, stored in the given file.

If no host or first ip is provided, ip is set to emulate 1.1.1.1 ip scan.

--community STRING Use given string as SNMP community (assume SNMPv1).

--credentials STRING Use given string as SNMP credentials specification. This specification is a comma-separated list of key:value authentication parameters, such as:

- version:2c,community:public
- version:3,username:admin,authprotocol:sha,authpassword:s3cr3t
- etc.

--timeout TIME Set SNMP timeout, in seconds.

--entity ENTITY Set GLPI entity.

--threads COUNT Use given number of inventory threads.

--control Output server-agent control messages, in addition to inventory result itself.

--debug Turn the debug mode on. Multiple usage allowed, for additional verbosity.

8.3.5 EXAMPLES

Run a discovery against a network range, using SNMP version 1:

```
$> glpi-netdiscovery --first 192.168.0.1 --last 192.168.0.254 --community public
```

Run a discovery against a network range, using multiple SNMP credentials:

```
$> glpi-netdiscovery --first 192.168.0.1 --last 192.168.0.254 \
--credentials version:2c,community:public \
--credentials version:3,username:admin,authprotocol:sha,authpassword:s3cr3t
```

Emulate discovery using a snmpwalk file:

```
$> glpi-netdiscovery --file device.walk
```

8.4 glpi-netinventory

8.4.1 NAME

glpi-netinventory - Standalone network inventory

8.4.2 SYNOPSIS

glpi-netinventory [options] [--host <host>|--file <file>]

```
Options:
  --host <HOST>          target host
  --port <PORT[,PORT2]>  SNMP port (161)
  --protocol <PROT[,P2]> SNMP protocol/domain (udp/ipv4)
  --file <FILE>          snmpwalk output file
  --community <STRING>   community string (public)
  --credentials <STRING> SNMP credentials (version:1,community:public)
  --timeout <TIME>       SNMP timeout, in seconds (15)
  --type <TYPE>          force device type
  --threads <COUNT>      number of inventory threads (1)
  --control              output control messages
  --debug                debug output
  -h --help              print this message and exit
  --version              print the task version and exit
```

8.4.3 DESCRIPTION

glpi-netinventory can be used to run a network inventory task without a GLPI server.

8.4.4 OPTIONS

--host HOST Run an online inventory against given host. Multiple usage allowed, for multiple hosts.

--port PORT[,PORT2] List of ports to try, defaults to: 161

Set it to 161,16100 to first try on default port and then on 16100.

--protocol PROTOCOL[,PROTOCOL2] List of protocols to try, defaults to: udp/ipv4

Possible values are: udp/ipv4,udp/ipv6,tcp/ipv4,tcp/ipv6

--file FILE Run an offline inventory against snmpwalk output, stored in given file. Multiple usage allowed, for multiple files.

--communty STRING Use given string as SNMP community (assume SNMPv1)

--credentials STRING Use given string as SNMP credentials specification. This specification is a comma-separated list of key:value authentication parameters, such as:

- version:2c,community:public
- version:3,username:admin,authprotocol:sha,authpassword:s3cr3t
- etc.

--timeout TIME Set SNMP timeout, in seconds.

--type TYPE Force device type, instead of relying on automatic identification. Currently allowed types:

- COMPUTER
- NETWORKING
- PRINTER
- STORAGE
- POWER
- PHONE

--threads count Use given number of inventory threads.

--control Output server-agent control messages, in addition to inventory result itself.

--debug Turn the debug mode on. Multiple usage allowed, for additional verbosity.

8.4.5 EXAMPLES

Run an inventory against a network device, using SNMP version 2c authentication:

```
$> glpi-netinventory --host 192.168.0.1 --credentials version:2c,community:public
```

Run an inventory against a network device, using SNMP version 3 authentication and forcing its type:

```
$> glpi-netinventory --host my.device --type NETWORKING \
--credentials version:3,username:admin,authprotocol:sha,authpassword:s3cr3t
```

8.5 glpi-esx

Attention: The informations on this man page are probably outdated.

8.5.1 NAME

glpi-esx - vCenter/ESX/ESXi remote inventory from command line

8.5.2 SYNOPSIS

glpi-esx --host <host> --user <user> --password <password> --directory <directory>

```
Options:
  --help                 this menu
  --host hostname        ESX server hostname
  --user username        user name
  --password xxxx        user password
  --directory directory  output directory
  --tag tag              tag for the inventoried machine

Advanced options:
  --dump                 also dump esx host full info datas in a *-hostfullinfo.dump file
  --dumpfile file        generate one inventory from a *-hostfullinfo.dump file
```

8.5.3 EXAMPLES

```
% glpi-esx --host myesx --user foo --password bar --directory /tmp
```

You can import the .xml file in your inventory server with the glpi-injector tool.

```
% glpi-injector -v --file /tmp/*.xml -u https://example/front/inventory.php
```

8.5.4 DESCRIPTION

glpi-esx creates inventory of remote ESX/ESXi and vCenter VMware. It uses the SOAP interface of the remote server.

Supported systems:

ESX and ESXi 3.5

ESX and ESXi 4.1

ESXi 5.0

vCenter 4.1

vCenter 5.0

Active Directory users, please note the AD authentication doesn't work. You must create a account on the VMware server.

8.5.5 LIMITATION

So far, ESX serial number are not collected.

8.5.6 SECURITY

The SSL hostname check of the server is disabled.

8.6 glpi-injector

8.6.1 NAME

glpi-injector - A tool to push inventory in an OCS Inventory or compatible server.

8.6.2 SYNOPSIS

glpi-injector [options] [--file <file>|--directory <directory>|--stdin|--useragent <user-agent>]

```
Options:
  -h --help      this menu
  -d --directory load every .ocs files from a directory
  -R --recursive recursively load .ocs files from <directory>
  -f --file      load a specific file
  -u --url       server URL
  -r --remove    remove succesfuly injected files
  -v --verbose   verbose mode
  --debug        debug mode to output server answer
  --stdin        read data from STDIN
  --useragent    set used HTTP User-Agent for POST
  -x --xml-ua    use Client version found in XML as User-Agent for POST
  -x --json-ua   use Client version found in JSON as User-Agent for POST
  --no-ssl-check do not check server SSL certificate
  -C --no-compression don't compress sent XML inventories
```

(continues on next page)

(continued from previous page)

```
Examples:
  glpi-injector -v -f /tmp/toto-2010-09-10-11-42-22.ocs --url https://login:pw@example/
↪front/inventory.php
  glpi-injector -v -R -d /srv/ftp/fusion --url https://login:pw@example/front/inventory.
↪php
```

8.6.3 DESCRIPTION

This tool can be used to test your server, do benchmark or push inventory from off-line machine.

8.7 glpi-remote

Hint: This 'agent' sub command can only be used toward GLPI agent with *Inventory Server Plugin* enabled.

8.7.1 NAME

glpi-remote - A tool to scan, manage and initialize virtual remote agents

8.7.2 SYNOPSIS

glpi-remote [options] [--server server|--local path] [command] [command options]

```
Options:
  -h --help           this menu
  -t --timeout <SECS> requests timeout in seconds (defaults to 10)
  -p --port <LIST>    remote ports list to scan (defaults to '22,5985,5986')
  --ssh               connect using SSH
  --ssl               connect using SSL (winrm or with agent sub-command)
  --no-ssl-check      do not check agent SSL certificate (winrm or agent sub-command)
  --ca-cert-dir <PATH> CA certificates directory
  --ca-cert-file <FILE> CA certificates file (winrm or for agent sub-command)
  --ssl-cert-file     Client certificate file (winrm)
  -u --user           authentication user
  -P --password       authentication password
  -X --show-passwords (list command) show password as they are masked by default
  -c --credentials    credentials list for scan
  -v --verbose        verbose mode
  --debug             debug mode
  -C --no-check       don't check given remote is alive when adding
  -i --inventory      don't register remotes, but run inventory on found remotes
  -T --threads <NUM>  number of threads while scanning (defaults to 1)
  -A --add            add scanned remotes to target so they always be inventoried
                      by RemoteInventory task
  -U --useragent      set used HTTP User-Agent for requests
```

(continues on next page)

(continued from previous page)

```
Target definition options:
  -s --server=<URI>   agent will send tasks result to that server
  -l --local=<PATH>   agent will write tasks results locally
  --target=<TARGETID> use target identified by its id (see list targets command)

Remote GLPI agent having inventory server plugin enabled options:
  -b --baseurl <PATH> remote base url if not /inventory
  -K --token <TOKEN>  token as shared secret
  -I --id <ID>        request id to identify requests in agent log
  --no-compression    ask to not compress sent XML inventories

Sub-commands
  list [targets]      list known remotes or targets
  add <url>+          add remote with given URL list
  del[ete] <index|deviceid>+
                      delete remote with given list index or given deviceid or
                      current known one when alone or all remotes while using
                      __ALL__ as id
  scan <first> [last] [TODO] scan given ip range for remote access or just <first> and
                      register it/them as remote agent
  agent [hosts]       remotely claim an inventory to given remote hosts with a
                      GLPI agent having inventory server plugin enabled
                      (see https://glpi-agent.rtfd.io/inventory-server-plugin.html)

Supported environment variables:
  USERNAME
  PASSWORD
  PORT
  CA_CERT_PATH
  CA_CERT_FILE
  SSL_CERT_FILE
  CREDENTIALS

Examples:
  glpi-remote list
  glpi-remote list targets
  glpi-remote add ssh://admin:pass@192.168.43.237
  glpi-remote add ssh://admin:pass@192.168.43.238 --no-check
  glpi-remote add winrm://admin:pass@192.168.48.250 --no-check --target server0
  glpi-remote delete 1
  glpi-remote scan 192.168.43.1 192.168.43.254
  glpi-remote scan 10.0.0.1 10.0.10.254 --inventory -s https://myglpi/front/inventory.php
  glpi-remote scan 10.0.0.1 10.0.10.254 --inventory -l /var/tmp/remotes
  glpi-remote scan --inventory
  glpi-remote scan 192.168.48.99 | glpi-injector -url https://myglpi/front/inventory.php

Examples for agent command:
  glpi-remote -T strong-shared-secret agent 192.168.43.236
  glpi-remote -v -T strong-shared-secret agent 192.168.43.237 | \
      glpi-injector -url https://myglpi/front/inventory.php
  glpi-remote -T strong-shared-secret -d /var/remote agent 192.168.43.236 192.168.43.237
```

8.7.3 DESCRIPTION

The *glpi-remote* tool is used to manage virtual agents known locally by *glpi-agent*. A virtual agent is used to make remote inventories and is essentially defined by a remote access. A remote access can be defined by ssh authorization for unix/linux platforms or WinRM authorizations for a WinRM enabled platform like win32.

8.7.4 OPTIONS

Most of the options are available in a *short* form and a *long* form. For example, the two lines below are all equivalent:

```
% glpi-agent -s localhost
% glpi-agent --server localhost
```

Target definition options

-s, --server=URI Send the results of tasks execution to given server.

Multiple values can be specified, using comma as a separator.

-l, --local=PATH Write the results of tasks execution locally.

--target=TARGETID Use the given TARGETID to look for the expected target for result submission.

For example, **server0** is the first server target setup in agent configuration.

Remark:

- target option is generaly mandatory while adding remote or scanning for remotes
- if one server and only one is still setup in the agent it will be selected as default target
- when scanning and making inventory, uses any target option or each inventory will be sent to standard output

General options

-t, --timeout=SECS Set the timeout for network requests (defaults to 10 seconds).

-p, --port=LIST Do not contact the target before next scheduled time.

A list of ports used when making a scan and to discover remote computers. The defaults is to scan the standard ssh port and winrm ports: *22,5985,5986*.

--ssh	Use ssh protocol for connection.
--ssl	Use SSL protocol for connecting with WinRM protocol or to a remote agent with inventory server plugin enabled.

--ca-cert-dir=DIRECTORY CA certificates directory.

--ca-cert-file=FILE CA certificates file.

--ssl-cert-file=FILE SSL certificate file for authentication

--no-ssl-check Do not check server SSL certificate.

-u USER, --user=USER Use *USER* for remote authentication.

-p, --password=PASSWORD Use *PASSWORD* for remote authentication.

-X, show-passwords By default, **list** sub-command won't show remotes passwords. This option asks to unmask them during remotes listing.

-c, --credentials=LIST List of credentials to try during a scan.

-v, --verbose Enable verbose mode.

--debug Turn the debug mode on. You can use the parameter up to 2 times in a row to increase the verbosity (e.g: **--debug --debug**).

-C, --no-check Don't check remote is alive while adding it.

-i, --inventory Don't register remotes as they are discovered but just run the inventory task on them.

-T, --threads=NUM Setup number of threads while doing a scan. By default, the agent only uses one thread.

-A, --add Add discovered remotes to local remotes list.

-U, --useragent=USER-AGENT Define HTTP user agent for request (mostly if required for winrm).

agent sub-command options

-b, --baseurl=PATH Remote base url if the default */inventory* has been changed in the remote plugin configuration.

-K, --token=TOKEN Shared secret required to request an inventory to the remote plugin.

-I, <--id>=ID Request-ID to identify the request in the agent log.

--no-compression Ask to skip requested inventory compression.

Sub-commands

- **list** [**targets**]

  list known remotes or list targets

- **add** *url+*

  add remote with given URL list

- **del[ete]** *index|deviceid+*

  Delete remote with given:

  - list index
  - given deviceid
  - current and only one known when no index is given
  - all known remotes while using the **__ALL__** magic word as index

- **scan** *first* [*last*]

  TODO: *This sub-command is still not implemented.*

  Scan given ip range for remote access or just *first* and register it/them as remote agent

- **agent** [*hosts*]

  Remotely claim an inventory to given remote hosts with a GLPI agent having inventory server plugin enabled.

  See online documentation for details: https://glpi-agent.rtfd.io/inventory-server-plugin.html

Environment variables

For security reasons, you can set few environment variables to store sensible datas.

- **USERNAME** to setup connection user
- **PASSWORD** to setup connection password
- **PORT** to setup connection port
- **CA_CERT_PATH**
- **CA_CERT_FILE** to setup the SSL CA certificate file
- **SSL_CERT_FILE** to setup the SSL client certificate file
- **CREDENTIALS** to setup a list of credentials

8.8 glpi-win32-service

Attention: This script is only available on **win32**. It is the base script used to setup the agent as a service during **win32** installation. You don't normally need to run it by itself.

Hint: This script is also handy to setup manually a **win32** service based on a development environment.

8.8.1 NAME

glpi-win32-service - GLPI perl Agent service for Windows

8.8.2 SYNOPSIS

glpi-win32-service [--register|--delete|--help] [options]

```
Options are only needed to register or delete the service. They are handy
while using GLPI perl agent from sources.

Register options:
  -n --name=NAME                  unique system name for the service
  -d --displayname="Nice Name"    display name of the service
  -l --libdir=PATH                full path to agent perl libraries use it if
                                  not found by the script
  -p --program="path to program"  perl script to start as service

Delete options:
  -n --name=NAME                  unique system name of the service to delete

Samples to use from sources base:
  perl bin/glpi-win32-service --help
  perl bin/glpi-win32-service --register
  perl bin/glpi-win32-service --delete
```

(continues on next page)

(continued from previous page)

```
  perl bin/glpi-win32-service --register -n glpi-agent-test -d "[TEST] GLPI Agent Service
↪"
  perl bin/glpi-win32-service --delete -n glpi-agent-test
```

CHAPTER

NINE

DOCUMENTATION LICENSE