GLPI JSON Protocol

Release 1.0

GLPI Project, Teclib'

Jan 26, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

GLPI 10+ and GLPI-Agent can communicate using a JSON protocol described here.

GLPI agent is a tool with few important constraints to know when thinking about protocol:

- by design, it schedules by itself to run its installed *tasks* at an expected time

This is why we will talking about protocol by task: **one task => one protocol spec**

By the way, the first protocol spec is not really an agent task but is inherited from FusionInventory agent history. It's related to a message named *PROLOG*, but for GLPI Agent we will name it *CONTACT* and it specifies mostly the first contact a GLPI Agent will have to do with its configured server, even when not knowing what kind of server it is.

GLPI Agent will be better designed to communicate directly with a GLPI server. With that goal in mind, we need to first specify few features *COMMON* to all task protocol specs.

CHAPTER

TWO

FUTURE EVOLUTION

Few evolutions are expected to this JSON protocol as we wanted to enhance communication between GLPI and agents:

- Agent will be able to use *REGISTER* to secure protocol exchanges
- Agent will be able to use *CONFIGURATION* to get its configuration from server
- As agent supported tasks will be supported in GLPI 10 Core, they will start using:
  - *NETDISCOVERY* protocol specs
  - *NETINVENTORY* protocol specs
  - *COLLECT* protocol specs
  - *DEPLOY* protocol specs
  - *ESX* protocol specs
  - *WAKEONLAN* protocol specs
- It should also be possible to setup remoteinventory tasks with *REMOTEINVENTORY*

CHAPTER

THREE

COMMON

Common specs to all task protocol specs with some specific explanations on some goal.

Hint: Supported since GLPI 10.0.0

3.1 Transport protocol

Messages are passed between client and server via HTTP or HTTPS protocol.

3.2 HTTP headers

3.2.1 `GLPI-Agent-ID`

An agent identity MUST be set in HTTP headers:

- This is the **agentid**
- **agentid** HTTP header: `GLPI-Agent-ID`
- **agentid** format: plain text UUID which can be reduced in a 128 bits raw id by code
- example: `3a609a2e-947f-4e6a-9af9-32c024ac3944`
- it is generated by the agent the first time it starts
- server must also set it in answer to specify agent destination, this can be used by a proxy agent to route server answer

3.2.2 `GLPI-Request-ID`

A request id CAN be set in HTTP headers:

- HTTP header: `GLPI-Request-ID`
- format: a 8 digit hexadecimal string in higher case like `42E6A9AF1`
- When present, it **MUST** be set back in answer
- When present, it **SHOULD** be logged
- It **MUST** be set by a proxy in an answer when the answer is still not definitively known

- It **MUST** be set when using a GET request to query the final request status from a proxy. It **MUST** then be identical the the one provided by the previous proxy answer.

3.2.3 Content-Type

The content type MUST be set in HTTP headers and follow the official public specs:

- HTTP header: `Content-Type`
- The following content type are to be supported:
  - `application/json`: **MANDATORY SUPPORT** This format must be supported as this will be the default format
  - `application/x-compress-zlib`: Compressed JSON with zlib compression
  - `application/x-compress-gzip`: Compressed JSON with gzip compression
  - `application/x-compress-br`: Compressed JSON with brotli compression
  - `application/xml`: Only for [CONTACT](specs/CONTACT) protocol

3.2.4 Accept

A party CAN announce which content types it supports setting `Accept` HTTP header

- HTTP header: `Accept`
- format: must follow RFC7231
- can be used to try another content type on "unsupported content-type" error
- Example:
  - `Accept: application/json`
  - `Accept: application/xml; q=0.1, application/json; q=0.5, application/x-compress-zlib; q=0.7, application/x-compress-gzip; q=0.8, application/x-compress-br`

3.2.5 Pragma

The agent requests MUST includes the pragma header to avoid any caching done by the server:

- HTTP header and value: `Pragma: no-cache`

3.2.6 GLPI-CryptoKey-ID

If an encryption key has been shared and encryption is enabled, a side party must announce the key owner used to encrypt data

- HTTP header: `GLPI-CryptoKey-ID`
- format: same as agentid
- the same encryption should be used when answering unless encryption is disabled on a side
- in a proxy context, this id can be different than `GLPI-Agent-ID`
- when this header is missing, data are not expected to be encrypted

3.2.7 GLPI-Proxy-ID

When a GLPI-Agent is used as a proxy, it MUST set/update this HTTP header with its own agentid.

- HTTP header: `GLPI-Proxy-ID`
- format: list of agentid separated by commas
- a proxy agent must check its agentid is still not in the list, otherwise it **MUST** answer a HTTP 404 error with a JSON error message (see below): "proxy-loop-detected"
- a proxy agent parameter should define the maximum number of proxy to be set in the list. In the case, the header reaches the maximum on a proxy, it **MUST** answer a HTTP 404 error with a JSON error message (see below): "too-many-proxy". By default, the maximum number of proxy should be set to 5 as that's still large and most usercase will only use 1 proxy.
- GLPI server should show the proxy list in the agent management panel.

3.3 JSON template messages

3.3.1 Requests

```
{
  "action": "<token>"
}
```

Description:

- "action": [optional] but **should** be set to a supported token otherwise action defaults to *inventory* and then **MUST** respect the *INVENTORY* protocol specs.

3.3.2 Answers

```
{
  "status": "<token>",
  "message": "<string>",
  "expiration": "<delay definition>"
}
```

Description:

- "status": [**mandatory**] the resulting status of the request related to the action. Must be a supported token, see related protocol.
- Common tokens: "pending", "error", "ok"
- The "pending" token with a short "expiration" delay could be used by proxy agents to tell agents to retry the same request soon as the proxy has the time to obtain the server expected answer.
- "message": [optional] a message to keep in log as an error reason or for debugging purpose
- "expiration": [optional] the delay before the agent can send another message for the current action. The delay may have different meanings regarding the related action. The delay definition must be a positive integer number immediately followed by an optional letter defining the number unit. The letter should be one of "s", "m", "h", "d" which are respectively representing seconds, minutes, hours and days. The default is "h" when no unit follows the number.

- if not set, no delay is required and next request can be handled immediately
- in case of error or no answer, the latest delay or configured one should be used by the client
- example: "1d", "24", "1m", "30s", "6h"

3.3.3 Error answers

```
{
  "status": "error",
  "message": "unsupported content-type"
}
```

Description:

- On error, the HTTP answer error code **MUST** be set with the 4XX or 5XX HTTP related error
- For 4XX errors, status code in free, but the answer **SHOULD** be a short JSON message with the "error" value set as "status" property and eventually a "message" property to help the other party to analyze the error

CHAPTER

FOUR

CONTACT

CONTACT protocol is needed to permit GLPI Agent to detect it communicates with a supported server (GLPI, FusionInventory plugin for GLPI).

Hint: Supported since GLPI 10.0.0

Features:

- Keep compatibility with FusionInventory as fallback
- Cover all the same features then FusionInventory protocol
- Add more features:
  - agent tells server which tasks are installed
  - agent tells server which tasks are enabled
  - server tells which tasks has to be disabled
  - agent tells its configured tag so it can have a meaning for any task

4.1 A. Agent CONTACT request

The agent **MUST** set new HTTP headers accordingly to the *COMMON* protocol specs.

2 cases: the agent knows or not server supported content types. A new agent option should permit to force this "knowledge".

4.1.1 A1. Agent does NOT know server supported content-types

Based on what is still doing FusionInventory agent:

- the agent can send a HTTP request named **PROLOG** with such following content:

```
<REQUEST>
  <QUERY>PROLOG</QUERY>
  <TOKEN>12345678</TOKEN>
  <DEVICEID>foo-agent-deviceid</DEVICEID>
</REQUEST>
```

About HTTP headers in *PROLOG* request:

- *Content-Type* is restricted to the following values, depending on supported compression scheme as they are the only ones supported by FusionInventory for GLPI plugin:
  - *application/xml*
  - *application/x-compress-zlib*
  - *application/x-compress-gzip*

4.1.2 A2. Agent think it knows server supported content-types

This can indeed be an assumption based on previously exchanged messages.

- the agent sends a HTTP request named **CONTACT** with the following JSON possible content:

```
{
  "action": "contact",
  "deviceid": "classic-agent-deviceid",
  "name": "GLPI-Agent",
  "version": "1.0",
  "installed-tasks": [
    "inventory",
    "register",
    "..."
  ],
  "enabled-tasks": [
    "collect",
    "deploy",
    "..."
  ],
  "httpd-port": "62354",
  "httpd-plugins": {
    "ssl": {
      "disabled": "no",
      "ports": [
        "0",
        "62356"
      ],
      "ssl_cert_file": "cert.pem",
      "ssl_key_file": "key.pem"
    },
    "proxy": "disabled",
    "toolbox": "disabled",
    "..."
  },
  "tag": "awesome-tag"
}
```

Description:

- `action`: [**mandatory**] must be set to "contact"
- `deviceid`: [**mandatory**] just a friendly string to name an agent
- `name`: [**mandatory**] the product name of the agent
- `version`: [**mandatory**] the product version of the agent

- `installed-tasks`: [**mandatory**] a list of agent installed tasks
- `enabled-tasks`: [optional] a list of enabled task if different than "installed-tasks"
- `httpd-port`: [optional] the httpd port used by the agent if it listens on
- `httpd-plugins`: [optional] a list of agent httpd plugins with "disabled" status or its configuration if enabled
- `tag`: [optional] the "tag" setup in the agent configuration

4.2 B. Server CONTACT answer

As the server detects `GLPI-Agent-ID` as HTTP header, it SHOULD always answer using the new protocol. The answer could have the following content:

```
{
  "status": "<token>",
  "message": "<optional string>",
  "tasks": {
    "inventory": {
      "no-category": "processes",
      "server": "glpi",
      "version": "10.0.0"
    },
    "networkinventory": {
      "server": "glpiinventory",
      "version": "1.0"
    },
    "deploy": {
      "server": "glpiinventory",
      "version": "1.0"
    }
  },
  "disabled": [
    "collect",
    "wakeonlan",
    "remoteinventory"
  ],
  "jobs": {
    "networkinventory": [
      {
        "task": "networkinventory",
        "jobid": "this job id",
        "credentials": [ "1", "2" ]
      }
    ],
    "deploy": [
      {
        "task": "deploy",
        "jobid": "this job id"
      }
    ]
  },
```

(continues on next page)

(continued from previous page)

```
  "credentials": {
    "1": {
      "community": "public",
      "type": "snmp",
      "version": "v1"
    },
    "2": {
      "community": "public",
      "type": "snmp",
      "version": "v2c"
    }
  },
  "expiration": "1d"
}
```

Description:

- `status`: [**mandatory**] the resulting status of the request in "error", "pending", "ok"
- `expiration`: [**mandatory**] the delay before asking for another CONTACT. It has the same purpose than PROLOG_FREQ but includes a unit, example "1d" for one day.
- `tasks`: [optional] a list of task configuration objects reduced as JSON structure
  - should be used to pass params to listed tasks
  - as example, "url" can be set to set another URL to request during the task processing
  - the list can be empty
- `disabled`: [optional] a list of tasks disabled on server side so it won't be run by the agent and the agent will log the task is disabled
- `jobs`: [optional] a dictionary with task names as key and list of jobs as values. Each jobs list is an ordered list of job objects in the task related format related as JSON structure.
- `credentials`: [optional] a dictionary of needed credentials. Keys are positive integer numbers and should be referenced in a job object. Values are credentials objects as JSON structure.
- `message`: [optional] message to be logged on agent side with error or at debug level

4.3 Error handling

In case of error on server side, "status" should be set to "error" in the answer and a meaningful and short string should be set as "message". The returned HTTP code should be set to 4XX. See *COMMON* for error message specs.

Example: .. code:

```
{
  "status": "error",
  "message": "malformed json",
  "expiration": "1d"
}
```

```
{
  "status": "error",
  "message": "forbidden",
  "expiration": "1d"
}
```

```
{
  "status": "error",
  "message": "unsupported content-type",
  "expiration": 24
}
```

```
{
  "status": "error",
  "message": "busy server",
  "expiration": "30m"
}
```

CHAPTER

FIVE

INVENTORY

INVENTORY protocol can be used by Inventory, NetDiscovery and NetInventory agent tasks, and also by injector, to submit any inventory.

Hint: Supported since GLPI 10.0.0

5.1 A. Agent request to submit an inventory

The request **MUST** match GLPI "inventory_format" specs and *COMMON*.

```
{
    "action": "inventory",
    "deviceid": "<device id>",
    "content": {
        "...inventory object following inventory_format..."
    },
    "itemtype": "Computer"
}
```

Description:

- "action": [optional] and could be set to "inventory", "netdiscovery" or "netinventory", defaults is "inventory" when missing
- "deviceid": [**mandatory**] as defined in inventory_format
- "content": [**mandatory**] as defined in inventory_format
- "itemtype": [optional] as defined in inventory_format but defaults to `Computer`

5.2 B. Server answer to a submitted inventory

See also *COMMON* protocol specs.

On successul submission, we expect a simple answer:

```
{
    "status": "ok",
    "expiration": 24
}
```

The server can return an error like:

```
{
    "status": "error",
    "message": "bad-format",
    "expiration": 24
}
```

or

```
{
    "status": "error",
    "message": "busy server",
    "expiration": 1
}
```

On *busy server* error, the agent should keep the inventory and retry its submission at the specified expiration.

CHAPTER

SIX

REGISTER

REGISTER protocol should be used by the GLPI Agent Register task to fully register the agent with a GLPI server. It could also be used to register the agent on another agent acting as a proxy agent thanks to its Proxy plugin.

> **Attention:** This specification is still considered as a **DRAFT** as not implemented in GLPI 10 and GLPI-Agent

Features:

- agent is identified by its **agentid** in UUID format
- new configuration parameter on agent side: *token*
  - without it or if it is invalid, only simple registration is possible and in that case the server should be configured to accept that
  - with a valid token, all exchanges with the server after registration will be secured and encrypted, even if not done throught a SSL connection
  - the token format is UUID string as:
    - this format is indeed representing 16 bytes in hexdecimal
    - this can be used as 128 bits bloc like 128 bits key for AES cipher
- a 128 bits encryption key can be provided during the registration
  - the key will have to be renew after an expiration
- this protocol could be supported by the agent Proxy plugin:
  - the proxy agent could manage its own tokens, unknown from the server
  - the proxy agent could not know the final token and then just pass messages being unable to know what secrets are exchanged
  - the registration process could also permit server to contact agents behind proxy agents
- on server side:
  - we need a way to manage tokens
    - token creation
    - token revocation involving all agent provided keys revocation
  - each token can be conditioned on the use of a tag
  - we must be able to revoke any key provided to an agent or to ask for a registration renew
  - agent keys management: cleanup expired keys
  - new agent could first have to be manually validated

- encryption could be optional: we need an option to disable encryption

6.1 Protocol

Few messages could be exchange between agent and server during agent registration. The base format of messages is JSON and should respect the *COMMON* protocol specs.

6.1.1 1. First message from the agent

Example:

```
{
    "action": "register",
    "deviceid": "classic-agent-deviceid",
    "port": 62354,
    "name": "GLPI-Agent",
    "version": "1.0",
    "tag": "awesome-tag"
}
```

Description:

- `action`: [**mandatory**] must be set to "register"
- `deviceid`: [**mandatory**] just a friendly string to name an agent
- `port`: [**mandatory**] the TCP port on which the agent is joinable or 0 if not joinable
- `name`: [**mandatory**] the product name of the agent
- `version`: [**mandatory**] the product version of the agent
- `tag`: [optional] the setup "tag" in the agent configuration

6.1.2 2. Server answer

Examples:

```
{
    "status": "registered",
    "expiration": "30d"
}
```

```
{
    "status": "error",
    "message": "forbidden",
    "expiration": "4h"
}
```

```
{
    "status": "pending",
    "needs": "token-validation",
```

(continues on next page)

(continued from previous page)

```
    "expiration": "1m",
    "challenge": "a3540c0e-ac3c-46cf-892f-692ca02209f8"
}
```

Description:

- `status`: [**mandatory**] the resulting status of the request in `registered`, `error` or `pending`
- `message`: [optional] a message to keep in log as an error reason or for debugging purpose
- `expiration`: [**mandatory**] the expiration of the status
- `needs`: [optional] a string in `token-validation`, `manual-validation`, `server-validation` but should be set if `status` is `pending`
- `challenge`: [optional] a string in UUID format which is a 128 bits cryptographic challenge *(details in Cryptographic exchanges chapter)*. It must be set when `status` is `pending` and `needs` is `token-validation`.

About `expiration`, it has different meanings:

- if `status` is `registered`, the agent will have to register again before the expiration:
  - by default, it tries to register again after the half of the expiration
  - if it has no answer, it will wait at the middle of the remaining delay
  - the delay should not be lower than the *CONTACT* protocol delay
- if `status` is `error`, the agent should not try to register (and even to communicate) before the given expiration
- if `status` is `pending`:
  - if `needs` is `token-validation`, this is the expiration of the challenge as the agent should answer the challenge asap
  - if `needs` is `server-validation` or `manual-validation`, this is the delay for the next contact with the same request. `server-validation` can be returned by a proxy and should not be used by GLPI server.

The agent is not expected to request another message unless `status` is `pending` and `needs` is `token-validation`. Unless that case, next agent register message is like a new registration, the big difference is the message is encrypted if it is registered and the expiration has not been reached.

6.1.3 3. Agent token validation message

Example:

```
{
    "action": "register",
    "challenge": "07f2cc8b-194c-45b9-a4e8-68a78129b8e6"
}
```

```
{
    "action": "register",
    "challenge": "failure"
}
```

Description:

- `action`: [**mandatory**] must be set to `register`
- `challenge`: [**mandatory**] in principle, a string in UUID format which is a 128 bits cryptographic challenge *(details in Cryptographic exchanges chapter)*
  - It must be the answer to the challenge defined by the server
  - It case of error on agent side, can be set to a message like simply `failure`

6.1.4 4. Server challenge answer

Examples:

```
{
    "status": "registered",
    "expiration": "30d",
    "challenge": "393c263e-1168-44a7-bbdc-6d2ce8514db0",
    "crypto": "680ca885-e017-44a4-81c9-729f759ee3c6"
}
```

```
{
    "status": "pending",
    "expiration": "1m"
}
```

```
{
    "status": "error",
    "message": "challenge failed",
    "expiration": "1h"
}
```

Description:

- `status`: [**mandatory**] the resulting status of the request in `registered`, `error` or `pending`
  - `pending` is to be used by proxy agents. The agent will have to send again the same challenge at expiration.
- `message`: [optional] a message to keep in log as an error reason or for debugging purpose
- `expiration`: [**mandatory**] the expiration of the status
- `challenge`: [optional] a string in UUID format which is the final 128 bits encrypted server answer challenge *(details in Cryptographic exchanges chapter)*
- `crypto`: [optional] a string in UUID format which is a 128 bits encrypted key *(details in Cryptographic exchanges chapter)*. It is optional as encryption may be not required by the server.

6.2 Cryptographic exchanges

All cryptographic exchanges are based on AES with 128 bits keys.

6.2.1 1. First challenge from the server

When the server has to create a 128 bits challenge:

- it uses 8 random bytes (64 bits) as first part, this is the **server secret**
- it select an agentid: the one from the HTTP header or one from the `GLPI-Proxy-ID` HTTP header list. This is to support the case where we are sure we didn't share a token with the agent but we trust a proxy. The tag could be used to trust a proxy.
- it concatenates the first 8 random bytes with the last 8 bytes of the selected agentid taken as raw 16 bytes
- it encrypts this 128 bits secret with AES cipher using the token as 128 bits key. The token is the one the server expects the target agent knows.
- it transforms the secret as UUID string to be included in the JSON answer as `challenge` parameter

6.2.2 2. Challenge handling in the agent

When an agent receive a first server answer with a `challenge`, it has to:

- transform the UUID challenge into a 128 bits bloc
- decrypt the bloc with AES cypher using its configured token as 128 bits key to obtain the secret
- compare the last 64 bits of the secret to its own agentid last 64 bits:
  - if the bits doesn't match:
    - if the agent is not a proxy, this is an error, the agent can send a message with `failure` as `challenge` parameter. The agent expect a `status` set to `error` and an `expiration` set to a delay before retrying a registration
    - if the agent is a proxy and does the registration on the behalf of another agent, it keeps the challenge to be include in the answer for the next contact of the related agent
      - security notes: if agent and proxy shares the same token, the proxy could see the 64 bits matched the target agentid and then it knows the secret in the first 64 bits. It is then advised to not use the same tokens for agent and proxy. To be safe, each proxy should even have its own and personal token. In that way, the proxy won't be able to know anything about all exchange between the agent and the server
  - if the bits matches:
    - the agent is the target of the challenge
    - the challenge secret is the first 64 bits
- the agent uses the challenge secret as first 64 bits for the answer challenge
- it uses 8 random bytes (64 bits) as **agent secret** for the last 64 bits and obtain a 128 bits answer challenge
- it encrypts this 128 bits secret with AES cipher using the token as 128 bits key. Of course, the token is the one the agent expects the server knows.
- it transforms the encrypted bloc as UUID string to be included in the JSON answer as `challenge` parameter

6.2.3 3. Answer challenge handling in the server

As an agent proxy knowing the answer is expected by a server:

- returns a message with `status` set to `pending`
- transmit the challenge to the server

Otherwise as the final server:

- transform the UUID answer challenge into a 128 bits bloc
- decrypt the bloc with AES cypher using the expected token as 128 bits key to obtain the secret
- compare the first 64 bits of the secret to the expected **server secret** defined in step 1
  - if the bits doesn't match:
    - return an `error` message and abort the registration
- as the bits matches, the last 64 bits will be used as **agent secret**
- **agent secret** and **server secret** are concatenated in that order into a 128 bits bloc
- the bloc is encrypted with AES cipher using the token as 128 bits key
- the encrypted bloc is transformed as UUID string to be included in the final JSON answer as `challenge` parameter
- a private 128 bits keys is generated as 16 random bytes an associated to the agent
- the private key as 128 bits blocs is encrypted with AES cipher using the token as 128 bits key
- that encrypted bloc is transformed as UUID string to be included in the final JSON answer as `crypto` parameter

6.2.4 4. Final answer challenge handling in the agent

As an agent proxy knowing the answer is not for itself:

- the message is saved
- the saved message is transmitted to the following agent at next contact

Otherwise as the target agent:

- transform the UUID answer challenge into a 128 bits bloc
- decrypt the bloc with AES cypher using the token as 128 bits key to obtain the secret
- compare the first 64 bits of the secret to the expected **agent secret** defined in step 2
  - if the bits doesn't match:
    - send an `register` message with `failure` as `challenge`
- compare the last 64 bits of the secret to the expected **server secret** defined in step 1
  - if the bits doesn't match:
    - send an `register` message with `failure` as `challenge`
- if present, transform the UUID in `crypto` into a 128 bits bloc
- decrypt the bloc with AES cypher using the token as 128 bits key to obtain the communication 128 bits key. This key can now be used for all future communications.

6.3 Remarks

6.3.1 About port & proxy

The port should be set to the proxy one on the first proxy transmitted message toward next server unless the agent or a proxy has its HTTP listener disabled. So if an option is enabled on proxy, it will also be able to join the agent on the behalf of the server. This should even work with more than one proxy between server and target agent. Only asynchronous messages should be handled that way, so each protocol specs should support asynchronous messaging.

CHAPTER

SEVEN

CONFIGURATION

Attention: This specification is planned for the future

CHAPTER

EIGHT

NETDISCOVERY

Attention: This specification is planned for the future

CHAPTER

NINE

NETINVENTORY

Attention: This specification is planned for the future

CHAPTER

TEN

ESX

Attention: This specification is planned for the future

CHAPTER

ELEVEN

COLLECT

Attention: This specification is planned for the future

CHAPTER

TWELVE

DEPLOY

Attention: This specification is planned for the future

CHAPTER

THIRTEEN

WAKEONLAN

Attention: This specification is planned for the future

CHAPTER

FOURTEEN

REMOTEINVENTORY

Attention: This specification is planned for the future

CHAPTER

FIFTEEN

SPECIFICATIONS LICENSE