Privet

(Cloud Device Local Discovery/API protocol)

1. Introduction

Cloud connected devices have many benefits. They can use online conversion services, host job queues while the device is offline, and be accessible from anywhere in the world. However, with many cloud devices accessible by a given user, we need to provide a method for finding the nearest device based on location. The purpose of the Privet protocol is to bind the flexibility of cloud devices with a suitable local discovery mechanism so that devices are easily discovered in new environments.

The goals of this protocol are:

- make cloud devices locally discoverable
- register cloud devices with a cloud service
- associate registered devices with their cloud representation
- enable offline functionality
- simplify implementation so that small devices can utilize it

The Privet protocol consists of 2 main parts: discovery and API. Discovery is used to find the device on the local network, and the API is used to get information about the device and perform some actions. Throughout this document, “the device” refers to a cloud connected device implementing the Privet protocol.

2. Discovery

Discovery is a zeroconf based (mDNS + DNS-SD) protocol. The device MUST implement IPv4 Link-Local Addressing. The device MUST comply with the mDNS and DNS-SD specs.

http://www.rfc-editor.org/rfc/rfc3927.txt (IPv4 Link-local)
http://www.rfc-editor.org/rfc/rfc4862.txt (IPv6 Link-local)
http://www.rfc-editor.org/rfc/rfc6762.txt (mDNS)
http://www.rfc-editor.org/rfc/rfc6763.txt (DNS-SD)

The device MUST perform name conflict resolution according to the above specifications.

2.1. Service Type

DNS Service Discovery uses the following format for service types: ***_applicationprotocol._transportprotocol***. In the case of the Privet protocol, the service type for DNS-SD should be: **_privet._tcp**

The device can implement other service types as well. It is advised to use the same service instance name for all service types implemented by the device. For example: a printer may implement “*Printer XYZ._privet._tcp*” and “*Printer XYZ._printer._tcp*” services. It will simplify setup for the user. However, Privet clients will look only for “*_privet._tcp*”.

In addition to the main service type, the device MUST advertise the PTR records for its corresponding subtype(s) (see DNS-SD spec: “7.1. Selective Instance Enumeration (Subtypes)”). Format should be following: `_<subtype>._sub._privet._tcp`

Currently the only device subtype supported is **printer**. So, all printers MUST advertise two PTR records:

`_privet._tcp.local.`
`_printer._sub._privet._tcp.local.`

2.2. TXT record

The DNS Service Discovery defines fields to add optional information about a service in the *TXT records*. A *TXT record* consists of key/value pairs. Each key/value pair starts from the length byte followed by up to 255 bytes of text. The key is the text before the first '=' character and the value is the text after the first '=' character until the end. The specification allows for no value in the record, in such case the will be no '=' character OR no text after the '=' character. (See DNS-SD spec: "6.1. General Format Rules for DNS TXT Records" for the DNS TXT record format and "6.2. DNS-SD TXT Record Size" for the recommended length).

Privet requires the device to send the following key/value pairs in the TXT record. Key/Value strings are case-insensitive, for example "`CS=online`" and "`cs=ONLINE`" are the same. Information in the TXT record MUST be the same as accessible through /info API (see 4.1. API section).

It is recommended to keep TXT record size under 512 bytes.

2.2.1. txtvers

Version of the TXT structure. txtvers MUST be the first record of the TXT structure. Currently the only supported version is 1.

```
txtvers=1
```

2.2.2. ty

Provides a user-readable name of the device. For example:

```
ty=Google Cloud Ready Printer Model XYZ
```

2.2.3. note (optional)

Provides a user-readable name of the device. For example:

```
note=1st floor lobby printer
```

This is an optional key and may be skipped. However, if present, user SHOULD be able to modify this value. The same description MUST be used when registering device.

2.2.4. url

Server URL this device is connected to (including protocol). For example:

```
url=https://www.google.com/cloudprint
```

2.2.5. type

Comma-separated list of device subtypes supported by this device. Format is:
"`type=_subtype1,_subtype2`". Currently, the only supported device subtype is `printer`.

```
type=printer
```

Each subtype listed should be advertised using a corresponding PTR record. For each supported service subtype, there should be one corresponding item. Service subtype name (<subtype>._sub._privet._tcp) should be equal to device type here.

2.2.6. id

Device ID. If the device has not been registered yet, this key should be present, but value should be empty. For example:

```
id=11111111-2222-3333-4444-555555555555
id=
```

2.2.7. cs

Indicates the device's current connection state. Four possible values are defined in this spec.

- "**online**" indicates that the device is currently connected to the cloud.
- "**offline**" indicates that the device is available on the local network, but can't talk to the server.
- "**connecting**" indicates that the device is performing its startup sequence and is not fully online yet.
- "**not-configured**" indicates that the device's internet access has not been configured yet. This value is not currently used, but may be useful in future versions of the specification.

For example:

```
cs=online
cs=offline
cs=connecting
```

If the device has been registered with a cloud, on startup it should check connectivity with a server to detect its connection state (for example, calling cloud API to get device settings). The device may use its notifications channel (e.g. XMPP) connection state to report this value. Unregistered devices on startup may ping a domain in order to detect their connection state (for example, ping www.google.com for cloud print devices).

3. Announcements

On device startup, shutdown or state change, the device MUST perform the announcement step as described in the mDNS specification. It SHOULD send the corresponding announcement at least twice with at least a one-second interval between them.

3.1. Startup

On device startup it MUST perform probing and announcing steps as described in the mDNS specification. SRV, PTR and TXT records should be sent in this case. It is recommended to group all records into one DNS response if possible. If not, the following order is recommended: SRV, PTR, TXT records.

3.2. Shutdown

On device shutdown it SHOULD try to notify all interested parties about it by sending a "goodbye packet" with TTL=0 (as described in mDNS documentation).

3.3. Update

In case of any information described in TXT has changed, the device MUST send an update announcement. It is enough to only send the new TXT record in this case. For example, after a device is registered, it MUST send an update announcement including the new device id.

4. API

After a cloud device has been discovered, client communication is enabled with the device directly over the local network. All APIs are HTTP 1.1 based. Data formats are JSON based. API requests may be GET or POST requests.

Each request MUST contain a valid "**X-Privet-Token**" header. The ONLY request allowed to have an empty "X-Privet-Token" header is the `/privet/info` request (note that the header MUST still be present). If the "X-Privet-Token" header is missing, the device MUST respond with the following HTTP 400 error:

```
HTTP/1.1 400 Missing X-Privet-Token header.
```

If "X-Privet-Token" header is empty or invalid, the device MUST respond with "*invalid X-Privet-Token error*" (*invalid_x_privet_token,* see Errors section for details). The only exception is the /info API. To see more info on why this is done and how tokens should be generated, see Appendix A: XSSI and XSRF attacks and prevention.

If a requested API does not exist or is not supported, the device MUST return an HTTP 404 error.

4.1. API availability

Before ANY API is exposed (including the /info API), the device MUST contact the server to check local settings. Local settings MUST be preserved between restarts. If the server is not available, the last known local settings should be used. If the device has not been registered yet, it should follow the default settings.

Cloud Print devices MUST follow the steps below to register, receive and update local settings.

4.1.1. Registration

When the device registers, it MUST specify the "local_settings" parameter, as follows:

```
{
      "current": {
            "local_discovery": true,
            "access_token_enabled": true,
            "printer/local_printing_enabled": true,
            "printer/conversion_printing_enabled": true,
            "xmpp_timeout_value": 300
      }
}
```

The following settings can be set.

Value name	Value type	Description
`local_discovery`	boolean	Indicates if local discovery functionality is allowed. If "false", ALL local API (including /info) and DNS-SD discovery MUST be disabled. By default, newly

		registering devices should pass “true”.
access_token_enabled	boolean (optional)	Indicates if /accesstoken API should be exposed on the local network. By default should be “true”.
printer/local_printing_enabled	boolean (optional)	Indicates if local printing functionality (/printer/createjob, /printer/submitdoc, /printer/jobstate) should be exposed on the local network. By default should be “true”.
printer/conversion_printing_enabled	boolean (optional)	Indicates if local printing may send job to server for conversion. Only makes sense when local printing is enabled.
xmpp_timeout_value	int (optional)	Indicates the number of seconds between XMPP channel pings. By default MUST be 300 (5 minutes) or more.

The lack of any optional value indicates that the corresponding functionality is completely unsupported by the device.

4.1.2. Startup

On device startup, it should contact the server to check what APIs are available to be exposed in the local network. For printers connected to Cloud Print, they should call:

```
/cloudprint/printer?id=<printer_id>
```

or

```
/cloudprint/list
```

/cloudprint/printer is preferred over /cloudprint/list, but both will work.

This API returns current device parameters, including settings for local API. The reply from the server will have the following format:

```
“local_settings”: {
      “current”: {
            “local_discovery”: true,
            “access_token_enabled”: true,
            “printer/local_printing_enabled”: true,
            “printer/conversion_printing_enabled”: true,
            “xmpp_timeout_value”: 300
      },
      “pending”: {
            “local_discovery”: true,
            “access_token_enabled”: true,
            “printer/local_printing_enabled”: false,
            “printer/conversion_printing_enabled”: false,
            “xmpp_timeout_value”: 500
      }
}
```

“current” object indicates settings that are in effect at the moment.

"pending" object indicates settings that should be applied to the device (this object may be missing).

Once the device sees "pending" settings, it MUST update its state (see below).

4.1.3. Update

If settings update is needed, an XMPP notification will be sent to the device. The notification will be in the following format:

```
<device_id>/update_settings
```

On receiving such a notification, the device MUST query the server to get the latest settings. Cloud Print devices MUST use:

```
/cloudprint/printer?id=<printer_id>
```

Once the device sees "pending" section as a result of the /cloudprint/printer API (at startup or due to the notification), it MUST update its internal state to remember the new settings. It MUST call the server API to confirm the new settings. For Cloud Printers, the device MUST call /cloudprint/update API and use "local_settings" parameter as during registration.

When re-connecting to XMPP channel, the device MUST call /cloudprint/printer API to check if local settings has been changed since the last time.

4.1.3.1. Local Settings Pending

"local_settings" parameter that device uses to call server API MUST NEVER contain "pending" section.

4.1.3.2. Local Settings Current

ONLY the device can change the "current" section of the "local_settings". Everybody else will change the "pending" section, and wait until changes get propagated to the "current" section by the device.

4.1.4. Offline

When unable to contact the server during startup, after notification, device MUST use last known local settings.

4.1.5. Deleting device from the service

If the device has been deleted from the service (GCP for example), an XMPP notification will be sent to the device. The notification will be in the following format:

```
<device_id>/delete
```

On receiving such a notification, the device MUST go to the server to check its state. Cloud Print devices MUST use:

```
/cloudprint/printer?id=<printer_id>
```

The device MUST receive a successful HTTP answer with success=false and no device/printer description. It means device has been removed from the server, and the device MUST erase its credentials and go to default factory settings mode.

ANY time the device receives a reply indicating it has been deleted as a result of the /cloudprint/printer

API (startup, update settings notification, daily ping), it MUST delete its credentials and go to default mode.

4.2. /privet/info API

The info API is MANDATORY and MUST be implemented by every device. It is an HTTP **GET** request for "**/privet/info**" url:

```
GET /privet/info HTTP/1.1
```

The info API returns basic information about a device and functionality it supports. This API MUST never change the device status or perform any action, since it is vulnerable to XSRF attacks. This is the ONLY API allowed to have an empty "X-Privet-Token" header. Clients should call /privet/info API with "X-Privet-Token" header set to "":

```
X-Privet-Token: ""
```

The info API MUST return data consistent with data available in the TXT record during discovery.

4.2.1. Input

`/privet/info` API has no input parameters.

4.2.2. Return

`/privet/info` API returns basic information about device and supported functionality.

The TXT column indicates the corresponding field in the DNS-SD TXT record.

Value name	Value type	Description	TXT
`version`	string	Highest version (major.minor) of API supported, currently 1.0.	
`name`	string	Human readable name of the device.	`ty`
`description`	string	(optional) Device description. SHOULD be modifiable by user.	`note`
`url`	string	URL of the server this device is talking to. URL MUST include protocol specification, for example: https://www.google.com/cloudprint.	`url`
`type`	list of strings	List of device types supported.	`type`
`id`	string	Device id, empty if device has not been registered yet.	`id`
`device_state`	string	State of the device. "**idle**" means device is ready "**processing**" means device is busy and functionality may be limited for some time "**stopped**" means device is not working and user intervention is required	

connection_state	string	State of the connection to the server (base_url) “**online**” - connection available “**offline**” - no connection “**connecting**” - performing startup steps “**not-configured**” - connection has not been configured yet A registered device may report its connection state based on the state of the notification channel (e.g. XMPP connection state).	cs
manufacturer	string	Name of the device manufacturer	
model	string	Model of the device	
serial_number	string	Unique device identifier. In this spec, this MUST be a UUID. (GCP 1.1 spec) (optional) We strongly recommend using the same serial number ID everywhere, so different clients can identify the same device. For example, printers implementing IPP may use this serial number ID in “printer-device-id” field.	
firmware	string	Device firmware version	
uptime	int	Number of seconds from the device boot.	
setup_url	string	(optional) URL (including protocol) of the page with setup instructions	
support_url	string	(optional) URL (including protocol) of the page with support, FAQ information	
update_url	string	(optional) URL (including protocol) of the page with update firmware instructions	
x-privet-token	string	Value of the “X-Privet-Token” header that has to be passed to all APIs to prevent XSSI and XSRF attacks. See 6.1. for details.	
api	description of APIs	List of supported APIs (described below)	
semantic_state	JSON	(optional) Semantic state of the device as described at https://developers.google.com/cloud-print/docs/cdd#cds. See examples below for details.	

api - is a JSON list containing the list of APIs available through the local network. Note that not all APIs may be available at the same time over the local network. For example, a newly connected device should only support the /register api:

```
"api": [
      "/privet/register",
]
```

Once device registration is complete, the device SHOULD stop supporting the /register API. The device should also check with the service to provide what APIs can be exposed over the local network. For example:

```
"api": [
      "/privet/accesstoken",
      "/privet/capabilities",
      "/privet/printer/submitdoc",
]
```

The following APIs are available at this time:

`/privet/register` - API for device registration over the local network. (see `/privet/register` API for details). This API MUST be hidden once the device is successfully registered in the cloud.

`/privet/accesstoken` - API to request access token from the device (see `/privet/accesstoken` API for details).

`/privet/capabilities` - API to retrieve device capabilities (see `/privet/capabilities` API for details).

`/privet/printer/*` - API specific to the device type "printer", see printer specific APIs for details.

Here is an example of the **`/privet/info`** response. (Note the lack of the `/privet/register` API, since this is already registered device).

```
{
      "version": "1.0",
      "name": "Gene's printer",
      "description": "Printer connected through Chrome connector",
      "url": "https://www.google.com/cloudprint",
      "type": [
            "printer"
      ],
      "id": "11111111-2222-3333-4444-555555555555",
      "device_state": "idle",
      "connection_state": "online",
      "manufacturer": "Google",
      "model": "Google Chrome",
      "serial_number": "1111-22222-33333-4444",
      "firmware": "24.0.1312.52",
      "uptime": 600,
      "setup_url": "http://support.google.com/cloudprint/answer/1686197/?hl=en",
      "support_url": "http://support.google.com/cloudprint/?hl=en",
      "update_url": "http://support.google.com/cloudprint/?hl=en",
      "x-privet-token": "AIp06DjQd80yMoGYuGmT_VDAApuBZbInsQ:1358377509659",
```

```
      "api": [
            "/privet/accesstoken",
            "/privet/capabilities",
            "/privet/printer/submitdoc",
      ]
}
```

Here is an example of the **`/privet/info`** response for printer that ran out of ink (notice `semantic_state` field).

```
{
      "version": "1.0",
      "name": "Gene's printer",
      "description": "Printer connected through Chrome connector",
      "url": "https://www.google.com/cloudprint",
      "type": [
            "printer"
      ],
      "id": "11111111-2222-3333-4444-555555555555",
      "device_state": "stopped",
      "connection_state": "online",
      "manufacturer": "Google",
      "model": "Google Chrome",
      "serial_number": "1111-22222-33333-4444",
      "firmware": "24.0.1312.52",
      "uptime": 600,
      "setup_url": "http://support.google.com/cloudprint/answer/1686197/?hl=en",
      "support_url": "http://support.google.com/cloudprint/?hl=en",
      "update_url": "http://support.google.com/cloudprint/?hl=en",
      "x-privet-token": "AIp06DjQd80yMoGYuGmT_VDAApuBZbInsQ:1358377509659",
      "api": [
            "/privet/accesstoken",
            "/privet/capabilities",
            "/privet/printer/submitdoc",
      ],
      "semantic_state": {
            "version": "1.0",
            "printer": {
                  "state": "STOPPED",
                  "marker_state": {
                        "item": [
                              {
                                    "vendor_id": "ink",
                                    "state": "EXHAUSTED",
                                    "level_percent": 0
                              }
                        ]
                  }
            }
      }
}
```

4.2.3. Errors

`/privet/info` API should ONLY return an error if “X-Privet-Token” header is missing. It MUST be HTTP 400 error.

```
HTTP/1.1 400 Missing X-Privet-Token header.
```

4.3. /privet/register API

`/privet/register` API is OPTIONAL. It is an HTTP **POST** request. `/privet/register` API MUST check for a valid “X-Privet-Token” header. Device MUST implement this API on “`/privet/register`” url:

```
POST /privet/register?action=start&user=user@domain.com HTTP/1.1
POST /privet/register?action=complete&user=user@domain.com HTTP/1.1
```

The device should expose `/privet/register` API ONLY when it allows anonymous registration at the moment. For example:

- When the device is turned on (or after clicking a special button on the device) and has not been registered yet, it should expose the `/privet/register` API to allow a user from the local network to claim the printer.
- After registration is complete, the device should stop exposing the `/privet/register` API to prevent another user on the local network from reclaiming the device.
- Some devices may have different ways to register devices and should not expose the `/privet/register` API at all (for example, Chrome Cloud Print connector).

The registration process consists of 3 steps (see anonymous registration for Cloud Print).

1. Initiate anonymous registration process.

A client initiates this process by calling the `/privet/register` API. The device may wait for user confirmation at that time.

2. Get claim token.

The client polls to find out when the device is ready to continue. Once the device is ready, it sends a request to the server to retrieve registration token and registration URL. Received token and URL SHOULD be returned to the client. During this step, if the device receives another call to initialize registration, it should:

- If this is the same user who started registration - drop all previous data (if any) and start a new registration process.
- If this is different user - return device_busy error and 30 seconds timeout.

Complete registration process.

After the client has claimed the device, the client should notify the device to complete registration. Once the registration process is complete, the device should send an update announcement, including the newly acquired device id.

Note: When the device is processing a `/privet/register` API call, no other `/privet/register` API calls may be processed simultaneously. The device MUST return the `device_busy` error and 30 seconds timeout.

User confirmation for registration on the device is HIGHLY recommended. If implemented, the device MUST

wait for user confirmation AFTER it receives a `/privet/register?action=start` API call. The client will be calling `/privet/register?action=getClaimToken` API to find out when user confirmation is complete and claim token is available. If the user cancels registration on the device (e.g. presses the Cancel button), the `user_cancel` error MUST be returned. If the user has not confirmed registration within a certain timeframe, the `confirmation_timeout` error MUST be returned. See defaults section for more details.

4.3.1. Input

`/privet/register` API has the following input parameters:

Name	Value
`action`	Can be one of the following: "**start**" - to start registration process "**getClaimToken**" - retrieve claim token for the device "**cancel**" - to cancel registration process "**complete**" - to complete registration process
`user`	Email of the user who will claim this device.

The device MUST check that the email address from all actions (start, getClaimToken, cancel, complete) matches.

4.3.2. Return

`/privet/register` API returns following data:

Value name	Value type	Description
`action`	string	Same action as in input parameter.
`user`	string (optional)	Same user as in input parameter (may be missing, if omitted in the input).
`token`	string (optional)	Registration token (mandatory for "getClaimToken" response, omitted for "start", "complete", "cancel").
`claim_url`	string (optional)	Registration URL (mandatory for "getClaimToken" response, omitted for "start", "complete", "cancel"). For Cloud Printers it must be the "complete_invite_url" received from the server.
`automated_claim_url`	string (optional)	Registration URL (mandatory for "getClaimToken" response, omitted for "start", "complete", "cancel"). For Cloud Printers it must be the "automated_invite_url" received from the server.
`device_id`	string (optional)	New device id (omitted for "start" response, mandatory for "complete").

The device MUST return its device id in the `/privet/info` API response ONLY after registration is complete.

Example 1:

```
{
      "action": "start",
      "user": "user@domain.com",
}
```

Example 2:

```
{
      "action": "getClaimToken",
      "user": "user@domain.com",
      "token": "AAA1112223333444555666777",
      "claim_url": "https://domain.com/SoMeUrL",
}
```

Example 3:

```
{
      "action": "complete",
      "user": "user@domain.com",
      "device_id": "11111111-2222-3333-4444-555555555555",
}
```

4.3.3. Errors

`/privet/register` API may return following errors (see Errors section for details):

Error	Description
`device_busy`	The device is busy and can't perform the requested action. Retry after timeout.
`pending_user_action`	In response to "getClaimToken" this error indicates that the device is still pending user confirmation, and "getClaimToken" request should be retrie after timeout.
`user_cancel`	User explicitly cancelled registration process from the device.
`confirmation_timeout`	User confirmation times out.
`invalid_action`	Invalid action is called. For example, if client called action=complete before calling action=start and action=getClaimToken.
`invalid_params`	Invalid parameters specified in the request. (Unknown parameters should be safely ignored for future compatibility). For example, return this if the client called action=unknown or user=.
`device_config_error`	Date/Time (or some other settings) is wrong on the device side. User need to go (to device internal website) and configure device settings.
`offline`	The device is currently offline and can't talk to the server.
`server_error`	Server error during registration process.

`invalid_x_privet_token`	X-Privet-Token is invalid or empty in the request.

The device MUST stop exposing the `/privet/register` API after registration has been successfully completed. If the device is not exposing the `/privet/register` API, it MUST return HTTP 404 error. Therefore, if a device is already registered, calling this API MUST return 404. If the X-Privet-Token header is missing, the device MUST return HTTP 400 error.

4.4. /privet/accesstoken API

`/privet/accesstoken` API is OPTIONAL. It is an HTTP **GET** request. `/privet/accesstoken` API MUST check for a valid "X-Privet-Token" header. The device MUST implement this API on the "`/privet/accesstoken`" url:

```
GET /privet/accesstoken HTTP/1.1
```

When the device receives the /accesstoken API call, it should call the server to retrieve the access token for the given user and return the token to the client. The client will then use the access token to access this device through the cloud.

Cloud Print devices MUST call the following API:

```
/cloudprint/proximitytoken
```

and pass "printerid=<printer_id>" and "user" parameter from the local API. If successful, the server response will contain the following object:

```
"proximity_token": {
      "user": "user@domain.com",
      "token": "AAA11122233344455566677",
      "expires_in": 600
}
```

Cloud Print devices MUST pass the value of the `"proximity_token"` object in the response to local `/privet/accesstoken` API calls. It is more advantageous (future-proof) if the device can pass ALL parameters (including ones that are not described in this spec).

4.4.1. Input

`/privet/accesstoken` API has following input parameters:

Name	Value
`user`	Email of the user who intended to use this access token. May be empty in th request.

4.4.2. Return

`/privet/accesstoken` API returns following data:

Value name	Value type	Description
token	string	Access token returned by the server
user	string	Same user as in input parameter.
expires_in	int	Number of seconds until this token expires. Received from the server and passed in this response.

Example:

```
{
      "token": "AAA1112223333444555666777",
      "user": "user@domain.com",
      "expires_in": 600
}
```

4.4.3. Errors

/privet/accesstoken API may return the following errors (see Errors section for details):

Error	Description
offline	Device is currently offline and can't talk to the server.
access_denied	Insufficient rights. Access denied. The device should return this error when the request has been explicitly denied by the server.
invalid_params	Invalid parameters specified in the request. (Unknown parameters shoul be safely ignored for future compatibility). For example, if client called /accesstoken?user= or /accesstoken.
server_error	Server error.
invalid_x_privet_token	X-Privet-Token is Invalid or empty in the request.

If device is not exposing the /privet/accesstoken API, it MUST return HTTP 404 error. If the X-Privet-Token header is missing, the device MUST return HTTP 400 error.

4.5. /privet/capabilities API

/privet/capabilities API is OPTIONAL. It is an HTTP **GET** request. /privet/capabilities API MUST check for a valid "X-Privet-Token" header. The device MUST implement this API on "/privet/capabilities" url:

```
GET /privet/capabilities HTTP/1.1
```

When the device receives /capabilities API call, if the device is able, it SHOULD contact the server to get updated capabilities. For example, if a printer supports posting a print job (received locally) to itself through the Cloud Print service, it should return capabilities that the Cloud Print service would return. Cloud Print in this case may alter the original printer capabilities by adding new features it may perform before sending job to the printer. The most common case is a list of supported document types. If the printer is offline, it should return

document types it supports. However, if the printer is online and registered with Cloud Print it MUST return “*/*” as one of the supported types. The Cloud Print service will perform the necessary conversion in this case. For offline printing, the printer MUST support at least the “image/pwg-raster” format.

4.5.1. Input

`/privet/capabilities` API has no input parameters.

4.5.2. Return

`/privet/capabilities` API returns device capabilities in the Cloud Device Description (CDD) JSON format (see the CDD document for details). Printers at minimum MUST return a list of supported types here. For examples, a Cloud Ready printer that is currently online may return something like this (at minimum):

```
{
      “version”: “1.0”,
      “printer”: {
            “supported_content_type”: [
                  {
                        “content_type”: “application/pdf”,
                        “min_version”: “1.4”
                  },
                  { “content_type”: “image/pwg-raster” },
                  { “content_type”: “image/jpeg” },
                  { “content_type”: “*/*” }
            ]
      }
}
```

and when it’s disconnected from the server, it may return:

```
{
      “version”: “1.0”,
      “printer”: {
            “supported_content_type”: [
                  {
                        “content_type”: “application/pdf”,
                        “min_version”: “1.4”
                  },
                  { “content_type”: “image/pwg-raster” },
                  { “content_type”: “image/jpeg” }
            ]
      }
}
```

Note: Printers express supported content type priority using order. For example, in the samples above, the printer specifies that it prefers “application/pdf” data over “image/pwg-raster” and “image/jpeg”. Clients should respect printer prioritization if possible (see the CDD document for details).

4.5.3. Errors

`/privet/capabilities` API may return following errors (see Errors section for details):

Error	Description
`invalid_x_privet_token`	X-Privet-Token is invalid or empty in the request.

If the device is not exposing the `/privet/capabilities` API, it MUST return HTTP 404 error. If the X-Privet-Token header is missing, the device MUST return HTTP 400 error.

4.6. Errors

Errors are returned from the above APIs in the following format:

Value name	Value type	Description
`error`	string	Error type (defined per API)
`description`	string (optional)	Human readable description of the error.
`server_api`	string (optional)	In case of server error, this field contains the server API that failed.
`server_code`	int (optional)	In case of server error, this field contains that error code that the server returned.
`server_http_code`	int (optional)	In case of server HTTP error, this field contains HTTP error code server returned.
`timeout`	int (optional)	Number of seconds for client to wait before retrying (for recoverable errors only). Client MUST randomize actual timeout from this value to a value that is + 20%.

All APIs MUST return HTTP 400 error if X-Privet-Token header is missing.

```
HTTP/1.1 400 Missing X-Privet-Token header.
```

Example 1:

```
{
      "error": "server_error",
      "description": "Service unavailable",
      "server_api": "/submit",
      "server_http_code": 503
}
```

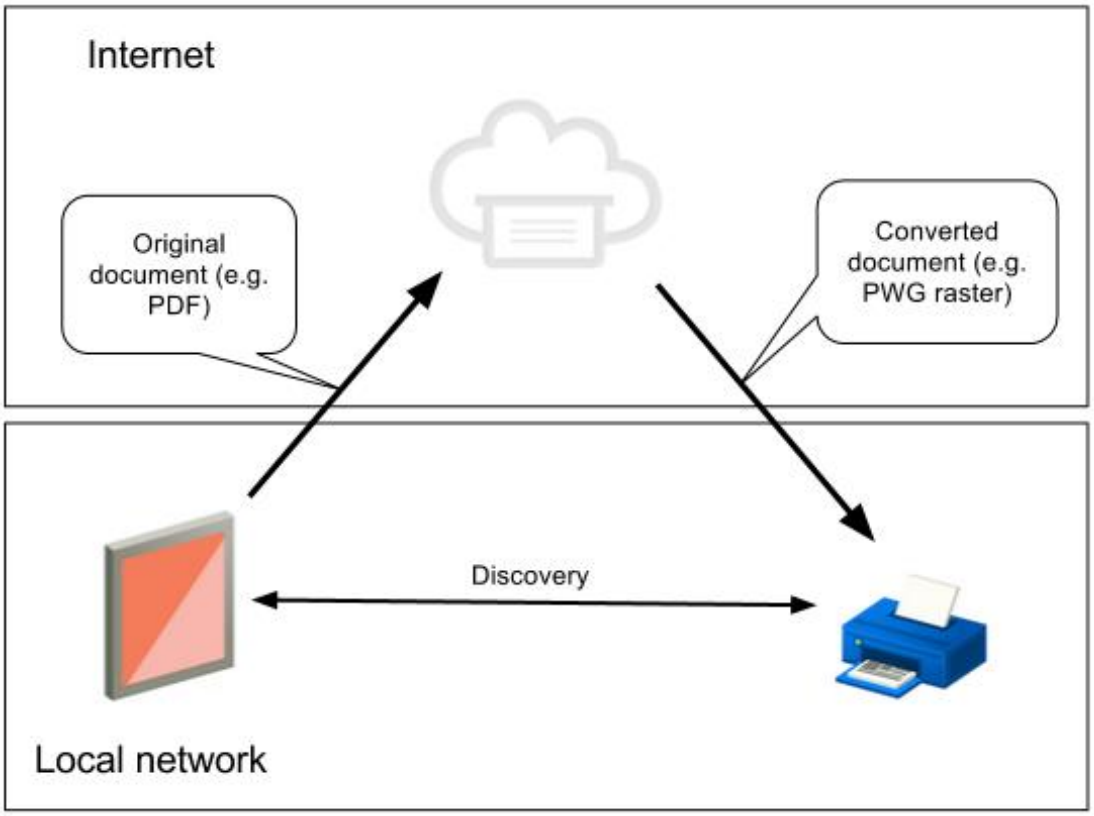
Example 2:

```
{
      "error": "printer_busy",
      "description": "Printer is currently printing other job",
```

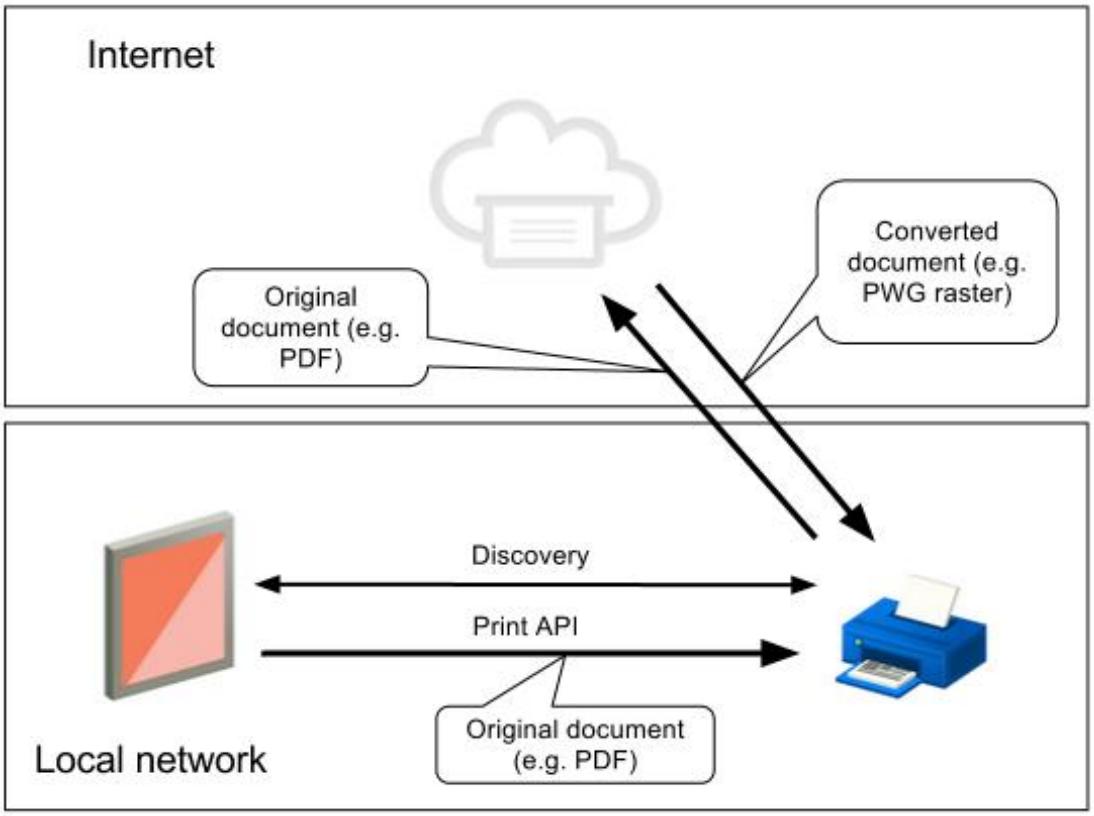
```
      "timeout": 15
}
```

5. Printer API

One of the device types this protocol supports is type **printer**. Devices supporting this type MAY implement some functionality specific to printers. Ideally, printing to cloud-ready printers will go through a Cloud Print server:



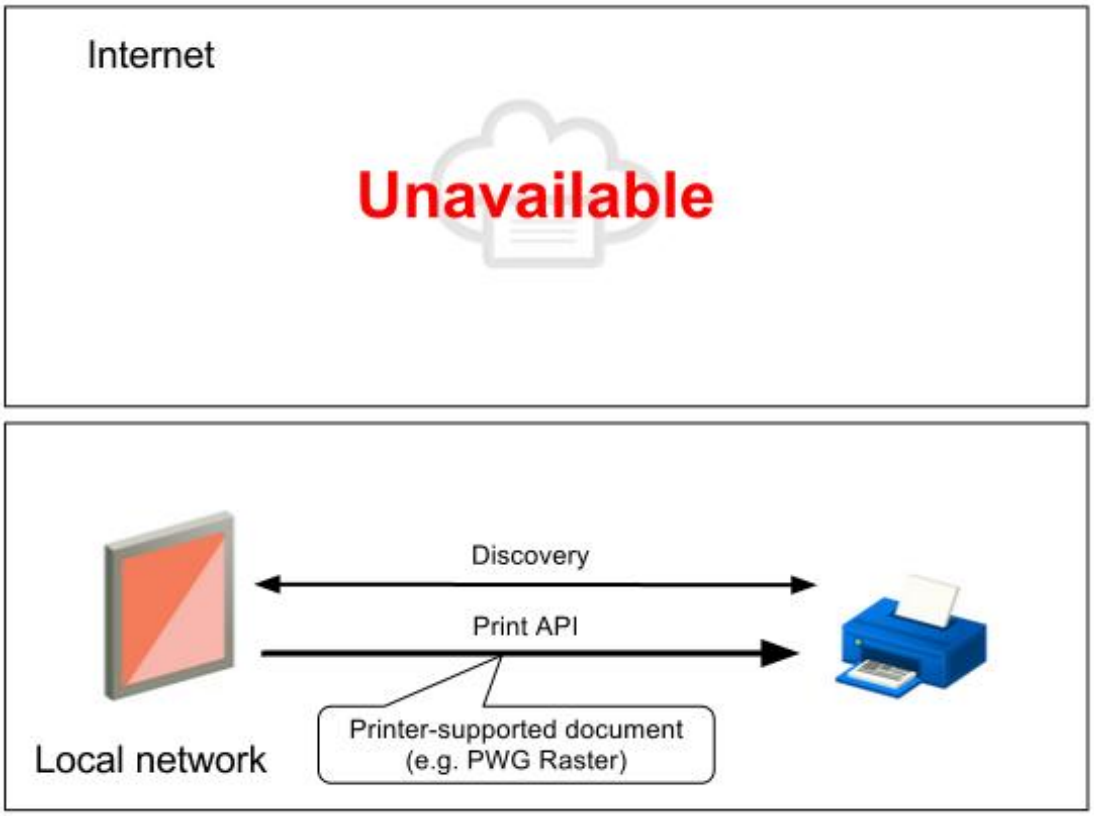
In some cases a client may need to send a document locally. It may be needed when client does not have a Google ID or is unable to talk to the Cloud Print server. In such case, the print job will be submitted locally to the printer. The printer, in turn, will use the Cloud Print service for job queueing and conversion. The printer will re-post the job submitted locally to the Cloud Print service and then request it, since it was submitted through the cloud. This process will provide a flexible user experience in terms of service (conversion) and print job management/tracking.



Since the Cloud Print service implements conversion, the printer SHOULD advertise supporting all input formats (“*/*”) among the list of the supported content types:

```
{
      “version”: “1.0”,
      “printer”: {
            “supported_content_type”: [
                  { “content_type”: “image/pwg-raster” },
                  { “content_type”: “*/*” }
            ]
      }
}
```

In some cases a completely offline solution is desired. Since printers support a limited number of input formats, a client will need to convert documents to a few natively supported printer formats.



This spec **REQUIRES** all printers to support at least the PWG Raster (“image/pwg-raster”) format for the offline printing case. A printer may support other formats (for example JPEG) and if a client supports it, it may send documents in that format. The printer MUST expose supported types through the /capabilities API, for example:

```
{
      “version”: “1.0”,
      “printer”: {
            “supported_content_type”: [
                  { “content_type”: “image/pwg-raster” },
                  { “content_type”: “image/jpeg” }
            ]
      }
}
```

There are two ways a client may initiate printing over the local network.

Simple printing - client sends the document over the local network to `/submitdoc` API (without specifying the `job_id` parameter). The submitted document will be printed using default print ticket settings and no print job statuses are needed. If the printer ONLY supports this type of printing, it MUST advertise ONLY `/submitdoc` API in the `/privet/info` API response.

```
      “api”: [
            “/privet/accesstoken”,
            “/privet/capabilities”,
            “/privet/printer/submitdoc”,
      ]
```

Advanced printing - client should first create a print job on the printer by calling the `/privet/printer/createjob` API with a valid CJT job ticket in the request. The printer MUST store the print ticket in memory and return a `job_id` back to the client. Then the client will call the /printer/submitdoc API and specify the previously received `job_id`. At that time the printer will start printing. The client will poll the printer for print job status by calling the `/privet/printer/jobstate` API.

In a multi-client environment, there is no guarantee how this API is called. It is possible for one client to call `/createjob` between another client's `/createjob->/submitdoc` calls. To eliminate possible deadlocks and improve usability, we recommended having a small queue of pending print jobs on the printer (at least 3-5):

- `/createjob` takes the first available spot in the queue.
- Job lifetime (in the queue) is at least 5 minutes.
- If all spots in the queue are taken, then the oldest, non-printing job shall be removed and a new one will be placed there.
- If there is a print job currently printing on the device (simple or advanced printing), `/submitdoc` should return status busy and propose a timeout to retry this print job.
- If `/submitdoc` refers to a job that has been removed from the queue (due to replacement or timeout), the printer should return an error `invalid_print_job` and the client will retry the process from the `/createjob` step. The client MUST wait for a random timeout period of up to 5 seconds before retrying.

If memory constraints prevent storing multiple pending jobs on the device, it is possible to have a queue of 1 print job long. It should still follow the same protocol as above.
After a job has completed or failed with an error, the printer should store information about the job's status for at least 5 minutes. The queue size for storing completed job statuses should be at least 10. If there are more job statuses that need to be stored, the oldest one may be removed from the queue before the 5 minute timeout.

<For now clients will poll for job status. In the future, we may require the printer to send TXT DNS notification when ANY print job status has changed.>

5.1. /privet/printer/createjob API

`/privet/printer/createjob` API is OPTIONAL (see Simple Printing above). It is an HTTP **POST** request. `/privet/printer/createjob` API MUST check for a valid "X-Privet-Token" header. The device MUST implement this API on "`/privet/printer/createjob`" url:

```
POST /privet/printer/createjob HTTP/1.1
```

When receiving `/privet/printer/createjob` API call, the printer MUST create a new print job ID, store the received print ticket in the CJT format, and return print job id back to the client.

5.1.1. Input

`/privet/printer/createjob` API has no input parameters in URL. The request body should contain the print job ticket data in CJT format.

5.1.2. Return

`/privet/printer/createjob` API returns the following data:

Value name	Value type	Description
job_id	string	ID of the newly created print job.
expires_in	int	Number of seconds this print job is valid.

Example:

```
{
      "job_id": "123",
      "expires_in": 600
}
```

5.1.3. Errors

/privet/printer/createjob API may return the following errors (see Errors section for details):

Error	Description
invalid_ticket	Submitted print ticket is invalid.
printer_busy	Printer is busy and can't currently process /createjob. Retry after timeout.
printer_error	Printer is in error state and requires user interaction to fix it. Description should contain more detailed explanation (e.g. "Paper jam in Tray 1").
invalid_x_privet_token	X-Privet-Token is invalid or empty in the request.

If device is not exposing /privet/printer/createjob, it MUST return HTTP 404 error. If X-Privet-Token header is missing, the device MUST return HTTP 400 error.

5.2. /privet/printer/submitdoc API

/privet/printer/submitdoc API is REQUIRED to implement printing over a local network (offline or repost to Cloud Print). It is an HTTP **POST** request. /privet/printer/submitdoc API MUST check for a valid "X-Privet-Token" header. The device MUST implement this API on "/privet/printer/submitdoc" url:

```
POST /privet/printer/submitdoc HTTP/1.1
```

When receiving the /privet/printer/submitdoc API call, the printer should start printing. If it is unable to begin printing, it MUST return the error printer_busy and a recommended timeout period for the client to wait before trying again.

If the printer is not able to hold all of the data in its internal buffer, it SHOULD use TCP mechanisms to slow down data transfer until it prints a portion of the document, making part of the buffer available again. (For example, the printer may set windowsize=0 on TCP layers, which will make the client wait.)

Submitting a document to the printer may take a significant amount of time. The client should be able to check

the state of the printer and job (advanced printing) while printing is in progress. In order to do that, the printer MUST allow the client to call the `/privet/info` and `/privet/printer/jobstate` APIs while processing `/privet/printer/submitdoc` API calls. It is recommended for all clients to start a new thread to execute the `/privet/printer/submitdoc` API call, so that the main thread can use the `/privet/info` and `/privet/printer/jobstate` APIs to check printer and job states.

Note: Upon completion or abortion of the local print job, it is strongly recommended (and will be required in a future version of this spec) to report the final state of the job to the `/cloudprint/submit` API for accounting and user experience purposes. The parameters “printerid”, “title”, “contentType” and “final_semantic_state” (in PrintJobState format, defined at https://developers.google.com/cloud-print/docs/cdd#pjs) are required, and the parameters “tag” (repeated parameter) and “ticket” (the ticket of the job in CloudJobTicket format, defined at https://developers.google.com/cloud-print/docs/cdd#cjt) are optional. Note that the provided PrintJobState must actually be final, i.e. its type must be DONE or ABORTED, and a cause must be provided in the case that it is ABORTED (see https://developers.google.com/cloud-print/docs/cdd#JobState for details).

5.2.1. Input

`/privet/printer/submitdoc` API has the following input parameters:

Name	Value
`job_id`	(optional) Print job id. May be omitted for simple printing case (see above). Must match the one returned by the printer.
`user_name`	(optional) Human readable user name. This is not definitive, and should only be used for print job annotations. If job is re-posted to the Cloud Print service this string should be attached to the Cloud Print job.
`client_name`	(optional) Name of the client application making this request. For display purposes only. If job is re-posted to the Cloud Print service this string should be attached to the Cloud Print job.
`job_name`	(optional) Name of the print job to be recorded. If job is re-posted to the Clou Print service this string should be attached to the Cloud Print job.
`offline`	(optional) Could only be “offline=1”. In this case printer should only try printin offline (no re-post to Cloud Print server).

Request body should contain a valid document for printing. “Content-Length” should include the correct length of the request. “Content-Type” header should be set to document MIME type and match one of the types in the CDD (unless CDD specifies “*/*”).

Clients are HIGHLY recommended to provide a valid user name (or email), a client name and a job name with this request. Those fields are only used in UIs to improve user experience.

5.2.2. Return

`/privet/printer/submitdoc` API returns following data:

Value name	Value type	Description

job_id	string	ID of the newly created print job (simple printing) or job_id specified in the request (advanced printing).
expires_in	int	Number of seconds this print job is valid.
job_type	string	Content-type of the submitted document.
job_size	int 64 bit	Size of the print data in bytes.
job_name	string	(optional) Same job name as in input (if any).

Example:

```
{
      "job_id": "123",
      "expires_in": 500,
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document"
}
```

5.2.3. Errors

/privet/printer/submitdoc API may return the following errors (see Errors section for details):

Error	Description
invalid_print_job	Invalid/expired job id is specified in the request. Retry after timeout.
invalid_document_type	Document MIME-type is not supported by the printer.
invalid_document	Submitted document is invalid.
document_too_large	Document exceeds maximum size allowed.
printer_busy	Printer is busy and can't currently process document. Retry after timeout.
printer_error	Printer is in error state and requires user interaction to fix it. Description should contain more detailed explanation (e.g. "Paper jam in Tray 1").
invalid_params	Invalid parameters specified in the request. (Unknown parameters should be safely ignored for future compatibility)
user_cancel	User explicitly cancelled prinitng process from the device.
server_error	Posting document to Cloud Print has failed.
invalid_x_privet_token	X-Privet-Token is invalid or empty in the request.

If the device is not exposing /privet/printer/submitdoc, it MUST return HTTP 404 error. If the X-Privet-Token header is missing, the device MUST return HTTP 400 error.

Note: `/privet/printer/submitdoc` API may require special handling on printer side (because of the large payload attached). In some cases (depends on the printer HTTP server implementation and platform), printer may close socket BEFORE returning HTTP error. In other, printer may return 503 error (instead of Privet error). Printers SHOULD try as much as possible to return Privet. However, every client implementing Privet specification SHOULD be able to handle socket close (no HTTP error) and 503 HTTP error cases for `/privet/printer/submitdoc` API. In this case, client SHOULD handle it as a Privet "`printer_busy`" error with "`timeout`" set to 15 seconds. To avoid infinite retries, client may stop retrying after a reasonable number of attempts (for example, 3).

5.3. /privet/printer/jobstate API

`/privet/printer/jobstate` API is OPTIONAL (see Simple Printing above). It is an HTTP **GET** request. `/privet/printer/jobstate` API MUST check for a valid "X-Privet-Token" header. The device MUST implement this API on "`/privet/printer/jobstate`" url:

```
GET /privet/printer/jobstate HTTP/1.1
```

When receiving a `/privet/printer/jobstate` API call, a printer should return the status of the requested print job or `invalid_print_job` error.

5.3.1. Input

`/privet/printer/jobstate` API has following input parameters:

Name	Value
`job_id`	Print job ID to return status for.

5.3.2. Return

`/privet/printer/jobstate` API returns following data:

Value name	Value type	Description
`job_id`	string	Print job id there status information is for.
`state`	string	"**`draft`**" - print job has been created on the device (no `/privet/printer/submitdoc` calls have been received yet). "**`draft`**" - (not used now) this state is for conformance with CDS spec and may be used in the future. "**`queued`**" - print job has been received and queued, but printing has not started yet. "**`in_progress`**" - print job is in the progress of printing. "**`stopped`**" - print job has been paused, but can be restarted manually or automatically. "**`done`**" - print job is done. "**`aborted`**" - print job failed.

description	string	(optional) Human readable description of the print job status. Should include additional information if state is "stopped" or "aborted". The semantic_state field usually provides better and more meaningful description to the client.
expires_in	int	Number of seconds this print job is valid.
job_type	string	(optional) Content-type of the submitted document.
job_size	int 64 bit	(optional) Size of the print data in bytes.
job_name	string	(optional) Same job name as in input (if any).
server_job_id	string	(optional) ID of the job returned from the server (if job has been posted to Cloud Print service). Omitted for offline printing.
semantic_state	JSON	(optional) Semantic state of the job as described at https://developers.google.com/cloud-print/docs/cdd#pjs. See examples below for details.

Example (printing by reporting through Cloud Print):

```
{
      "job_id": "123",
      "state": "in_progress",
      "expires_in": 100,
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document",
      "server_job_id": "1111-2222-3333-4444"
}
```

Example (offline printing error):

```
{
      "job_id": "123",
      "state": "stopped",
      "description": "Out of paper",
      "expires_in": 100,
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document"
}
```

Example (print job aborted by the user):

```
{
      "job_id": "123",
```

```
      "state": "aborted",
      "description": "User action",
      "expires_in": 100,
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document",
      "semantic_state": {
            "version": "1.0",
            "state": {
                  "type": "ABORTED",
                  "user_action_cause": {"action_code": "CANCELLED"}
            },
            "pages_printed": 7
      }
}
```

Example (print job stopped due to out of paper). Notice the reference to the device state. The client will need to call the /privet/info API to get more details about the device state:

```
{
      "job_id": "123",
      "state": "stopped",
      "description": "Out of paper",
      "expires_in": 100,
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document",
      "semantic_state": {
            "version": "1.0",
            "state": {
                  "type": "STOPPED",
                  "device_state_cause": {"error_code": "INPUT_TRAY"}
            },
            "pages_printed": 7
      }
}
```

5.3.3. Errors

`/privet/printer/jobstate` API may return the following errors (see Errors section for details):

Error	Description
`invalid_print_job`	Invalid/expired job ID is specified in the request.
`server_error`	Getting print job status (for print jobs posted to Cloud Print) has failed.
`invalid_x_privet_token`	X-Privet-Token is invalid or empty in the request.

If device is not exposing `/privet/printer/jobstate`, it MUST return HTTP 404 error. If the X-Privet-Token

header is missing, the device MUST return HTTP 400 error.

6. Appendix

6.1. Default behavior and settings

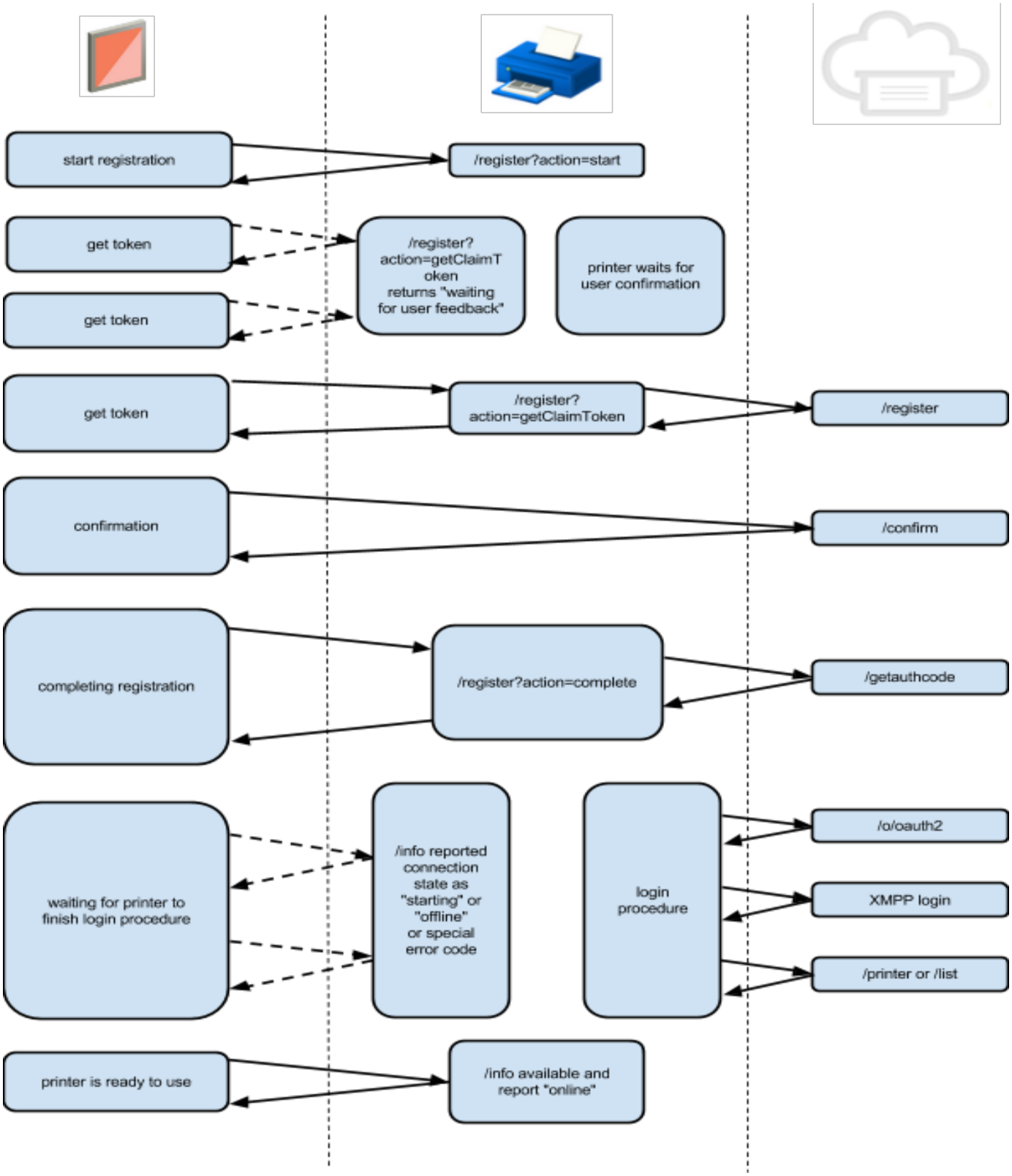
This section will explain the default behavior we expect from ALL Privet-compatible devices.

- Out-of-the-box devices should support only `/privet/info` and `/privet/register` APIs. All other APIs (e.g. `/privet/accesstoken`, local printing) should be disabled.
- Registration requires physical interaction with a device.
    - User MUST take a physical action on the device (e.g., pressing a button) to confirm his/her access to the device.
    - After the user takes the action noted above, the printer should send the /cloudprint/register request. It should not send this request until after the action is taken (see Sequence Diagram 1).
    - If the device is processing a `/privet/register` request (for instance, waiting on the action above), it must reject all other `/privet/register` requests. The device MUST return the `device_busy` error in this case.
    - The device should timeout any /register request that does not receive the physical action mentioned above within 60 seconds. The device MUST return the `confirmation_timeout` error in this case.
    - Optional: Recommended but not required, the following may improve user experience:
        - The printer might flash a light or its screen to indicate that the user needs to take an action to confirm registration.
        - The printer might state on its screen that 'it is being registered to Google Cloud Print for user 'abc@def.com' - press OK to continue', where abc@def.com is the `user` parameter from the /register API call. This would make it clearer to a user that a) it is his/her registration request that s/he is confirming, and b) what is happening if s/he didn't trigger the request.
        - In addition to a physical action to confirm from the printer (e.g., 'Press the OK button'), a printer may also offer the user a button to cancel the request (e.g., 'Press Cancel to reject'). This would allow users who did not trigger the registration request to cancel it before the 60 second timeout. The device MUST return the `user_cancel` error in this case.
- Ownership transfers:
    - The device may be deleted explicitly from the Cloud service.
        - If the device receives success, but no device description as a result of /cloudprint/printer (for GCP) call, it MUST revert to default (out-of-the-box) mode.
        - If the device's credentials no longer work (explicitly because of "invalid credentials" response from the server), it MUST revert to default (out-of-the-box) mode.
    - Local factory reset MUST clear device's credentials and set it to default state.
    - Optional: The device may provide a menu item to clear credentials and put it into default mode.
- Devices supporting XMPP notifications MUST include the ability to ping the server. The ping timeout MUST be controllable from the server through "local_settings".
- The device may explicitly ping the server (/cloudprint/printer API for GCP, in addition to XMPP pings) no more often than once a day (24 hours) to make sure they are in sync. It is recommended to randomize the check window within 24-32 hour window.
- Optional: For Cloud Print devices, it is recommended but not required to have a manual way (button) to

allow the user to initiate a check for new print jobs from the device. Some printers already have this.

- Optional. Enterprise printers may have an option to disable local discovery completely. In such case, the device MUST update these local settings on the server. New local settings MUST be empty (setting “local_discovery” to “false”, means that it can be re-enabled from the GCP Service).

6.1.2 Default Registration Diagram



6.2. XSSI and XSRF attacks and prevention

This section will explain the possibility of XSSI and XSRF attacks on the device and how to protect from them (including token generation techniques).

More details are here: http://googleonlinesecurity.blogspot.com/2011/05/website-security-for-webmasters.html

Normally, XSSI and XSRF attacks are possible when a site is using cookie authentication mechanisms. While Google doesn't use cookies with their Cloud Print Service, such attacks are still possible. Local network access, by design, implicitly trusts requests.

6.2.1. XSSI

It is possible for a malicious website to guess the IP address and port number of a Privet-compatible device and to try to call the Privet API using "`src=<api name>`" inside of a <script> tag:

```
<script type="text/javascript"
src="http://192.168.1.42:8080/privet/info"></script>
```

Without protection, malicious websites would be able to execute API calls and access results.

To prevent this type of attack, **ALL** Privet API calls **MUST** require the "**X-Privet-Token**" header in the request. "src=<api>" script tags are not able to add headers, effectively guarding against this type of attack.

6.2.2. XSRF

http://en.wikipedia.org/wiki/Cross-site_request_forgery
It is possible for a malicious website to guess the IP address and port number of a Privet-compatible device and try to call Privet API using an <iframe>, forms, or some other cross-website loading mechanism. Attackers would not be able to access the results of the request, but if the request would perform an action (e.g. printing), they could trigger it.

To prevent this attack, we require the following protection:

- Leave `/privet/info` API open to XSRF
- `/privet/info` API **MUST NOT** perform any actions on the device
- Use `/privet/info` API to receive `x-privet-token`
- All other APIs MUST check for a valid `x-privet-token` in "X-Privet-Token" header.
- `x-privet-token` SHOULD be valid for only 24 hours.

Even if an attacker is able to execute the `/privet/info` API, they would not be able to read `x-privet-token` from the response and therefore would not be able to call any other API.

It is strongly recommended to generate the XSRF token using the following algorithm:

```
XSRF_token = base64( SHA1(device_secret + DELIMITER + issue_timecounter) +
DELIMITER + issue_timecounter )
```

XSRF Token Generation Elements:

- `DELIMITER` is a special character, usually '**:**'
- `issue_timecounter` is a number of seconds since some event (epoch for timestamp) or device boot time (for CPU counters). `issue_timecounter` is constantly increasing when the device is up and running (*see token verification below)*.
- `SHA1` - hash function using SHA1 algorithm
- `base64` - base64 encoding
- `device_secret` - secret specific to the device. Device secret MUST be updated on every restart.

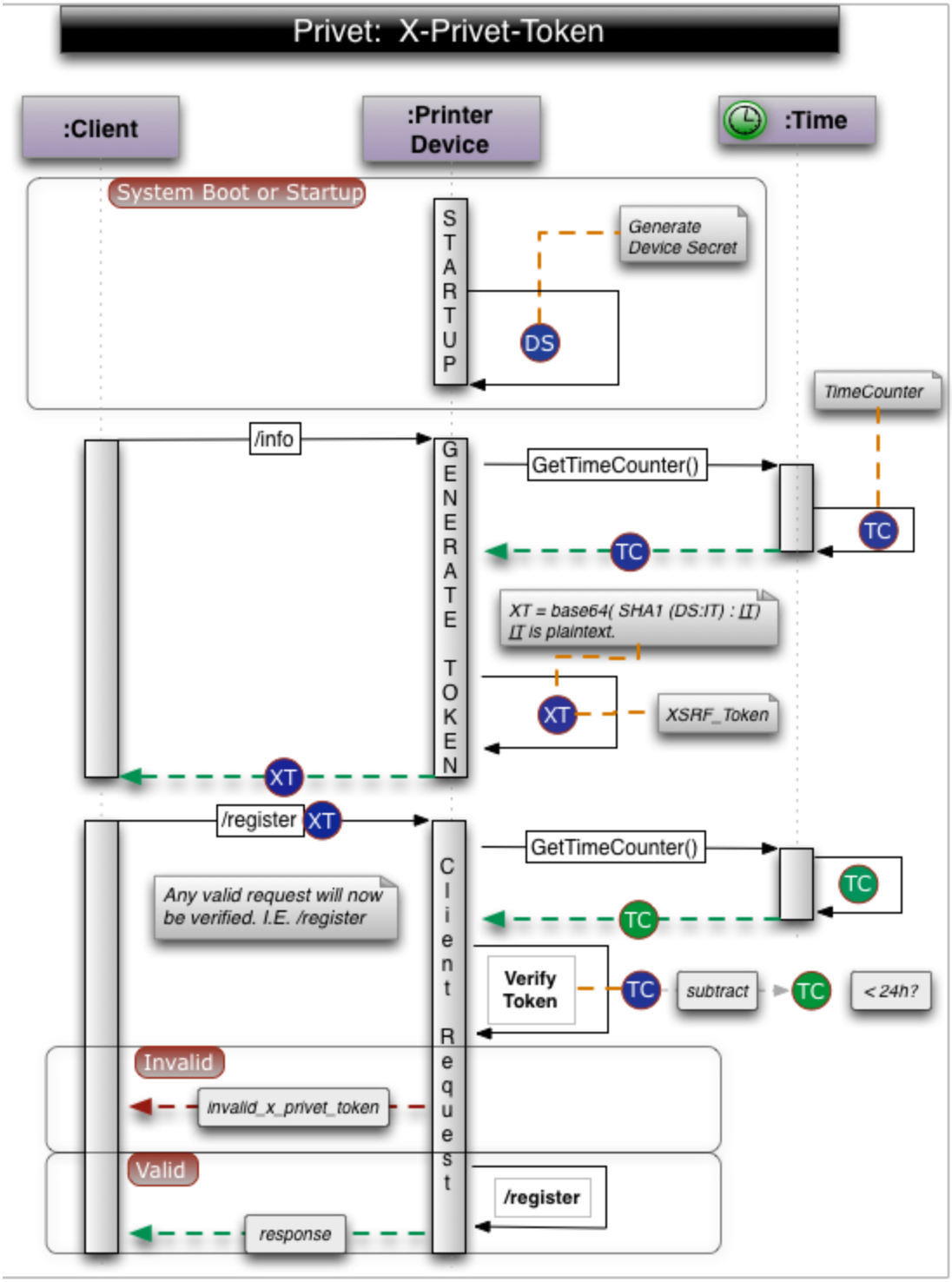
Recommended ways to generate device secret:

- Generate a new UUID on every restart
- Generate a 64 bit random number on every restart

The device is not required to store all of the XSRF tokens it has issued. When the device needs to verify a XSRF token for validity, it should base64-decode the token. Get the `issue_timecounter` from the second half (cleartext), and try to produce SHA1 hash of `device_secret + DELIMITER + issue_timecounter` where `issue_timecounter` is from the token. If the newly generated SHA1 matches the one in the token, device must now check if the `issue_timecounter` is within the validity period from (24 hours) of the current time counter. To do so, device takes the current time counter (CPU counter for example) and subtracts `issue_timecounter` from it. The result MUST be the number of seconds since token issue.

This is the recommended way to implement XSRF protection. Clients of the Privet specification shall not try to understand XSRF token, instead they shall treat is as a blackbox. Figure 6.2.3 illustrates a recommended way to implement the X-Privet-Token and verification of a typical request.

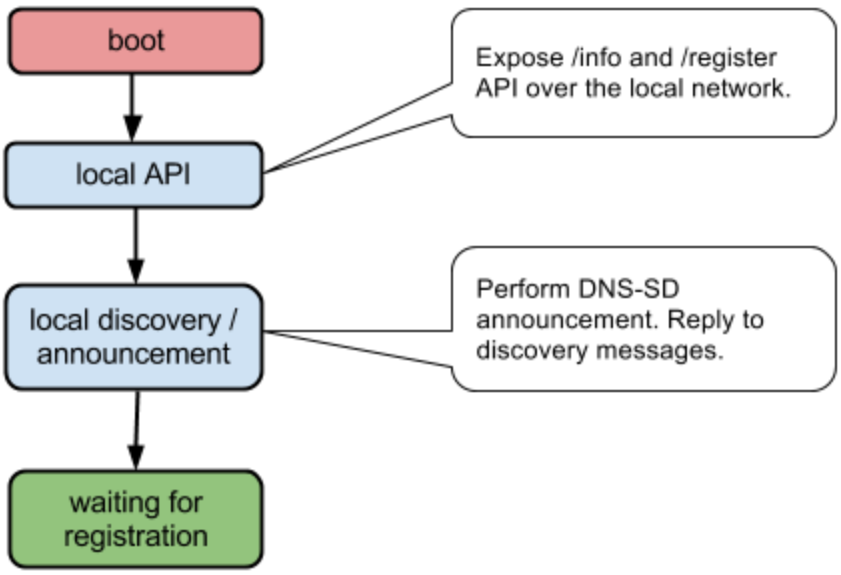
6.2.3 X-Privet Token Generation and Verification Sequence Diagram



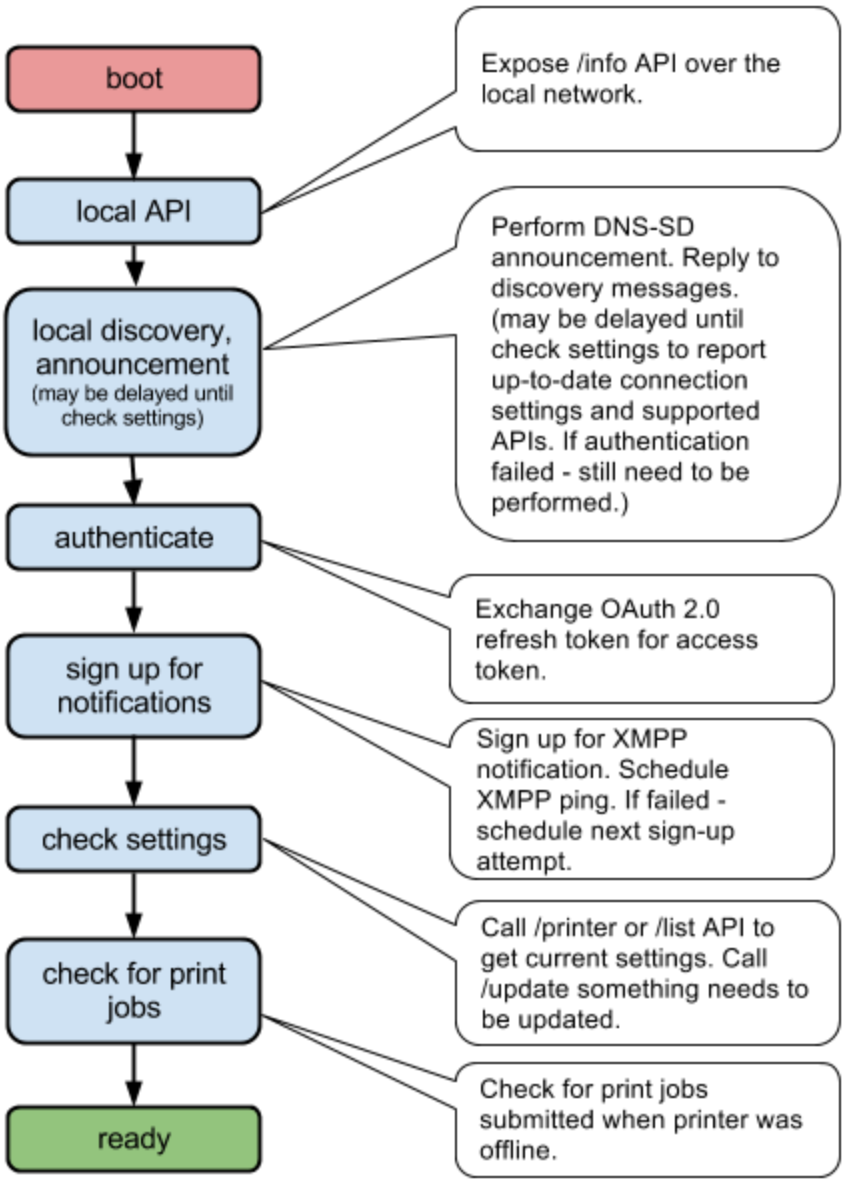
6.3. Workflow diagrams

This section will illustrate a workflow in different cases.

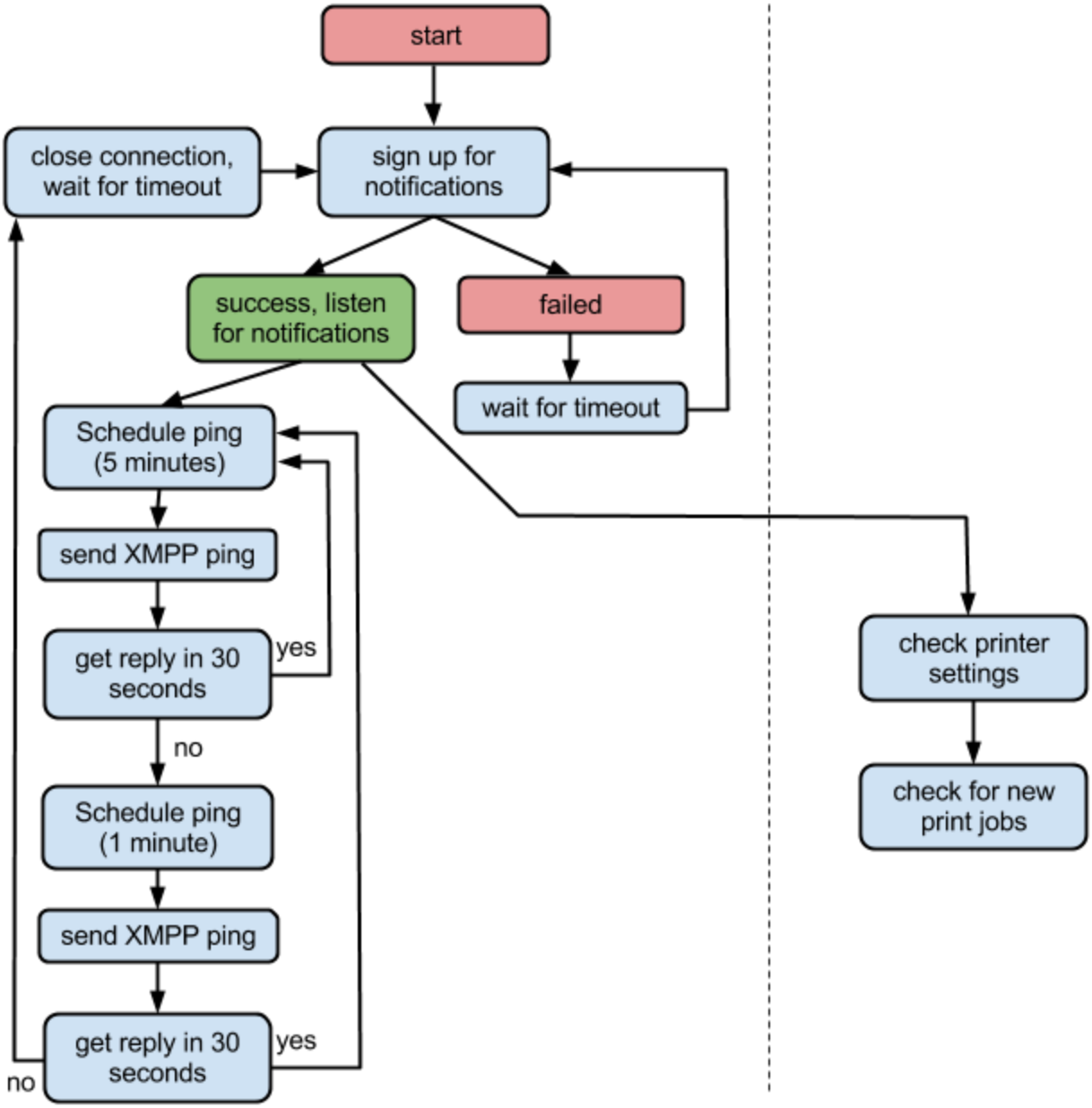
6.3.1. Printer out of the box workflow:



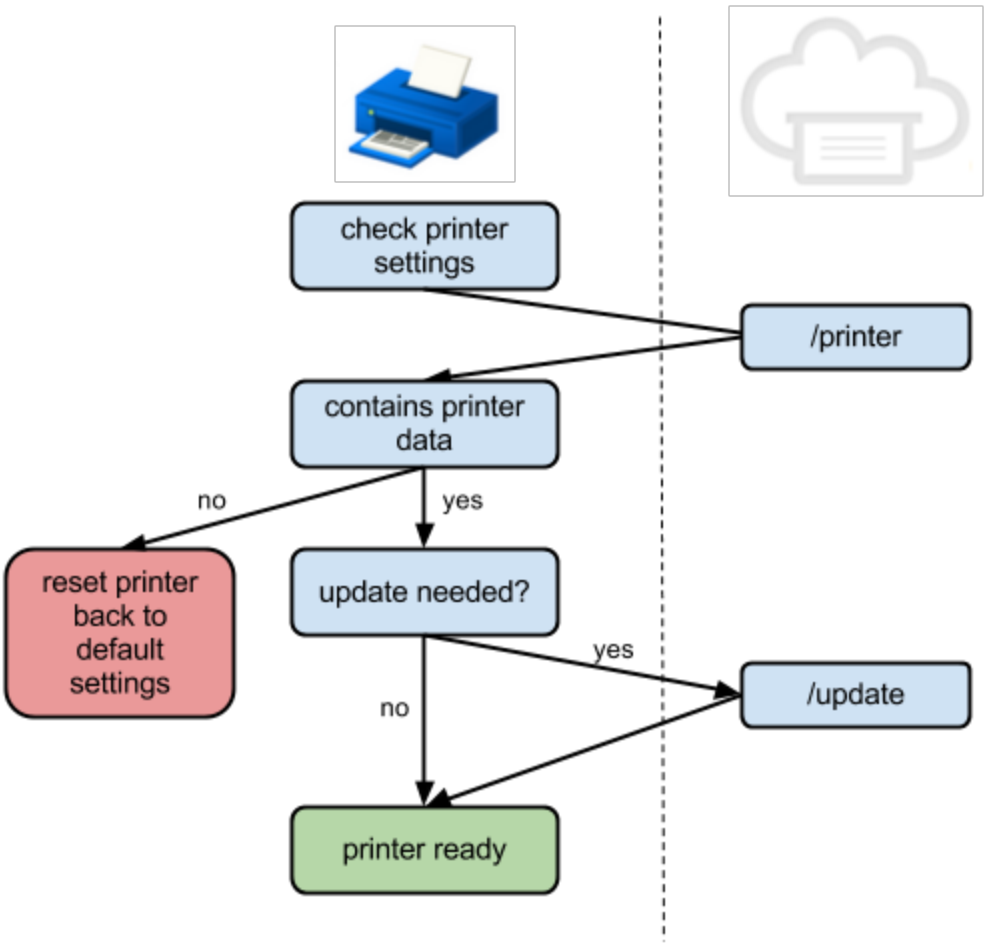
6.3.2. Registered printer startup workflow:



6.3.3. XMPP notifications handling workflow:



6.3.4. Check printer settings workflow:



7. Document History

- 04/05/2013 (gene): Changed some field reported by /info API.
- 04/05/2013 (gene): Changed section 6.2.2. XSRF to time counters instead of timestamps.
- 04/05/2013 (gene): Added 6.3 Workflow diagram section
- 04/08/2013 (kdlucas): Added 6.2.3 X-Privet-Token Generation/Verification diagram.
- 04/17/2013 (gene): Added comment about claim_url for Cloud Printers.
- 05/07/2013 (gene): Adding new state to the /info API for printer (connecting)
- 05/07/2013 (gene): Adding new action to register API (getClaimToken, cancel)
- 05/07/2013 (gene): Adding /privet/ prefix to all APIs
- 05/07/2013 (gene): Removed secure printing part to avoid confusion
- 05/24/2013 (gene): Adding new error codes for /createjob
- 05/31/2013 (gene): Explained simultaneous access to /submitdoc and /info, /jobstate APIs in the specification.
- 06/02/2013 (gene): Added semantic_state fields to /info and /jobstate APIs, to allow printers to express state of the device and job according to the Semantic States spec.
- 06/14/2013 (gene): Minor typo fix (invalid_print_job vs. unknown_job_id).
- 06/14/2013 (gene): Clarified return value “description” for /jobstate API.

- 06/14/2013 (gene): Added job state “draft” for compatibility with Semantic States spec.
- 06/25/2013 (gene): Added new error for registration (invalid_action).
- 07/02/2013 (gene): Added “printer” prefix to “local_printing_enabled” field in local settings.
- 07/18/2013 (gene): Clarified registration steps in 4.3 (handling multiple registration requests).
- 07/22/2013 (gene): Added “automated_claim_url” to the /register response.
- 07/30/2013 (gene): Added “invalid_params” error for /register and other APIs.
- 07/31/2013 (gene): Added recommendation for client behavior calling /info API with X-Privet-Token.
- 09/09/2013 (jsg): Fixed lots of grammar errors and a few correctness errors.
- 09/09/2013 (jsg): Updated CDS and CJS examples to comply with the latest Semantic States spec.
- 11/01/2013 (jsg): Updated CJS examples to comply with the latest Semantic States spec.
- 11/05/2013 (gene): Updated CDD example to comply with the latest CDD spec.
- 11/05/2013 (gene): Clarified printer/client error handling for /submitdoc API.
- 11/15/2013 (gene): Added “user cancel” error for /submitdoc API.
- 11/15/2013 (gene): Changed defaults for local printing and access token to true.
- 12/10/2013 (gene): Added error for device internal date/time config error.
- 12/10/2013 (gene): Added printer/conversion_printing_enabled to local_settings.
- 12/15/2013 (jsg): Added strong recommendation to report completed and aborted local print jobs to the server using the /cloudprint/submit API (see section 5.2 of this document).