This is document is a draft. If you have any suggestions on local discovery, announcements, API or anythin
else, please let us know. Specifically, we would like feedback on:

- DNS discovery and announcements format
- Local printing API
- Difficulty in implementation

Privet

(Cloud Device Local Discovery/API protocol)

1. Introduction
2. Discovery
    2.1. Service Type
    2.2. TXT record
        2.2.1. txtvers
        2.2.2. ty
        2.2.3. note (optional)
        2.2.4. base_url
        2.2.5. type
        2.2.6. id
        2.2.7. ds
        2.2.8. cs
3. Announcements
    3.1. Startup
    3.2. Shutdown
    3.3. Update
4. API
    4.1. /info API
        4.1.1. Input
        4.1.2. Return
        4.1.3. Errors
    4.2. /register API
        4.2.1. Input
        4.2.2. Return
        4.2.3. Errors
    4.3. /accesstoken API
        4.3.1. Input
        4.3.2. Return
        4.3.3. Errors
    4.4. /capabilities API

- 4.4.1. Input
- 4.4.2. Return
- 4.4.3. Errors
- 4.5. Errors
- 5. Printer API
- 5.1. /printer/createjob API
- 5.1.1. Input
- 5.1.2. Return
- 5.1.3. Errors
- 5.2. /printer/submitdoc API
- 5.2.1. Input
- 5.2.2. Return
- 5.2.3. Errors
- 5.3. /printer/jobstate API
- 5.3.1. Input
- 5.3.2. Return
- 5.3.3. Errors
- 6. Appendix
- 6.1. XSSI and XSRF attacks and prevention
- 6.1.1. XSSI
- 6.1.2. XSRF
- 6.2. Secure printing over local network (TBD)

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

Privet protocol consist of 2 main parts: discovery and API. Discovery is used to find the device on the local network, and API is used to get information about the device and perform some actions.

2. Discovery

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

http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt
http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt
http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt

Device MUST perform name conflict resolution according to the above specifications.

2.1. Service Type

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

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

In addition to the main service type, device MUST advertise PTR record for a 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 2 PTR records:
```
_privet._tcp.local.
_printer._sub._privet._tcp.local.
```

2.2. TXT record

DNS Service Discovery defines fields to add optional information about a service in the TXT records. TXT record consist of a key/value pairs. Each key/value pair starts from the length byte followed by up to 255 bytes of text. Key is the first part up to the first '=' character. Value is the text to the right of first '=' character till 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 for DNS TXT record format and recommended length).

Privet requires 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 change its value. The same description MUST be used when registering device.

2.2.4. base_url

Server URL this devices is connected to. For example:

```
      base_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 corresponding PTR record. For each supported service subtypes, here should be one corresponding item. Service subtype name (<subtype>._sub._privet._tcp) should be equal to device type here.

2.2.6. id

Cloud ID of the device. If 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. ds

Indicates current device state. Device may be in one of the 3 possibles states. “**idle**” means device is ready to handle to work. “**processing**” means device is busy processing something, so some functionality may be limited at the moment (e.g. printer may be printing right now). “**stopped**” means device is not working and require a user intervention in order to continue. For example:

```
        ds=idle
        ds=processing
        ds=stopped
```

2.2.8. cs

Indicates current connection state. Two possible values are defined in this spec. “**online**” indicates that device is currently connected to the cloud. “**offline**” indicates that device is available on the local network, but can’t talk to the server. For example:

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

If 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). 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, device MUST perform announcement step as described in the mDNS specification. It SHOULD send corresponding announcement at least twice with at least a 1 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 grouping all records into one DNS response if possible. If not, following order is recommended: SRV, PTR, TXT records.

3.2. Shutdown

On device shutdown it MUST 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, device MUST send an update announcement. It is enough only send 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 `/info` request (note that header MUST still be present). If "X-Privet-Token" header is missing, device MUST respond with 400 HTTP error as following:

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

If "X-Privet-Token" header is empty or invalid, device MUST respond with "*invalid X-Privet-Token error*" (*invalid_x_privet_token,* see Errors section for details). The only exception is /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 requested API does not exist or not supported, device MUST return HTTP 404 error.

4.1. /info API

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

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

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.

Info API MUST return data consistent with data available in TXT record during discovery.

A device that is already registered with a cloud service MUST call the server prior to exposing `/info` API in order to confirm what functionality can be exposed over the local network. Device can cache functionality for subsequent /info calls during runtime or over reboots (if device boots up offline).
<A specific Google Cloud Print API to query for the set of the supported functionality will be released shortly. Google Cloud Print may add a notification to notify the device of changes.>

4.1.1. Input

`/info` API has no input parameters.

4.1.2. Return

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

TXT column indicates corresponding field in 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
base_url	string	URL of the server this device is talking to. Url MUST include protocol specification, for example: https://www.google.com/cloudprint.	base_url
type	list of strings	List of device types supported.	type
id	string	Device cloud 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	ds
connection_state	string	State of the connection to the server (base_url) "**online**" - connection available "**offline**" - no connection	cs
access_scopes	list of strings	List of allowed scopes to be used in /accesstoken API (send from server). Format of the scope is still TBD.	
manufacturer	string	Name of the device manufacturer	
model	string	Model of the device	
firmware	string	(optional) Device firmware version	
serial_number	string	(optional) Device serial number.	
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)	

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

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

Once device registration is complete, 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": [
      "/accesstoken",
      "/capabilities",
      "/printer/submitdoc",
]
```

Following APIs are available at this time:

> **`/register`** - API for device registration over the local network. (see `/register` API for details). This API MUST be hidden once the device is successfully registered in the cloud.
>
> **`/accesstoken`** - API to request access token from the device (see `/accesstoken` API for details).
>
> **`/capabilities`** - API to retrieve device capabilities (see `/capabilities` API for details).
>
> **`/printer/*`** - API specific to the device type "printer", see printer specific APIs for details.

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

```
{
      "version": "1.0",
      "name": "Gene's printer",
      "description": "Printer connected through Chrome connector",
      "base_url": "https://www.google.com/cloudprint",
      "type": [
            "printer"
      ],
      "id": "11111111-2222-3333-4444-555555555555",
      "device_state": "idle",
      "connection_state": "online",
      "access_scopes": [
            "sharing:confirmation",
            "access:auto"
      ]
      "manufacturer": "Google",
      "model": "Google Chrome",
      "firmware": "24.0.1312.52",
      “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”: [
            “/accesstoken”,
            “/capabilities”,
            “/printer/submitdoc”,
      ]
}
```

4.1.3. Errors

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

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

4.2. /register API

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

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

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

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

Registration process consists of 2 steps (See anonymous registration for Cloud Print).

Initiate anonymous registration process
Client initiates this process by calling the `/register` API. Device sends 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 drop all previous data (if any) and start a new registration process.

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

4.2.1. Input

`/register` API has following input parameters:

Name	Value
`action`	Can be one of the following: “**start**” - to start registration process “**complete**” - to complete registration process
`user`	(optional) Email of the user who will claim this device.

Device should pass user email (if present) as is to the server register API.

It is highly recommended for client to specify a user email in this API call. Server will verify the provided user email matches the account used to claim the printer.

4.2.2. Return

`/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 “start” response, omitted for “complete”).
`claim_url`	string (optional)	Registration url (mandatory for “start” response, omitted for “complete”).
`device_id`	string (optional)	New device id (omitted for “start” response, mandatory for “complete”).

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

Example 1:

```
{
	“action”: “start”,
	“user”: “user@domain.com”,
	“token”: “AAA111222333444555666777”,
	“claim_url”: “https://domain.com/SoMeUrL”,
}
```

Example 2:

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

4.2.3. Errors

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

Error	Description
`offline`	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.

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

4.3. /accesstoken API

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

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

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

4.3.1. Input

`/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.
`scope`	Scope of how this token is intended to be used. This is an arbitrary string, that should be passed to the server API as is. This may be empty.

4.3.2. Return

`/accesstoken` API returns following data:

Value name	Value type	Description
`token`	string	Access token returned by the server
`token_info`	JSON Object	Device MUST copy this parameter from the server response as is. This allows for flexibility in adding additional or different parameters in the future. For example, it might contain

		number of usages this token is valid for.
`user`	string	Same user as in input parameter.
`scope`	string	Scope returned by the server (may be empty).
`expiration`	string (optional)	Timestamp of the token expiration. Omitted if there is no known expiration of the token. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970).

Example:

```
{
       “token”: “AAA111222333444555666777”,
       “token_info”: {
              “some_info”: “abcdefg”
       },
       “user”: “user@domain.com”,
       “scope”: “full-access”,
       “expiration”: “1359143880”,
}
```

4.3.3. Errors

`/accesstoken` API may return 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. Device should return this error when request has been explicitly denied by server.
`server_error`	Server error.
`invalid_x_privet_token`	X-Privet-Token is Invalid or empty in the request.

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

4.4. /capabilities API

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

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

When device receives /`capabilities` API call, if the device is able, it SHOULD contact server to get updated capabilities. For example, if printer supports posting 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 printer is offline, it should return document types it supports. However, if printer is online and registered with Cloud Print it MUST return “*/*” as one of the supported types. Cloud Print service will perform necessary conversion in this case. For offline printing, printer MUST support at least “image/pwg-raster” format.

4.4.1. Input

`/capabilities` API has no input parameters.

4.4.2. Return

`/capabilities` API returns device capabilities in the Cloud Device Description (CDD) JSON format (see 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”: {
			“option”: [
				{
					“content_type”: “application/pdf”,
					“rank”: 1,
					“min_version”: “1.4”
				},
				{
					“content_type”: “image/pwg-raster”,
					“rank”: 2
				},
				{
					“content_type”: “image/jpeg”,
					“rank”: 3
				},
				{
					“content_type”: “*/*”,
					“rank”: 4
				}
			]
		}
	}
}
```

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

```
{
	“version”: “1.0”,
	“printer”: {
		“supported_content_type”: {
			“option”: [
				{
					“content_type”: “application/pdf”,
					“rank”: 1,
                                “min_version”: “1.4”
                          },
                          {
                                “content_type”: “image/pwg-raster”,
                                “rank”: 2
                          },
                          {
                                “content_type”: “image/jpeg”,
                                “rank”: 3
                          }
                    ]
             }
      }
}
```

4.4.3. Errors

`/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 `/capabilities` API, it MUST return HTTP 404 error. If X-Privet-Token header is missing, device should return 400 HTTP error.

4.5. Errors

Error is returned from any of 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 server API that failed.
`server_code`	int (optional)	In case of server error, this field contains error code 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 error 400 if X-Privet-Token header is missing.

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

Example 1:

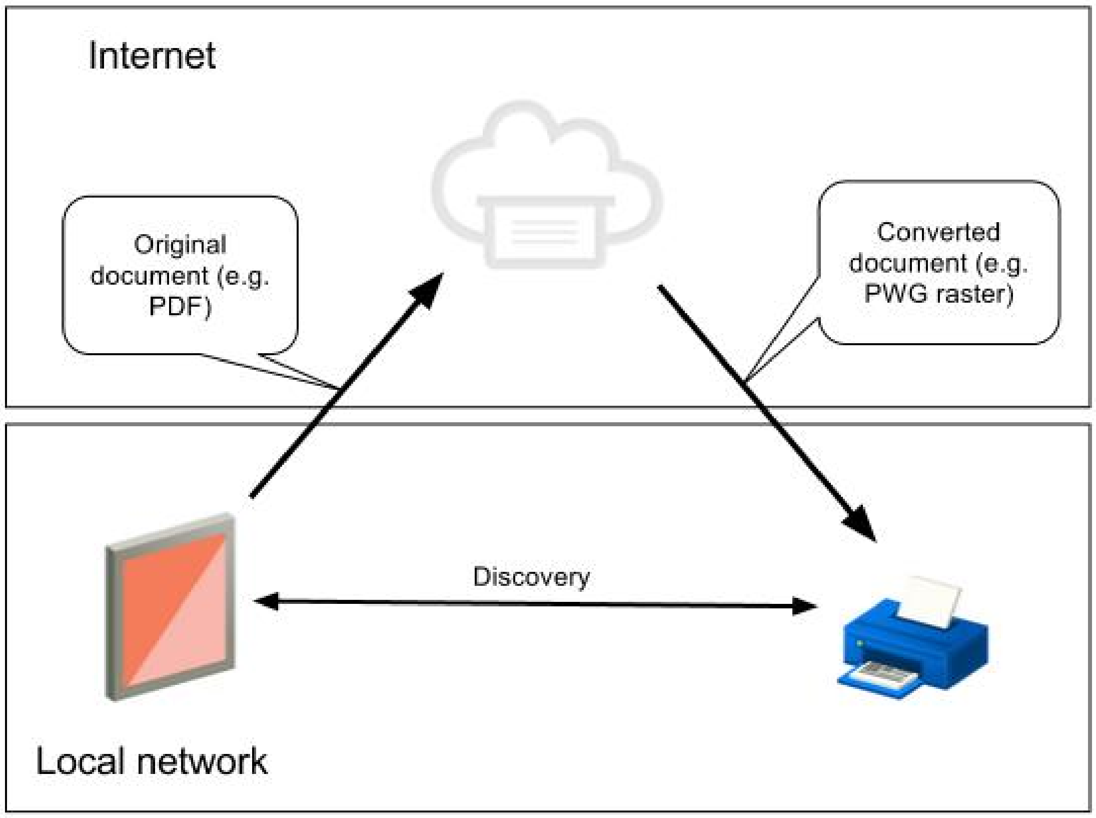
```
{
        “error”: “server_error”,
        “description”: “Service unavailable”,
        “server_api”: “/submit”,
        “server_http_code”: 503
}
```

Example 2:

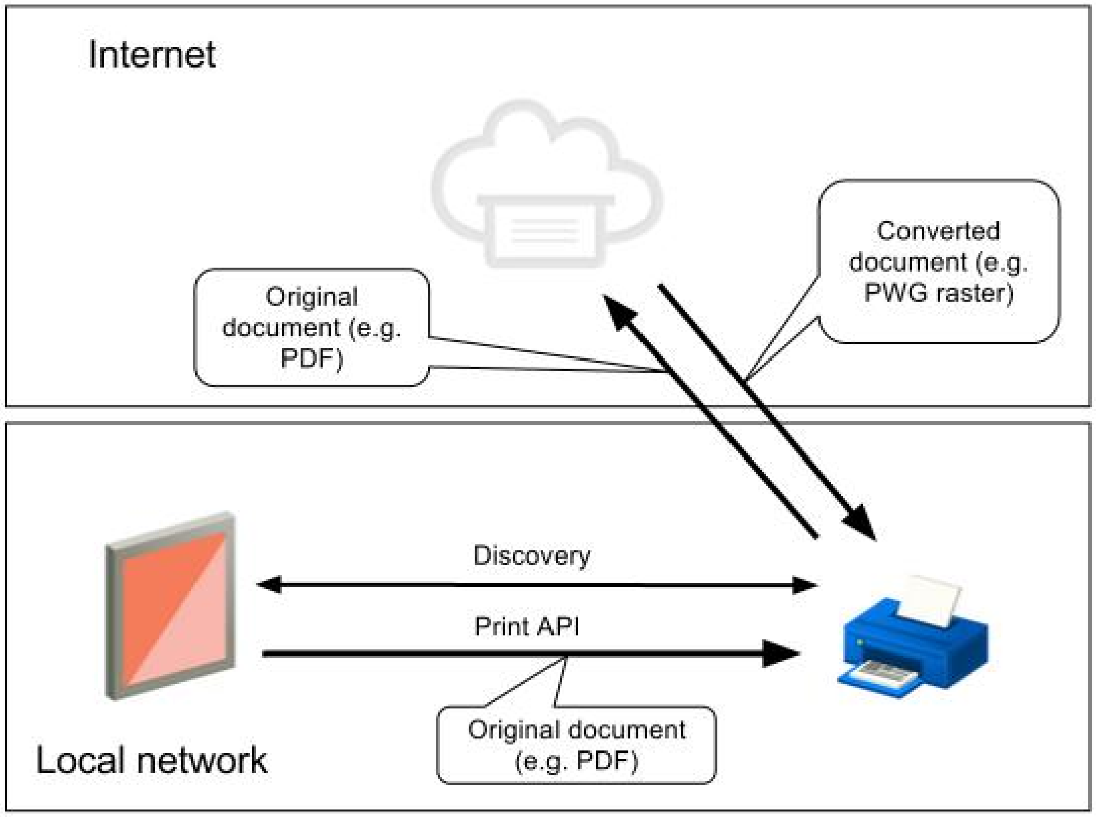
```
{
        “error”: “printer_busy”,
        “description”: “Printer is currently printing other job”,
        “timeout”: 15
}
```

5. Printer API

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


Internet
Original document (e.g. PDF)
Converted document (e.g. PWG raster)
Discovery
Local network

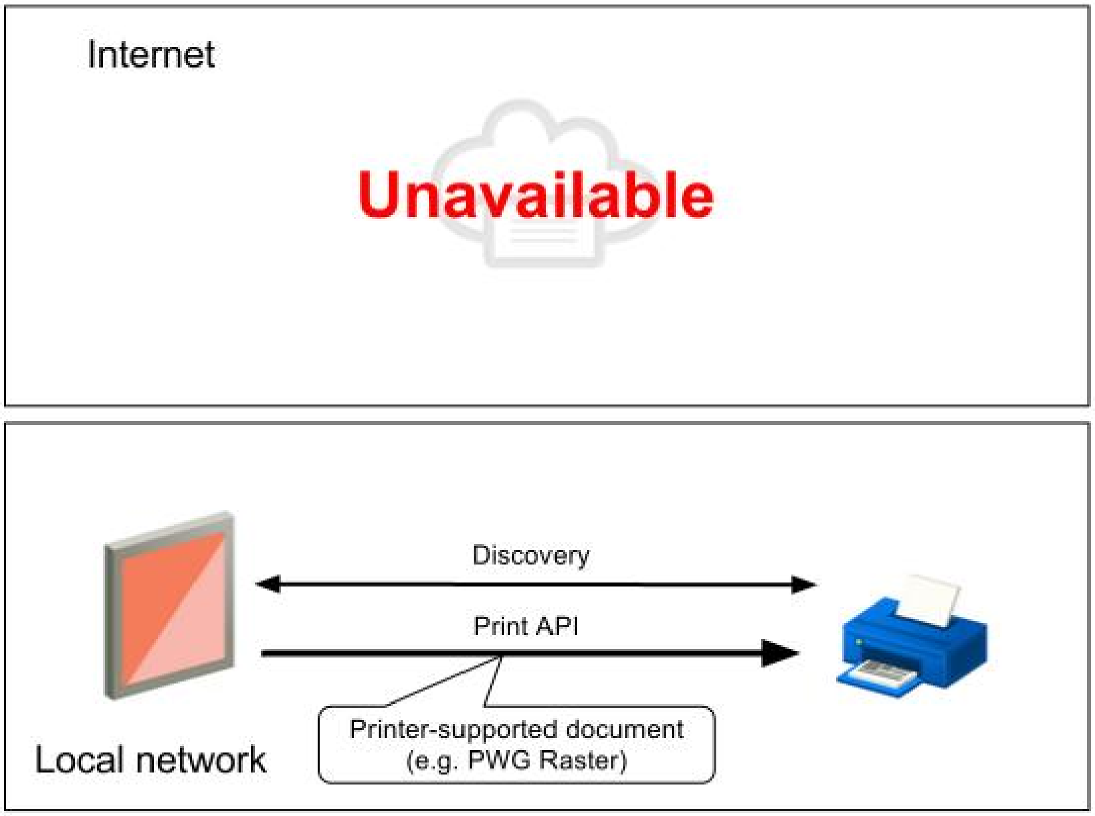
However, in some cases client may need to send a document locally. It may be needed when client does not have Google ID or can't talk to the Cloud Print server for some reasons. In such case, the print job will be submitted locally to the printer. The printer, in turn, will use the Cloud Print service 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.


Internet
Original document (e.g. PDF)
Converted document (e.g. PWG raster)
Discovery
Print API
Original document (e.g. PDF)
Local network

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

```
{
      "version": "1.0",
      "printer": {
            "supported_content_type": {
                  "option": [
                        {
                              "content_type": "image/pwg-raster",
                              "rank": 1
                        },
                        {
                              "content_type": "*/*",
                              "rank": 2
                        }
                  ]
            }
      }
}
```

In some cases a complete offline solution is desired. Since printer capabilities understanding different formats are limited, client will have to convert document to a few natively supported printer format.



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

```
{
        “version”: “1.0”,
        “printer”: {
                “supported_content_type”: {
                        “option”: [
                                {
                                        “content_type”: “image/pwg-raster”,
                                        “rank”: 1
                                },
                                {
                                        “content_type”: “image/jpeg”,
                                        “rank”: 2
                                }
                        ]
                }
        }
}
```

There are 2 ways client may initiate printing over the local network.

Simple printing - client sends the document over the local network to `/submitdoc` API (without specifying `job_id` parameter). 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 `/info` API response..

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

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

In the multiclient 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 printjobs on the printer (at least 3-5):

- `/createjob` takes the first available stop in the queue.
- Job lifetime (in the queue) is at least 5 minutes.
- If all spots in the queue is taken, 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 the job that has been removed from the queue (due to replacement or timeout), printer should return an error `unknown_job_id` and client will retry process from the `/createjob` step. 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 job is complete/error, printer should store information about jobs status for at least 5 minutes. 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 printer to send TXT DNS notification when ANY print job status has changed.>

5.1. /printer/createjob API

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

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

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

5.1.1. Input

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

5.1.2. Return

`/printer/createjob` API returns following data:

Value name	Value type	Description
`job_id`	string	ID of the newly created print job.
`job_timestamp`	string	Time stamp when print job has been created. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970.)
`job_expiration`	string	Time stamp when print job expires. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970.)

Client SHOULD not match `job_timestamp` or `job_expiration` against client time, since time of the client and device might be different. Instead client should make its own timestamp when receiving this response and use `job_expiration - job_timestamp` time interval to see if job is still valid.

Example:

```
{
      “job_id”: “123”,
      “job_timestamp”: “1359143880”,
      “job_expiration”: “1359144480”
}
```

5.1.3. Errors

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

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

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

5.2. /printer/submitdoc API

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

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

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

5.2.1. Input

`/printer/submitdoc` API has 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. Can't be trusted and used just to annotate print job. 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), client name and job name with this request. Those fields are only used in UI to improve user experience.

5.2.2. Return

`/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).
`job_timestamp`	string	Time stamp when print job has been created. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970.)
`job_expiration`	string	Time stamp when print job expires. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970.)
`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",
      "job_timestamp": "1359143880",
      "job_expiration": "1359144480",
      "job_type": "application/pdf",
      "job_size": 123456,
      "job_name": "My PDF document"
}
```

5.2.3. Errors

`/printer/submitdoc` API may return 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 is too large.
`printer_busy`	Printer is busy and can't currently process document. Retry after timeout.
`printer_error`	Printer is in error state and require user interaction to fix it. Description should contain more detailed explanation (e.g. "Paper jam in Tray 1").
`server_error`	Posting document to Cloud Print has failed.
`invalid_x_privet_token`	X-Privet-Token is invalid or empty in the request.

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

5.3. /printer/jobstate API

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

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

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

5.3.1. Input

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

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

5.3.2. Return

`/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`**" - when print job has been created on the device (no `/printer/submitdoc` calls has been received yet). "**`queued`**" - when print job has been received and queued, bu printing has not started yet. "**`in_progress`**" - when print job is in the progress of printing. "**`stopped`**" - print job has been paused, but can be restarted manually or automatically. "**`done`**" - when print job is done. "**`aborted`**" - when print job failed.
`description`	string	(optional) Human readable description of the print job status. Should include additional information if `status` is "`paused`" or "`error`".
`job_timestamp`	string	Time stamp when print job has been created. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970).
`job_expiration`	string	Time stamp when print job expires. This should be in the format of the UNIX 64-bit timestamp. (Number of seconds since epoch, Jan 1st, 1970).
`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)	(optional) ID of the job returned from the server (if job has been posted to Cloud Print service). Omitted for offline printing.

Example (printing by reporting through Cloud Print):

```
{
      “job_id”: “123”,
      “status”: “in_progress”,
      “job_timestamp”: “1359143880”,
      “job_expiration”: “1359144480”,
      “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”,
      “status”: “paused”,
      “description”: “Out of paper”,
      “job_timestamp”: “1359143880”,
      “job_expiration”: “1359144480”,
      “job_type”: “application/pdf”,
      “job_size”: 123456,
      “job_name”: “My PDF document”
}
```

5.3.3. Errors

`/printer/jobstate` API may return 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 `/printer/jobstate`, it MUST return HTTP 404 error. If X-Privet-Token header is missing, device should return 400 HTTP error.

6. Appendix

6.1. XSSI and XSRF attacks and prevention

This section will explain possibility of the 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 site is using cookie authentication mechanism. While we don't use cookies here, we are still vulnerable to such attacks. By using the local network we are trusting requests implicitly.

6.1.1. 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/info"></script>
```

Without any 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.

6.1.2. 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 still trigger it.

To prevent this attack, we require the following protection:

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

Even if an attacker is able to execute the `/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 following algorithm:

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

where:
`DELIMITER` is a special character, usually '**:**'
`issue_timestamp` is a Unix 64 bit timestamp (number of seconds from the 1st of Jan 1970)
`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 new UUID on every restart

- Generate 64 bit random number on every restart

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_timestamp` from the second half (cleartext), and try to produce SHA1 hash of `device_secret + DELIMITER + issue_timestamp` where `issue_timestamp` is from the token. If newly generated SHA1 matches the one in the token, device must now check if the `issue_timestamp` is within the validity period from (24 hours) of the current time.

6.2. Secure printing over local network (TBD)

While the local network is assumed more secure than public networks, additional measures can be taken when sending sensitive documents to a printer. To provide encryption capabilities of a print job, use the following procedures:

- Printer must expose it's public key and algorithm in the /info API.
- Client will use printer's public key to encrypt content and send it over /submitdoc API (adding an extra parameter or header that content is encrypted).

To mitigate the risk of keys being compromised, a printer should regenerate it's keys on every startup and/or on a daily basis, or may ask server (GCP) to perform key generation for it.