Polar Cloud Printing Protocol

19 May 2017

27 December 2017 (revised)

Copyright © 2017, Polar 3D, LLC, All Rights Reserved

Contact: Dan Newman <dnewman@polar3d.com>

0.0 Introduction

The Polar Cloud is comprised of two services:

1. A cluster of Web Servers which handle the web interface used by human users (members) of the Polar Cloud. These Web Servers are behind load balancers and reached via the URL https://polar3d.com.
2. A cluster of Status Servers which receive updates from printers and send commands to the printers from the Web Servers. These Status Servers are behind load balancers and reached via an http:// or https:// URL.

The WebSocket protocol is used by printers to communicate with the Status Servers. This document concerns itself with commands and responses spoken between printers and the Status Servers over the WebSocket protocol.

Presently, all of these systems are written in Node.js as is the Polar Software itself running on Polar3D printers. For Node.js, the WebSocket library used is socket.io 1.4.6. OctoPrint's open source implementation of this protocol in Python may be found at OctoPrint-PolarCloud. Some implementations in C/C++ have used the Qt WebSockets implementation available on GitHub at qt/qtwebsockets.

0.1 Command synopsis

Briefly, the commands sent from the Status Server to the Printer are

Required	Command	Section	Brief
✔	cancel	2.1	Cancel print
	capabilitiesResponse	2.2	capabilities command response
	command	2.3	Execute gcode command
	connect	2.4	Connect controller to printer
	customCommand	2.5	Execute a custom command
✔	delete	2.6	Reset printer to unregistered state
	getQueueResponse	2.7	getQueue command response
	getUrlResponse	2.8	getUrl command response
✔	helloResponse	2.9	hello command response
	keyPair	2.10	makeKeyPair response
✔	pause	2.11	Pause current print
✔	print	2.12	Start a print
✔	registerResponse	2.13	register response
✔	resume	2.14	Resume a print
	sendNextPrintResponse	2.15	sendNextPrint response
	temperature	2.16	Set target temperature
	unregisterResponse	2.17	unregister response
	update	2.18	Begin printer software update
✔	welcome	2.19	Challenge from the Status Server

And the comands sent from the Printer to the Status Server are

Required	Command	Section	Brief
	capabilities	3.1	List cloud-enabled capabilities for the printer
	commandResponse	3.2	Send response to prior command command
	customCommandList	3.3	Send to cloud printer's custom commands
	getQueue	3.4	List printer's queued jobs
	getUrl	3.5	Obtain signed POST URL for uploading images
✔	hello	3.6	Respond to welcome challenge
✔	job	3.7	Report job completion
	makeKeyPair	3.8	Request from the cloud a crypto key pair
✔	register	3.9	Register printer; receive cloud serial number
	sendNextPrint	3.10	Request a print job
	setVersion	3.11	Report software version
✔	status	3.12	Report printer status
	unregister	3.13	Unregister printer

0.2 Client reference implementation

The open source OctoPrint implementation of this protocol serves as a reference implementation and may be found at https://github.com/markwal/OctoPrint-PolarCloud. This implementation is written and maintained by Mark Walker, one of the main developers of OctoPrint. The code, like much of OctoPrint, is written in Python.

Implementations in C and C++ have successfully used the QtWebSockets library to implement the WebSockets protocol.

1.0 Command Flow

For the most part, commands do **not** have responses (acknowledgements) and the Status Servers are passive, merely receiving commands from the printers. The commands from the Status Server are user initiated; e.g., when a user requests that a print be started, the Web Server sends a `print` command to the Status Server which then relays it to the actual printer.

Once connected to the Status Servers, the printer remains connected until powered off. Should the network connection be disrupted, the printer should attempt to re-establish the connection. Many WebSocket libraries will have automatic reconnection strategies (e.g., socket.io for Node.js).

1.1 Registration connection

When connecting to the Polar Cloud for the first time, a printer must register itself and receive a serial number. Part of the registration involves sending a public encryption key to the Status Server which will then save it to later validate the identity of the printer by asking the printer to sign a challenge with its private key.

The typical command flow when registering is illustrated below:

Printer: The printer establishes a connection to the Status Server.
Server `welcome`: The Status server sends to the printer a `welcome` command which, in this specific case, the printer will ignore.
Printer `register`: The printer sends a `register` command to the Status Server. The `register` command contains the printer owner's Polar Cloud account's e-mail address and PIN number, the printer's ethernet MAC address, and the public RSA key of a 2048 bit RSA key pair generated by the printer.
Server `registerResponse`: The server responds with a `registerResponse` command.
Printer: The printer then disconnects and then reconnects using its newly registered identity. (See the next section.)

Note that if the printer simply does not have enough processing power to generate an RSA key pair, then the printer may request a key pair from the Status Server. In that case, the registration command flow becomes:

Printer: The printer establishes a connection to the Status Server.
Server `welcome`: The Status server sends to the printer a `welcome` command which, in this specific case, the printer will ignore.
Printer `makeKeyPair`: The printer sends a `makeKeyPair` request to the Status Server and then awaits a `keyPair` response back. **Server** `keyPair`: The server generates a RSA key pair and sends the key pair to the Printer. The printer must save the private RSA key in non-volatile memory for future use. **Printer** `register`: The printer sends a `register` command to the Status Server. The `register` command contains the printer owner's Polar Cloud account's e-mail address and PIN number, the printer's ethernet MAC address, and the public RSA key from the key pair sent by the Status Server to the printer.
Server `registerResponse`: The server responds with a `registerResponse` command.
Printer: The printer then disconnects and then reconnects using its newly registered identity. (See the next section.)

1.2 Registered connections

Upon connecting to the Polar Cloud, a printer will receive a `welcome` command containing a challenge it must sign with its private key. The digitally signed challenge is then sent back to the server using the `hello` command. The server will ignore the printer until such time that it receives the signed challenge. Upon receipt of the signed challenge, it is validated against the printer's previously provided public key. If the signature is valid, the printer may begin interacting with the server. If it is not valid, the connection will be terminated.

The typical command flow when registering is illustrated below:

Printer: The printer establishes a connection to the Status Server.
Server `welcome`: The Status server sends to the printer a `welcome` command which will contain a challenge which the printer must digitally sign.
Printer `hello`: The printer sends a `hello` response back to the Status Server. The response includes the printer's serial number as well as a digital signature of the challenge. The challenge is signed with the printer's private RSA key.
Server: The server validates the received signature using the printer's public RSA key which the server saved when the printer first registered itself with the cloud. If the signature is valid, the printer is then allowed to remain connected, sending and receiving commands. **Server**: Server responds by sending a `helloResponse` commmand to the printer indicating if the `hello` command succeeded or not. (If it does not succeed then the TCP socket will be closed by the server shortly after sending the `helloResponse`.)
Printer: The printer periodically sends status updates to the Status Server using the `status` command.
Printer: The printer may request a signed POST URL with which to periodically POST camera images to the cloud.

Printer: The printer listens for and responds to command from the cloud.

Once the printer's identity has been validated, then typically every 10 seconds the printer sends a `status` to the Status Server. (If nothing has changed, it can omit sending an update.) The printer can also upload a JPEG camera image every minute using a pre-signed POST URL of type `idle` (Section 1.4).

Should the printer begin printing, it should then upload camera images every 10 seconds using a pre-signed POST URL of type `printing`. When it has finished printing and is ready to upload a time-lapse video, it uploads the video using a pre-signed POST URL of type `timelapse`. See Section 1.4 for further details.

1.3 RSA cryptographic keys

> Note: printers which simply cannot generate an RSA key pair owing to processor constraints may request a key pair from the Status Server using the `makeKeyPair` command as discussed in Section 3.8.

Each printer must generate its own, unique 2048 bit RSA cyrptographic key pair and send the public key to the cloud. The command line commands to generate a 2048 bit RSA key pair with `ssh-keygen` is as follows,

```
# Generate a private RSA key in the file key.priv
ssh-keygen -t rsa -b 2048 -f key.priv
# Generate the corresponding public key in the proper format
ssh-keygen -e -m PEM -f key.priv > key.pub
```

To instead use OpenSSL, use the commands

```
# Generate the key pair
openssl genrsa -des3 -out private.pem 2048
Enter pass phrase for private.pem: abcd
Verifying - Enter pass phrase for private.pem: abcd

# Save the RSA public key in the file key.pub
openssl rsa -in private.pem -outform PEM -pubout -out key.pub
Enter pass phrase for private.pem: abcd
writing RSA key

# Save the RSA private key in the file key.priv
openssl rsa -in private.pem -out key.priv -outform PEM
Enter pass phrase for private.pem: abcd
writing RSA key
```

The following sample Node.js program demonstrates:

1. Writing the public key in the same format it must be transmitted to the Status Server.
2. Signing a challenge with the private key from the file `key.priv`.
3. Writing the signature out in the same format it must be sent to the Status Server.

4. Validating the challenge using the public key from the file `key.pub`.

.

```
var fs = require('fs');
var crypto = require('crypto');
var challenge = '1234 ABCD';

// Synchronously load our keys
var key = { };
try {
  key.public = fs.readFileSync('key.pub');
  key.private = fs.readFileSync('key.priv');
}
catch (e) {
  console.log(`Startup: unable to load the RSA key pair; err = ${e.message}`);
  console.log(JSON.stringify(e));
  process.exit(1);
}

// Transmit the public key
var keyMsg = {
  mfg: 'test',
  email: 'dnewman@sample.com',
  pin: '1234',
  publicKey: key.public.toString('utf8'),
  myInfo: { MAC: '12.34.de.ad.be.ef' }
};
console.log('Transmitted public key:')
console.log(JSON.stringify(keyMsg, null, '    '), '\n');

// Sign the challenge
var sign = crypto.createSign('RSA-SHA256');
sign.update(challenge);
var signature = sign.sign(key.private, 'base64');

// Transmit the signature
var data = { serialNumber: "P3D99999", signature: signature};
console.log('Serial number and signature:');
console.log(JSON.stringify(data, null, '    '), '\n');

// Verify the signature
var verify = crypto.createVerify('RSA-SHA256');
verify.update(challenge);
var sig_buffer = new Buffer(signature, 'base64');
var result = verify.verify(key.public, sig_buffer);
console.log('Signature matches:', result);
```

The above node.js program produces output similar to

```
Transmitted public key:
{
    "mfg": "test",
    "email": "dnewman@sample.com",
    "pin": "1234",
    "publicKey": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1nlQRlIlpv9i
    "myInfo": {
        "MAC": "00.34.DE.AD.BE.EF"
    }
}

Serial number and signature:
{
    "serialNumber": "P3D99999",
    "signature": "SiW1A9gIIxzcOaj16chBrIZM7C9UtdbuWq9i580eHl6U+Ki5FM6HJ0DyTsN8CKA6yShlIMrcEOg7MXEV//lc
}

Signature matches: true
```

1.4 Time-Lapse videos

The time-lapse videos uploaded should be MP4 encoded using the H.264/MPEG-4 AVC format. The specific command used on Linux to perform the encoding is

```
gst-launch-1.0 qtmux name=mux ! filesink location=\"$ARG2\"  \
  multifilesrc location=\"$ARG1\" index=1 \
  caps=\"image/jpeg,framerate=\(fraction\)12/1\" ! jpegdec ! videoconvert ! \
  videorate ! x264enc ! mux .
```

The duration of the videos should be kept to 30 seconds of play time. Time-Lapse videos should not be produced for prints which are canceled.

1.5 Uploading camera images and videos

To upload to the cloud a camera image or time-lapse video, the printer must request from the Status Server a pre-signed POST URL using the `getUrl` command. The POST URL is then used with an HTTP or HTTPS POST request to send the camera image to the Polar Cloud. The same URL may be used many times -- typically for up to 24 hours. When the URL expires, a new URL can be requested from the Status Server. (A new URL can be requested before the old one expires.)

> These POST URLs are cryptographically signed by the Status Server. Amazon's S3 service will validate the signature of the URL before actually storing the transmitted data. Amazon S3 refers to these URLs as "pre-signed URLs".

There are two different camera image types and one video type which can be uploaded to the Polar Cloud. Each type requires its own signed POST URL.

- `idle`: JPEG camera image while idle and not printing.
- `printing`: JPEG camera image while printing a job from the cloud. Not for local prints!
- `timelapse`: MP4 time-lapse video from printing a job from the cloud. Again not for local prints.

No other types of images should be sent. If a printer is printing a local job -- a job not supplied by the cloud -- only send up `idle` images for it if you wish cloud members to be able to view the printer as it prints the local job. (It's okay to send such images more frequently than once a minute.)

> **NOTE:** The printer must flip any rotated or upside down camera images and time-lapse videos before uploading to the cloud. The `rotateImg` and `transformImg` fields of the `hello` command are merely to inform web browsers whether or not they must transform the **live camera feed** or not.

Each pre-signed POST URL is requested by sending a `getUrl` command to the Status Server with the `type` property set to `idle`, `printing`, or `timelapse`. As might be expected, `idle` is for when the printer is idle and not printing, `printing` for when it is printing a job from the cloud, and finally `timelapse` for a time-lapse video of a cloud print.

The Status Server returns a JSON-formatted object as described in Section 2.8. The object will indicate when the URL will expire as "seconds from the present", the maximum file size which may be uploaded expressed in bytes, the HTTP/HTTPS URL to POST to, and all the POST form data fields and their values which **must** be supplied. Additionally, the form data must include a `file` field which must be supplied along with the appropriate MIME content-type header line and the file data itself.

For example, if `getUrl` returns

```
{
    "status": "SUCCESS",
    "serialNumber": "P3D99979",
    "method": "post",
    "type": "idle",
    "maxSize": 76800,
    "expires": 86400,
    "contentType": "image/jpeg",
    "url": "https://s3.amazonaws.com/polar3d.com",
    "fields": {
        "key": "files/printer/P3D99979/snapshot.jpg",
        "acl": "public-read",
        "bucket": "polar3d.com",
        "X-Amz-Algorithm": "AWS4-HMAC-SHA256",
        "X-Amz-Credential": "AKIAJKCJDY6DKKJSKFD7A/20170521/us-east-1/s3/aws4_request",
        "X-Amz-Date": "20170521T234827Z",
        "Policy": "eyJleHBpcmF0aW9uIjoiMj=",
        "X-Amz-Signature": "972b7428be734f2ad3f8b3f6c895087436ae1651a"
    }
}
```

Then the corresponding POST request executed with the curl utility would take the form

```
curl -X POST \
     -F 'key=files/printer/P3D99979/snapshot.jpg' \
     -F 'bucket=polar3d.com' \
     -F 'acl=public-read' \
     -F 'X-Amz-Algorithm=AWS4-HMAC-SHA256' \
     -F 'X-Amz-Credential=AKIAJKCJDY6DKKJSKFD7A/20170521/us-east-1/s3/aws4_request' \
     -F 'X-Amz-Date=20170521T234827Z' \
     -F 'Policy=eyJleHBpcmF0aW9uIjoiMj=' \
     -F 'X-Amz-Signature=972b7428be734f2ad3f8b3f6c895087436ae1651a' \
     -F 'file=@/Users/bob/Desktop/camera-image.jpg;type=image/jpeg' \
     'https://s3.amazonaws.com/polar3d.com'
```

In that curl command, the image file to be uploaded is specified with the line

```
-F 'file=@/Users/bob/Desktop/camera-image.jpg;type=image/jpeg' \
```

That tells curl to upload the contents of the file

```
/Users/bob/Desktop/camera-image.jpg
```

and to use a MIME Content type/subtype value of `image/jpeg`.

2.0 Commands from the Cloud to Printers

The following commands set by the Status Servers to printers.

2.1 cancel

Cancel an ongoing print. The print may either be a job from the Polar Cloud or a local print job. In the case of a print job from the Polar Cloud, the job will be left in the printer's queue as typically this command is used when a problem is detected at the start of the print for which the user intends to correct the issue and then start the print over again (e.g., poor bed adhesion, filament was not loaded, etc.).

Command: `cancel`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number"    // string, required
}
```

2.2 capabilitiesResponse

This command is sent by the cloud to a printer in response to a capabilities request made by the printer. It contains a list of zero or more strings, each string a special capability supported by the cloud for the printer.

Presently, the only capability is the `sendNextPrint` capability. A printer with that capability may use the `sendNextPrint` command to request that the next queued print job be sent to it.

> Presently, all printers may use the `sendNextPrint` command.

Command: `capabilitiesResponse` Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",     // string, required
    "capabilities": [ "cap-1", "cap-2", ... ]    // array of zero or more strings, required
}
```

2.3 command

The `command` command allows any arbitrary gcode command (or string of gcode commands) to be sent to a printer. It is possible that more than one command may be sent at once by using a US-ASCII line feed character, 0x0A, as a delimiter between each command.

Command: `command`

Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",                     // string, required
    "command": "string containing the gcode command(s) to send"  // string, required
}
```

Example argument:

```
{
    "serialNumber": "P3D0000",
    "command": "M115\n"
}
```

In the above, the `\n` represents a US-ASCII line feed character with ordinal hexadecimal value 0x0A.

The printer may send the responses elicited by the commands back to the cloud with a `commandResponse` command.

> ***Note Bene:*** Any M104 or M140 temperature commands placed in the `command` field will be extracted and placed into a single, separate `temperature` command. The resulting `temperature` command will be sent before the other commands. And no `commandResponse` will necessarily be generated for those temperature settings. It is best, therefore, to never include M104 or M140 gcode commands in the command string sent with this command.

2.4 connect

This command tells the listener to connect to the printer if it is in a disconnected state. The format of the command is

Command: `connectPrinter`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number"    // string, required
}
```

2.5 customCommand

If the printer has supplied the cloud with a list of one or more custom commands, then the printer owner may from the cloud execute one of those commands. This is communicated to the printer by the cloud via the `customCommand` command.

Command: `customCommand` Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",    // string
    "customCommand": "command"                  // string
}
```

The value `command` will be one of the commands previously supplied with a `customCommandList` command from the printer. For example, if the printer previously sent to the cloud

```
{
    "serialNumber": "OP000001",
    "customCommandList": [
        {
            "label": "printer off",
            "command": "COMMAND-1",
            "helpText": "Turn your printer's power off",
            "confirmText": "You are about to turn off your printer."
        },
        {
            "label": "printer on",
            "command": "COMMAND-2",
            "helpText": "Turn your printer's power on"
        }
    ]
}
```

then the cloud might send to the printer at some point the `customCommand`

{ "serialNumber": "OP000001", "command": "COMMAND-1" }

which would signify a request to turn of the user's printer.

2.6 delete

When the owner of a printer deletes the printer in the Polar Cloud, a `delete` command is sent to the printer if it is currently connected to the Status Server. Upon receipt of a `delete` command, a printer should consider itself as no longer registered in the Polar Cloud.

The command takes the form,

Command: `delete`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number"    // string
}
```

It is up to the printer to determine what it should do if it receives this command while actively printing.

Note that once deleted, the printer will receive "DELETED" responses from the Status Server until it again successfully registers itself with a `register` command. These "DELETED" responses will be sent by the Status Server in a `helloResponse` command after the printer sends a `hello` command.

2.7 getQueueResponse

With the `getQueue` command, a printer may request from the Status Server information on all jobs presently queued to the printer and in a ready to print state. The responses includes pagination information even if the pagination parameters `skip` and `limit` were not used in the `getQueue` request.

The information is returned in the form of a JSON array containing zero or more objects. Each object describes a queued job. The order of the array elements is the order of the respective jobs in the queue. The first array element is the first job in the queue, the second array element the second in the queue, and so on.

Command: `getQueueResponse` Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",     // string
    "totalCount": total number of queued jobs,   // integer
    "skip": skip value,                          // integer
    "limit": limit value,                        // integer
    "jobCount": count of jobs returned,          // integer
    "jobs": [ job-object ]                       // array of job-object
}
```

where each `job-object` is of the form

```
{
    "jobId": "print-job-id",                                         // string
    "jobName": "job-name",                                           // string
    "imageFile": "full URL for a rendering of the job",              // string
    "imageThumbnailFile": "full URL for a small rendering of the job",  // string
    "owner": "Display name of the job's owner",                      // string
    "ownerPhotoFile": "full URL to the owner's profile image"        // string
}
```

These fields are each described in the table below.

Field	Description
`serialNumber`	The printer's serial number
`totalCount`	The total count of jobs queued to the printer and in a ready state
`skip`	The `skip` value specified in the `getQueue` request and 0 if not specified
`limit`	The `limit` value specified in the `getQueue` request and 0 if not specified
`jobCount`	The count of jobs returned in this response
`jobs`	An array of zero or more `job-object` objects
`jobId`	The print jobId for the job
`imageFile`	A full URL to a rendering of the print job; this is typically a PNG file; if the print job was created from a gcode file uploaded by a cloud member, then this parameter may supply the URL of a generic image
`imageThumbnailFile`	A full URL to a small rendering of the print job; this is typically a PNG file; if the print job was created from a gcode file uploaded by a cloud member, then this parameter may supply the URL of a generic image
`owner`	The display name of the job's owner
`ownerPhotoFile`	The full URL to the job owner's profile image

An example response is shown below. This response indicates that two jobs are queued and ready to print.

```
{
  "serialNumber": "DREM000001",
  "totalCount": 2,
  "skip": 0,
  "limit": 0,
  "jobCount": 2,
  "jobs": [
    {
      "jobId": "DREM000001-5141",
      "jobName": "Box.stl",
      "imageFile": "https://s3.amazonaws.com/polar3d.com/printer/DREM000001/5141/yG713NmM-object.png",
      "imageThumbnailFile": "https://s3.amazonaws.com/polar3d.com/printer/DREM000001/5141/yG713NmM-obj
      "owner": "Daniel Newman",
      "ownerPhotoFile": "https://lh5.googleusercontent.com/BdO8lU8zk/MKdiSlH/photo.jpg?sz=50",
    },
    {
      "jobId": "DREM000001-5142",
      "jobName": "Fulcrum test",
      "imageFile": "https://s3.amazonaws.com/polar3d.com/printer/DREM000001/5142/0m8q4eWB-object.png",
      "imageThumbnailFile": "https://s3.amazonaws.com/polar3d.com/printer/DREM000001/5142/0m8q4eWB-obj
      "owner": "3D Fred",
      "ownerPhotoFile": "https://lh3.googleusercontent.com/qdMkCWA/4252rsc/photo.jpg?sz=50",
    }
  ]
}
```

2.8 getUrlResponse

In response to a `getUrl` command from a printer, the Status Server responds with a `getUrlResponse` command containing the requested timelimited, pre-signed POST URL which may be used to upload the requested content. Typical URL lifetimes are 86400 seconds (1 day) for static camera images and 3600 seconds (1 hour) for time-lapse videos.

The `field` sub-object contains the POST form data which must be included in the post along with the file data in a POST data field named `file`.

Command: `getUrlResponse` Argument: a JSON object of the form

```
{
    "status": "SUCCESS" | "FAILED",              // string
    "serialNumber": "printer-serial-number",     // string
    "type": "idle" | "printing" | "timelapse",   // string
    "expires": seconds-until-url-expires,        // integer
    "maxSize": max-file-size-in-bytes,           // integer
    "contentType": "image/jpeg" | "video/mp4",   // string
    "method": "post",                            // string
    "url": "string",                             // string
    "fields": {                                  // POST form data
        "bucket": "S3-bucket-name",              // string
        "key": "bucket-key-name",                // string
        "acl": "public-read",                    // string
        "X-Amz-Algorithm": "AWS4-HMAC-SHA256",   // string
        "X-Amz-Credential": "AWS-credential",    // string
        "X-Amz-Date": "time-stamp",              // string
        "Policy": "encrypted-policy",            // string
        "X-Amz-Signature": "policy-signature"    // string
    }
}
```

In the event of a failure (e.g., invalid `jobId` supplied), the response is instead

```
{
    "status": "FAILED",     // string
    "message": "reason"     // string
}
```

The `message` field may or may not be present.

See Section 1.5 for further details on using pre-signed URLs.

2.9 helloResponse

After receiving a `hello` command from a printer, the Status Server will send back a `helloResponse` command indicating whether the printer supplied signature was valid or not. If it is valid, a "SUCCESS" is indicated in the `helloResponse`. Otherwise, a failure is indicated and the TCP socket closed by the Status Server

command: `helloResponse`
Argument: a JSON object of the form

```
{
    "status": "SUCCESS" || "FAILED" || "DELETED",  // string
    "message" "error message"                      // string
}
```

In the event of a "FAILED" or "DELETED" status, the TCP socket is closed by the Status Server. A "DELETED" status indicates that the printer was deleted in the cloud or has never registered. In either case, the printer must register itself with the Status Server.

A "FAILED" response indicates that either the `hello` command was missing required parameters or the signature did not agree with the public crypto key stored in the cloud for the printer.

2.10 keyPair

In response to a `makeKeyPair` command from a printer, the Status Server will generate a RSA key pair and return it with the `keyPair` command

Command: `keyPair`
Argument: a JSON object of the form

```
{
    "status": "SUCCESS" || "FAILED", // string
    "public": "public RSA key",      // string
    "private": "private RSA key"     // string
}
```

Note that if at all possible, printers should generate the RSA key pair themselves. Having the Status Server generate it and then transmit it over the Internet is suboptimal and admits the possibility of the private key being intercepted.

The printer must save the private RSA key in non-volatile memory. Each time the printer connects to the Status Server, it must use the private RSA key to digitally sign a challenge.

In the event of a failure, a `status` of `FAILED` is returned along with a reason in the `message` field,

```
{
    "status": "FAILED",
    "message": "reason for failure"
}
```

2.11 pause

Request that the printer pauses printing with the intention of later resuming the print. This command applies to both print jobs sent to the printer from the Polar Cloud as well as local print jobs.

Command: `pause`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",   // string, required
    "type": "filament" | "cold" | "pause"      // string, optional (defaults to "pause")
}
```

2.12 print

To initiate a print, the cloud will send to the printer a `print` command containing information necessary to execute the print. The printer is expected to perform an HTTP or HTTPS GET to retrieve from the Internet the necessary files.

Command: `print`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",                              // string
    "jobId": "print-job-id",                                              // string
    "stlFile": "full URL to the STL to slice",                            // string
    "configFile": "full URL to the Cura slicer settings",                 // string
    "gcodeFile": "full URL to the gcode for the print",                   // string
    "imageFile": "full URL to a PNG rendering of the STL",                // string
    "imageThumbnailFile": "full URL to a small PNG rendering of the job", // string
    "jobName": "user-supplied job name"                                   // string
}
```

For example,

```
{
    "serialNumber": "PB000112",
    "jobId": "PB000112-178572",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-cc
    "imageFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/OP000010/1227/joWL5lGz-objec
    "imageFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/OP000010/1227/joWL5lGz-objec
    "jobName": "Fulcrum test"
}
```

The `gcodeFile` field is only relevant to printers which cannot themselves slice a model. For those printers, the Polar Cloud will perform the slicing, saving the gcode in the cloud for retrieval by the printer. When `gcodeFile` is supplied, `stlFile`, `configFile`, `imageFile`, and `imageThumbnailFile` are not supplied.

The `jobName` field is a user-supplied name for the print job. Most often, it is the file name name of the underlying STL being printed. Printers may optionally display this field.

2.13 registerResponse

In response to a `register` command from a printer, the cloud responds with a `registerResponse` command. The response from the cloud provides a unique serial number which the printer must use or the string "FAILED" in the event of a failure.

If the same printer requests a serial number more than once, it will be given the same serial number back each time. The serial numbers are registered under the printer's ethernet MAC address.

Command: `registerResponse`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",   // string
    "status": "SUCCESS" | "FAILED",             // string
    "reason": "SUCCESS" | "SERVER_ERROR" |      // string
       "MFG_MISSING" | "MFG_UNKNOWN" |
       "EMAIL_PIN_ERROR" | "FORBIDDEN" |
       "INVALID_KEY"
}
```

The values of the `reason` field indicate the nature of a FAILED `status`:

- EMAIL_PIN_ERROR: the supplied e-mail address does not match an existing Polar Cloud account, the account does not have a PIN, or the supplied PIN does not match that of the account.
- FORBIDDEN: a printer already exists with the supplied MAC address and it is not owned by the supplied account.
- INVALID_KEY: an invalid public key value was supplied. The supplied public key must be a non-empty string.
- MFG_MISSING: the `register` command from the printer is missing the required `mfg` field.
- MFG_UNKNOWN: The `mfg` field in the `register` command specifies an unrecognized manufacture. Use `test` if testing.
- SERVER_ERROR: a server error of some sort has occurred; try again in a bit.

2.14 resume

The `resume` command is used to resume a print which has been paused. This command applies to both print jobs sent to the printer from the Polar Cloud as well as local print jobs.

Command: `resume`
Argument: a JSON object of the form

{ "serialNumber": "printer-serial-number" // string, required }

For example,

```
{
    "serialNumber": "P3D99979"
}
```

2.15 sendNextPrintResponse

When a `sendNextPrint` command from the printer fails or there are no prints queued, the Status Server will send back a `sendNextPrintResponse` command indicating the cause of the failure.

> If there are jobs available, then the Status Server responds to a `sendNextPrint` command by ending a `print` command. As such, the `sendNextPrintResponse` only arises when the `sendNextPrint` command fails.

Command: `sendNextPrintResponse`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number", // string, required
    "status": "NONE" || "FAILED",            // string, required
    "message": see below                     // string, required
}
```

A `status` of "NONE" indicates that either there are no queued jobs ready to print, or, if a specific job was requested, the job no longer exists or is not ready to print (e.g., may have been removed from the queue or may be reslicing).

> Note that the `getQueueResponse` only reports jobs which are ready to print. However, a Polar Cloud member may edit a queued job causing it to temporarily be unavailable for printing. In that situation, it is possible for a printer to be told a job exists, request that it be started, and then receive a response indicating that the job is now not ready to print.

Presently the `status` and `message` values are

`status`	`message`	**Description**
NONE	NO_QUEUED_JOBS	No ready jobs in queue
FAILED	INVALID_JOBID	Syntactically invalid `jobId` specified
FAILED	PRINTER_DELETED	The printer appears to have been deleted
FAILED	SERVER_ERROR	Server error prevents completion of `sendNextPrint` command

2.16 temperature

Tell a printer to set the target temperature of one or more heaters All temperatures are in degrees Celsius.

Command: `temperature`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",    // string, required
    "bed": temp,                                // number, optional
    "chamber": temp,                            // number, optional
    "tool0": temp,                              // number, optional
    "tool1": temp,                              // number, optional
    ...
}
```

The `bed` temperature, if supplied, is intended for a heated bed (platform). Similarly, `chamber` temperature, if supplied, is intended for the build chamber.

A temperature specification of 0 indicates that the corresponding heater should be turned off. If a heater is not present in the request, then its setting should be left unchanged.

2.17 unregisterResponse

In response to an `unregister` command received from a printer, the cloud sends back to the printer an `unregisterResponse` command indicating if the command succeeded or failed.

Command: `unregisterResponse`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",                          // string
    "status": "SUCCESS" | "FAILED",                                   // string
    "message": "Printer unregister successful" | "Printer not found"  // string
}
```

2.18 update

This command is sent at the initiation of the printer's owner or managers and intended to tell a printer to update its internal software.

Command: `update`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",   // string
}
```

2.19 welcome

As soon as a printer connects to the Status Server, the server will send a `welcome` command. The command contains a challenge string which the printer, once registered, must digitally sign using its private encryption key and send back with the `hello` command. The Status Server will ignore further commands from the printer until such time that a valid `hello` command is received. The exception is the `register` command: an unregistered printer can ignore the `welcome` challenge and send a `register` command. Once registered it should disconnect and reconnect.

Command: `welcome`
Argument: a JSON object of the form

```
{
    "challenge": "random text string to encrypt"    // string
}
```

3.0 Commands from Printers to the Cloud

3.1 capabilities

A printer may request from the cloud the list of any special cloud capabilities which the printer is allowed to have. The capabilities are returned with the `capabilitiesResponse` command. If the printer has no capabilities, then an empty list (array) is returned.

Command: `capabilities`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number"    // required, string
}
```

3.2 commandResponse

When the cloud member manually sends gcode commands to a printer, the printer may optionally send back the printer's response (output) to those gcode commands with the `commmandResponse` command. This is optional. The Polar Cloud primarily uses the information to aid cloud administrators when diagnosing a printer issue for a cloud member.

A multi-line response should use US-ASCII line feed characters, 0x0A, as line delimeters within the single returned string.

Command: `commandResponse`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",    // string, required
    "response": "string"                        // string, required
}
```

3.3 customCommandList

A printer may ask the cloud to present users with additional, printer specific commands. This is done with the `customCommandList` command. Each time this command is sent to the cloud, the list will be permanently saved in the cloud's databases. As such, it is not necessary to send this command each time the printer connects to the cloud.

Command: `customCommandList`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",          // string, required
    "customCommandList": [
        {
            "label": "menu-label-1",                  // string, required
            "command": "COMMAND-1",                   // string, requied
            "helpText": "command-help-1",             // string, optional
            "confirmText": "confirmation-request-1",  // string, optional
        }, ...
    ]
}
```

In the above, there can be zero or more custom commands, each one represented by an object. Each object must containt at least a `label` and `command` field. The label is the text placed into a menu label while the command is the actual command sent back to the printer by the cloud via a `customCommand` command. It need not be the actual command the printer should execute: it may just be a reference of some sort to the command the printer should execute.

The fields `helpText` and `confirmText` are optional and neither, one, or both may be supplied on a per command basis. The help text may be used by the cloud for a description displayed in a mouse hover-over event. The confirm text indicates that the cloud must ask for user confirmation before sending that particular command. The confirm text itself is used in the confirmation request presented to the user.

A simple `customCommandList` from a printer to the cloud might be,

```
{
    "serialNumber": "OP000001",
    "customCommandList": [
        {
            "label": "printer off",
            "command": "COMMAND-1",
            "helpText": "Turn your printer's power off",
            "confirmText": "You are about to turn off your printer."
        },
        {
            "label": "printer on",
            "command": "COMMAND-2",
            "helpText": "Turn your printer's power on"
        }
    ]
}
```

3.4 getQueue

A printer may request information on the jobs presently queued in the cloud to the printer and which are ready to print (i.e., are not waiting to be sliced should the printer require cloud-based slicing). This is done with the `getQueue` command which takes the form

Command: `getQueue`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number"   // string, required
    "skip": integer,                          // integer, optional
    "limit": integer,                         // integer, optional
}
```

The response, described under the `getQueueResponse` command in Section 2.7, provides basic information on each queued job.

The optional `skip` and `limit` parameters may be used for pagination; i.e., to control which queued jobs and how many are returned. For example, to return the first 10 jobs, use

```
"skip": 0,
"limit": 10
```

and the next 10 jobs with

```
"skip": 10,
"limit": 10
```

When not specified, both `skip` and `limit` default to the value 0. For `limit`. a value of 0 indicates "unlimited".

Note that the `getQueueResponse` always includes the total count of queued jobs thus allowing a paginated queue listing to always know the total count available.

3.5 getUrl

To store images and time-lapse videos in the cloud, a printer posts the images (JPEG) or videos (MP4) directly to the cloud storage using a pre-signed HTTPS POST URL. These pre-signed POST URLs are provided by the Status Server and are valid for a limited period of time, generally one or twenty-four hours.

The POST URLs specify the exact location and name of the file (in the cloud storage) as well as the maximum size of the file. If two files are uploaded with the same POST URL, the second file uploaded "survives", overwriting the first file uploaded. That is intentional: it is how, for example, consecutive camera images are handled by the Web Servers. The WebServers HTTPS GET the image from a fixed location -- the location the POST URL uploads to. By updating the web page every 20 seconds, the viewing cloud member sees the print evolve as it is printed; they see each new image after it is uploaded by the printer.

Three types of image data may be uploaded to the cloud by a printer:

- `idle`: camera images from the printer while it is idle and not printing **or** when it is printing a local print job.
- `printing`: camera images from the printer while it is printing a job sent to it from the cloud.
- `timelapse`: a MP4 time-lapse video of a print job from the cloud. See Section 1.4 for details on the proper encoding of the video.

No other types of data should be sent. If a printer is printing a local print job -- a job not supplied by the cloud -- only send up `idle` images for it if you wish cloud members to be able to view the printer as it prints the local job.

The pre-signed POST URLs received from the Status Server are then used in HTTPS POST requests. See Section 1.5 for further details. When a POST URL has expired, request a new one from the Status Server. It is up to the printer to track when the URL is expected to expire; use of it after it expires will result in a failed HTTPS POST attempt with an error message formatted in XML from the cloud storage service (presently Amazon's AWS S3 service). The pre-signed POST URLs returned by the status server include a field indicating for how many seconds the URL will be valid.

Command: `getUrl`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",         // string, required
    "method": "post",                                // string, required
    "type": "idle" | "printing" | "timelapse",       // string, required
    "jobId": "print-job-jobId"                       // string, see below
 }
```

The `jobId` field must be supplied for `printing` and `timelapse` types. It must be the `jobId` supplied with the received `print` command. If printing a local print job, only upload images of type `idle`.

Upon receipt of a `getUrl` command, the Status Server will respond with a `getUrlResponse` command.

3.6 hello

Upon connecting to the Status Server, a printer sends a `hello` command containing the JSON object

```
{
    "serialNumber": "printer's serial number",           // string, required
    "signature": "signed challenge",                     // BASE64 encoded string, required
    "MAC": "network MAC address in use by the printer"   // string, required
    "protocol": "protocol-version",                      // string, required
    "mfgSn": "manufacturer's serial number",             // string, optional
    "printerMake": "printer make",                       // string, optional
    "version": "currently install software version",     // string, optional
    "localIP": "printer's local IP address",             // string, optional
    "rotateImg": 0 | 1,                                  // integer, optional
    "transformImg": 0 - 7,                               // integer, optional
    "camOff": 0 | 1,                                     // integer, optional
    "camUrl": "URL for printer's live camera feed"       // string, optional
}
```

Of the above fields, only `serialNumber`, `signature`, and `protocol` are required.

After validating the supplied signature, the server will send back a `helloResponse` command. If the status indicated in that response is not "SUCCESS", then the server will shortly close the TCP socket.

The `protocol` field should have the value "2.0" indicating that the 2.0 protocol is being used to communicate with the Status Server. (This document describes the 2.0 protocol.)

The `signature` field is the SHA-256 digital signature of the challenge received from the Status Server in the server's `welcome` message. The digital signature must be signed using the printer's private RSA encryption key and the SHA-256 algorithim. Before transmission to the Status Server, the signature should be BASE64 encoded. The server will verify the signature with the previously shared public key. If it does not validate, the connection will be closed by the Status Server. If it does validate, the connection will be marked as validated and the printer may remain connected, sending commands for the `serialNumber` which was validated.

The `printerMake` field can supply one of the printer make names returned by the endpoint https://polar3d.com/api/v1/printer_makes using an HTTP GET request (no authentication needed). That endpoint returns a JSON object of the form

```
{
    "printerMakes": [ make-1, make-2, make-3, ... ]
}
```

in which `make-1`, `make-2`, `make-3`, ... are each strings.

The optional `mfgSn` field will be saved in the cloud and used for tracking the printer via its manufacturer's serial number. This field may also be supplied with the `register` command.

The `rotateImg` field tells the Polar Cloud whether or not the live camera feed needs to be rotated 180 degrees for proper display. Send a value of 0 if it does not need to be rotated and 1 otherwise. If a more complex transformation is needed, use the `transformImg` field instead. That field is a bitmask of three bits: rotate counter clockwise by 90 degrees ("rotate-90"), a reflection about the horizontal midline ("reflect-H"), and a reflection about the vertical midline ("reflect-V"),

bit	value	operation	description
0	1	reflect-H	reflection about the horizontal midline
1	2	reflect-V	reflection about the vertical midline
2	4	rotate-90	rotate counter clockwise by 90 degrees

The "rotate-90" operation will always be applied last. In this system, a rotation by 180 degrees is brought about by applying both reflections and specified with a `transformImg` value of 3 (decimal). For a rotation 90 degrees clockwise, request both reflections and the rotation -- the decimal value 7.

> **NOTE:** When both `rotateImg` and `transformImg` are specified, the `rotateImg` field is ignored. That is, the `transformImg` field takes precedence.
>
> **NOTE:** Camera and time-lapse videos uploaded to the cloud must be properly rotated before uploading to the cloud. The `rotateImg` and `transformImg` fields are merely to tell web browsers whether or not they need to transform the **live camera feed** for proper viewing.

The `camOff` field tells the Polar Cloud whether the camera exists and is enabled (1) or does not exist or is disabled (0).

Command: `hello`
Argument: a JSON object of the form shown above

Polar3D printers send additional data in the `hello` message, some of which is then stored for the printer to aid support staff. For example,

```
{
    "serialNumber": "P3D99979",
    "signature": "....",
    "transformImg": 0,
    "camOff": 0,
    "printerMake": "Polar3D 2.0",
    "version": "17.03.25.01",
    "MAC": "b8:27:eb:71:01:46",
    "localIP": "192.168.2.126",
    "interfaces": {
        "eth0": {
            "iface": "eth0",
            "mac": "B8:27:EB:71:01:46",
            "address": "none",
            "netmask": "none"
        },
        "wlan100": {
            "iface": "wlan1",
            "mac": "7C:DD:90:82:75:13",
            "address": "192.168.2.126",
            "netmask": "255.255.255.0"
        },
        "current": "wlan100",
        "current_eth": "eth0"
    }
}
```

3.7 job

Whenever a print job has successfully completed or been canceled, the printer should send to the cloud a `job` command indicating that the job is finished and its final state, completed or canceled. If the printer was executing a "local" print job -- a print job not sent from the cloud -- it should still send this command to the Status Server, but with the string "123" for the `jobId` field.

Upon receipt of this command from a printer, a completed print will be moved from the printer's queue of print jobs to the list of past prints. And if the print was canceled, the print will be left in the printer's queue.

Command: `job`
Argument: a JSON object of the form

```
{
    "serialNumber": "serial-number",      // string, required
    "jobId": "job id",                    // string, required
    "state": "completed" | "canceled",    // string, required
    "printSeconds": integer,              // integer, optional
    "filamentUsed": integer               // integer, optional
}
```

`jobId` is the string sent with the `print` command that started the print. Send the string "123" for if a local print job.

Use the `printSeconds` and `filamentUsed` fields to report on the print duration in seconds and the amount of filament used in millimeters. Do this for both local and cloud print jobs.

For example, upon completing job 178137, the printer P3D999979 would send

```
{
    "serialNumber": "P3D99979",
    "jobId": "P3D99979-178137",
    "state": "completed",
    "filamentUsed": 11872,
    "printSeconds": 3957
}
```

3.8 makeKeyPair

If a printer lacks sufficient computing resources to generate a RSA cryptographic key pair, then it may request that the Status Server generate a key pair and send it back to the printer. The printer must then save the private RSA key in non-volatile memory for future use. It must send the public RSA key back to the Status Server as part of its registration process.

Command: `makeKeyPair`
Argument: a JSON object of the form

```
{
    "type": "RSA",   // string, required
    "bits": 2048     // integer, required
}
```

The Status Server will respond with a `keyPair` command to the printer. That command will provide the public and private keys of a 2048 bit RSA keypair. For example,

```
{
    "public": "-----BEGIN RSA PUBLIC KEY-----\nMIIBCgKCAQEAq1E56TSUEh3NB0+Hdac6hFoy19yr2HN+ngR+lJguBgz
    "private": "-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEAq1E56TSUEh3NB0+Hdac6hFoy19yr2HN+ngR+l
}
```

3.9 register

A printer, upon first running, new printers must request that a unique serial number be assigned to the printer by the cloud. To initiate the process, the printer ignores the `welcome` command from the cloud and instead sends a `register` command back to the cloud. The `register` command from the printer includes:

1. The printer's ethernet MAC address,
2. The public key from an RSA key pair generated by the printer and which is unique to the printer,
3. The e-mail address associated with the printer owner's Polar Cloud account, and
4. The Polar Cloud account's PIN code. A PIN code may be 4 to 16 decimal digits.

The printer must be capable of generating an RSA key pair and digitally signing short strings with its private key (`hello` command). Additionally, the printer's owner must first create a Polar Cloud account before allowing their printer to register itself: while servicing a printer registration request, the Status Server will use the supplied e-mail address to associate the printer with the specified Polar Cloud account. The PIN code serves to ensure the registering user truly has access to that account as opposed to merely knowing the account owner's email address.

If the printer has more than one ethernet MAC address, provide only one of the MAC addresses. Use a deterministic method to select which MAC address to supply: in the event that the printer loses its serial number, the MAC address is used to retrieve it from the cloud.

Command: `register`
Argument: a JSON object of the form

```
{
    "mfg": "agreed upon prefix for printer type",  // 'pb', 'xyz', 'ff', 'octoprint'
    "email": "owner's Polar Cloud account",        // string, required
    "pin": "Polar Cloud account's PIN code",       // string, required
    "publicKey": "BASE64 encoded public key",      // string, required
    "mfgSn": "Manufacturer's serial number",       // string, optional
    "myInfo": {
        "MAC": "Polar Box's ethernet MAC address"
    }
}
```

Once successfully registered as indicated by receipt of a `registerResponse` command from the Status Server, the printer should disconnect from the Status Server and then reconnect. Upon reconnecting it will be sent a new `welcome` command to which it can respond with `hello` and validate the connection with its new serial number.

Note that the `myInfo` sub-object of the `register` command can contain additional information. Presently, only the MAC field is used from the `myInfo` sub-object.

For reference PIN codes in the Polar Cloud are, by default, 4 decimal digits. Polar Cloud members may change their PIN code and may use a code of length 4 to 16 decimal digits in length.

If supplied, the optional `mfgSn` field will be saved in the cloud and used for tracking the printer via its manufacturer's serial number. This field may also be supplied with the `hello` command.

3.10 sendNextPrint

If a printer has the `sendNextPrint` capability, then it may issue a `sendNextPrint` command to request the next queued print job, if any exists in the printer's queue. If a queued print exists, then the cloud will send back to the printer a `print` command for that queued print job. If there are no queued print jobs or the printer lacks the capability, then no response is generated by the cloud.

The optional `jobId` parameter may be used to request that a specific job from the queue be started. A list of queued jobs is obtained with the `getQueue` command.

Command: `sendNextPrint`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer-serial-number",   // string, required
    "jobId": "job-id"                          // string, optional
}
```

When the `sendNextPrint` command succeeds, a `print` command is sent by the Status Server to the printer. If there are no queued jobs or an error occurs, then a `sendNextPrintResponse` will be sent by the Status Server to the printer. This response is primarily of interest when a specific job is requested with the optional `jobId` parameter.

3.11 setVersion

A printer may wish to inform the cloud as to the version of software it is running as well as the latest version available. The cloud can present this information to the printer's owner and managers and, should the latest version be greater than the running version, present an UPDATE button in the cloud's web interface. Selecting that button will then cause the cloud to send the printer an `update` command.

Command: `setVersion`
Argument: a JSON object of the form

```
{
    "serialNumber": "string",   // required, string
    "runningVersion": "string", // required, string
    "latestVersion": "string"   // optional, string
}
```

The version strings may take any form provided that a latest version, when lexically compared to the running version statisfies the requirement that if `latestVersion` is greater than `runningVersion` then `latestVersion` is a newer version.

For example, for the JSON object

{ "serialNumber": "OP000123", "runningVersion": "1.3.3", "latestVersion": "1.3.4" }

the printer would be considered to be running version 1.3.3. Morever, a newer version, 1.3.4, would be available and the owner and managers would see an UPDATE button which they could elect to click on the management pages for printer OP000123.

3.12 status

Each printer reports its status several times a minute to the cloud. This is the primary command which printers send to the Status Servers. In the JSON object described below, only the `status` and `serialNumber` fields are required. All other fields are optional.

Command: `status`
Argument: a JSON object of the form

```
{
    "serialNumber": "string",
    "status": integer,
    "progress": "string",
    "progressDetail": "string",
    "estimatedTime": integer,
    "filamentUsed": integer,
    "startTime": "string",
    "printSeconds": integer,
    "bytesRead": integer,
    "fileSize": integer,
    "tool0": floating-point,
    "tool1": floating-point,
    "bed": floating-point,
    "chamber": floating-point,
    "targetTool0": floating-point,
    "targetTool1": floating-point,
    "targetBed": floating-point,
    "targetChamber": floating-point,
    "door": integer,
    "jobId": "string",
    "file": "string",
    "config": "string"
}
```

The interpretation of the fields are as follows

serialNumber

The printer's serial number.

status

The present status of the printer expressed as an integer value and chosen from

Value	Interpretation
0	Ready; printer is idle and ready to print
1	Serial; printer is printing a local print over its serial connection
2	Preparing; printer is preparing a cloud print (e.g., slicing)
3	Printing; printer is printing a cloud print
4	Paused; printer has paused a print
5	Postprocessing; printer is performing post-printing operations
6	Canceling; printer is canceling a print from the cloud
7	Complete; printer has completed a print job from the cloud
8	Updating; printer is updating its software
9	Cold pause; printer is in a "cold pause" state
10	Changing filament; printer is in a "change filament" state
11	TCP/IP; printer is printing a local print over a TCP/IP connection
12	Error; printer is in an error state
13	Disconnected; controller's USB is disconnected from the printer
14	Door open; unable to start or resume a print
15	Clear build plate; unable to start a new print

The states 6 and 7 should be sent for several `status` updates to ensure receipt by the cloud. The other states should be sent in each status update during which they are relevant.

progress

A UTF-8 string briefly expressing the current status of the printer in human-readable text. E.g., "Idle", "Printing", etc.

progressDetail

A UTF-8 string with more detail on the status. If printing, the cloud would like to see "Percent Complete: xx%" in this string. E.g.,

```
Printing Job: nnnn Percent Complete: xx%
```

estimatedTime

The estimated print time in seconds. Possibly generated by the slicer.

filamentUsed

The estimated amount of filament which will be used by the print when completed, in millimeters. This is not the filament used so far.

startTime

The time the print started expressed as an ISO 8601 date/time string. E.g., "1970-01-01T00:00:00Z".

printSeconds

The elapsed number of seconds spent printing the current print.

bytesRead

The number of bytes read so far from the gcode file (or X3G, or whatever file).

fileSize

The size in bytes of the file of printing instructions (gcode, X3G, etc.).

tool0, tool1, bed, chamber

The temperatures in degrees Celsius of the first and second extruder, chamber, and heated print bed.

targetTool0, targetTool1, targetBed, targetChamber

The target temperaturs in degrees Celsius of the first and second extruder, heated chamber, and heated print bed.

door

The status, if available of the door to the printer's build chamber. | value | Description | | ----: | :---------- | | 0 | door open, unlocked | | 1 | door closed, unlocked | | 2 | door open, locked | | 3 | door closed, locked |

jobId

The value of the `jobId` sent by the cloud with the `print` command. If printing a local file, send the string "123".

stlFile, configFile

The URLs of the STL file and slicing configuration file sent by the `print` command.

3.12.1 Ready to print status update

```
{
    "serialNumber": "P3D99979",
    "status": 0,
    "progress": "",
    "progressDetail": "",
    "estimatedTime": 0,
    "filamentUsed": 0,
    "startTime": 0,
    "printSeconds": 0,
    "bytesRead": 0,
    "fileSize": 0,
    "tool0": 23.1,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 0,
    "jobId": 0,
    "stlFile": null,
    "configFile": null
}
```

3.12.2 Preparing a print

Below are three samples of status updates for the status of 2, preparing a print.

```
{
    "serialNumber": "P3D99979",
    "status": 2,
    "progress": "Preparing to print a job",
    "progressDetail": "Downloading file for job: P3D99979-178137",
    "estimatedTime": 0,
    "filamentUsed": 0,
    "startTime": "2017-03-29T14:00:43.895Z",
    "printSeconds": 0,
    "bytesRead": 0,
    "fileSize": 0,
    "tool0": 23.1,
    "tool1": 0,
    "bed": 0.0,
    "targetTool0": 0,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-cc
}
```

and

```
{
    "serialNumber": "P3D99979",
    "status": 2,
    "progress": "Preparing to print a job",
    "progressDetail": "Downloading config file for job: P3D99979-178137",
    "estimatedTime": 0,
    "filamentUsed": 0,
    "startTime": "2017-03-29T14:00:43.895Z",
    "printSeconds": 0,
    "bytesRead": 0,
    "fileSize": 0,
    "tool0": 23.1,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 0,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-co
}
```

and

```
{
    "serialNumber": "P3D99979",
    "status": 2,
    "progress": "Job Printing",
    "progressDetail": "Slicing Job: P3D99979-178137",
    "estimatedTime": "105",
    "filamentUsed": "167",
    "startTime": "2017-03-29T14:17:45.087Z",
    "printSeconds": 0,
    "bytesRead": 0,
    "fileSize": 50845,
    "tool0": 152.0,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 150,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-co
}
```

3.12.3 Printing

```
{
    "serialNumber": "P3D99979",
    "status": 3,
    "progress": "Job Printing",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 49.7%",
    "estimatedTime": "105",
    "filamentUsed": "167",
    "startTime": "2017-03-29T14:17:45.087Z",
    "printSeconds": 166,
    "bytesRead": 25268,
    "fileSize": 50845,
    "tool0": 185.1,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 185,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-co
}
```

And an example of a job which was canceled (killed),

```
{
    "serialNumber": "P3D99979",
    "status": 3,
    "progress": "Killing Job",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 12.0%",
    "estimatedTime": "3259",
    "filamentUsed": "4152",
    "startTime": "2017-03-29T14:01:08.335Z",
    "printSeconds": 932,
    "bytesRead": 262127,
    "fileSize": 2183901,
    "tool0": 185.3,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 185,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-co
}
```

3.12.4 Paused

```
{
    "serialNumber": "P3D99979",
    "status": 4,
    "progress": "Job Paused",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 10.9%",
    "estimatedTime": "3259",
    "filamentUsed": "4152",
    "startTime": "2017-03-29T14:01:08.335Z",
    "printSeconds": 821,
    "bytesRead": 241983,
    "fileSize": 2183901,
    "tool0": 185.2,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 185,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-co
}
```

3.12.5 Print finishing

```
{
    "serialNumber": "P3D99979",
    "status": 5,
    "progress": "Post processing job",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 100.0%",
    "estimatedTime": "105",
    "filamentUsed": "167",
    "startTime": "2017-03-29T14:17:45.087Z",
    "printSeconds": 227,
    "bytesRead": 50845,
    "fileSize": 50845,
    "tool0": 178.3,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 0,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-co
}
```

3.12.7 Print finished

```
{
    "serialNumber": "P3D99979",
    "status": 7,
    "progress": "Complete",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 100.0%",
    "estimatedTime": "105",
    "filamentUsed": "167",
    "startTime": "2017-03-29T14:17:45.087Z",
    "printSeconds": 227,
    "bytesRead": 50845,
    "fileSize": 50845,
    "tool0": 145.8,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 0,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/yPad5lml-co
}
```

3.12.9 Cold pause

```
{
    "serialNumber": "P3D99979",
    "status": 9,
    "progress": "Cold Pause",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 11.4%",
    "estimatedTime": "3259",
    "filamentUsed": "4152",
    "startTime": "2017-03-29T14:01:08.335Z",
    "printSeconds": 856,
    "bytesRead": 247875,
    "fileSize": 2183901,
    "tool0": 185,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 0,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-co
}
```

3.12.10 Changing filament

```
{
    "serialNumber": "P3D99979",
    "status": 10,
    "progress": "Changing Filament",
    "progressDetail": "Printing Job: P3D99979-178137 Percent Complete: 11.5%",
    "estimatedTime": "3259",
    "filamentUsed": "4152",
    "startTime": "2017-03-29T14:01:08.335Z",
    "printSeconds": 907,
    "bytesRead": 251005,
    "fileSize": 2183901,
    "tool0": 183.5,
    "tool1": 0,
    "bed": 0,
    "targetTool0": 185,
    "jobId": "P3D99979-178137",
    "stlFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-objec
    "configFile": "https://s3.amazonaws.com/dev2.polar3d.com/files/printer/P3D99979/178137/JMaeKlWv-co
 }
```

3.13 unregister

When a printer owner unregisters their printer from the cloud, this command is sent by the printer to the cloud. Upon receipt, the cloud will send back to the printer an `unregisterResponse` command with success or failure information.

Command: `unregister`
Argument: a JSON object of the form

```
{
    "serialNumber": "printer's serial number"  // string, required
}
```

4.0 Revision History

27 December 2017

1. Added status value 15, Clear build plate, to the list of printer status values. This value may be used to indicate that a print has finished, but a new print cannot be started until the build plate is first cleared and an action taken by the printer's operator (e.g., pressing an "Okay" button on the printer's LCD screen).

2 December 2017

1. Added `imageThumbnailFile` to the `getQueueResponse` and `print` commands.

2. Added the optional `mfgSn` field to the `register` and `hello` commands.

28 November 2017

1. Added "INVALID_KEY" failure case to `registerResponse` result.

2 November 2017

1. Added `jobName` field to `getQueueResponse`.

31 October 2017

1. Added Section 0.1 with tables of the commands.

26 October 2017

1. Added `status` values of 13 and 14 to the `status` command sent from the printer to the Status Server.
2. Added `door` value to the `status` command from the printer to the Status Server.
3. Added `getQueue` and `getQueueResponse` commands. The printer may request from the Status Server information on the currently queued jobs with the `getQueue` command. The Status Server responds with a `getQueueResponse` command which provides a listing of all the queued jobs for the printer.
4. Extended the `sendNextPrint` command to have an optional `jobId` parameter. When that parameter is specified, the requested job will be moved to the head of the queue and started printing.
5. Added a `sendNextPrintResponse` which the Status Server will send to the printer whenever a `sendNextPrint` command fails or there are no jobs queued to start.

20 October 2017

1. Added the `delete` command from the Status Server to the printer. This command is sent to the printer if it is connected to the Polar Cloud when the printer owner deletes it in the cloud.

19 October 2017

1. Added `imageFile` field to the `print` command's data. This field provides a URL to the rendering of the print job. The renderings are typically PNG files. No renderings are available for print jobs created from a user-uploaded gcode file.

18 October 2017

1. Added `helloResponse` command from Status Server to the printer. Primary purpose is to allow a printer to detect when it has been deleted in the cloud and needs to re-register.

17 October 2017

1. Added `chamber` and `targetChamber` temperature fields.

9 October 2017

1. Indicate the length of possible PIN codes (4 - 16 decimal digits).

15 August 2017

1. Added the `makeKeyPair` and `keyPair` commands. To be used by printers with insufficient compute resources to generate a 2048 bit RSA key pair.

5 August 2017

1. Added `printSeconds` and `filamentUsed` to the `job` command. This can be used to report time and materials used by local print jobs. For local print jobs, the information is otherwise not available.

2 August 2017

1. Added to the `print` command the `jobName` field. This field is the user-supplied name for the print job. Printers may wish to display that name on any informational display while printing. Printers which save the STL file for the print may want to use the `jobName` when saving the STL to storage. Printers may choose to ignore this field.

18 July 2017

1. Added the `capabilities` command which may be sent by a printer to request any special capabilities permitted for the printer (e.g., `sendNextPrint`).

2. Added the `capabilitiesResponse` command sent from the cloud to a printer in response to a `capabilities` request from the printer.

3. Added the `sendNextPrint` command. With that command a printer may request that the next available print job be sent. A printer must have the `sendNextPrint` capability in order to use that command.

2 July 2017

1. Updated text on image uploading to make it clear that the printer should flip any upside camera images and time-lapse videos before uploading to the cloud.

27 June 2017

1. Added a failure reason field `reason` to the `registerResponse` command

26 June 2017

1. Added printer to cloud command `customCommandList` to allow a printer to inform the cloud of custom commands to present to a user of that printer.
2. Added cloud to printer command `customCommand` to tell the printer to execute one of its custom commands.

8 June 2017

1. Added specification of RSA digital signing algorithm: SHA-256.

5 June 2017

1. Corrected local IP address field of `hello` command to read as `localIP` and not `localIp`.

4 June 2017

1. Added optional `printerMake` field to the `hello` command from a printer to the Status Server.
2. Added directions on how to obtain a list of printer makes to the `hello` command description.

1 June 2017

1. For status updates, changed temperature and target temperature field names to `tool0`, `tool1`, `bed`, `chamber`, `targetTool0`, `targetTool1`, `targetBed`, and `targetChamber`. This makes the names more consistent and in agreement with the command to set target printer temperatures.
2. Corrected status udpate examples to correctly show temperature values as floating point numbers and not strings.
3. Added `camUrl` to the `hello` command from the printer. This URL provides the cloud with the URL for the printer's live camera feed. If the cloud's web server detects that a member viewing a printer is on the same network as the printer, it will preferrentially display the live camera feed from the printer using this URL. That as opposed to displaying a static camera snapshot previously uploaded to the cloud by the printer.
4. Added a `setVersion` command which the printer may send to the Status Server to inform the Polar Cloud as to the current and latest software versions available for the printer. If the cloud knows the latest software version and it is lexically greater than the version the printer is running, then an UPDATE button will be displayed in the cloud on the printer's management page. If the printer owner or manager clicks that button, an `update` command will then be sent to the printer.
5. Re-ordered some of the commands listed in Section 3.0.