



# Netfabb Application Server

Documentation, July 2019

## Contents

Overview and Architecture .....	3
Task Handler.....	3
Installation.....	4
Configuration.....	5
API Documentation for the Task Handler .....	6
General Request Structure .....	6
New Session Request .....	7
Session authenticate Request .....	7
New Task Request .....	9
Get Task Status Request.....	10
Authentication and task creation example using Postman.....	11
More examples.....	12



## Overview and Architecture

Netfabb Application Server is an Addon Server tool for Netfabb Ultimate that allows an easy task scheduling and distributed automated data processing. Netfabb Application Server also serves as a reference implementation to demonstrate the capabilities of Netfabb's integrated network APIs.

```
R:\NetfabbStorageServer\Bin\NetfabbTaskServer.exe

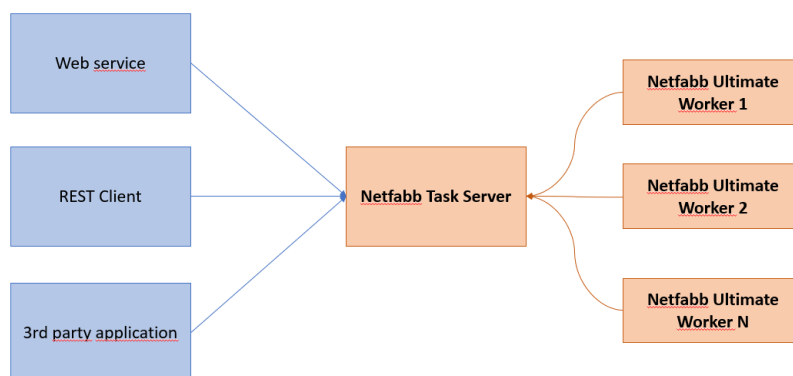
.';::;'.          .oxddol:.
.:oodxkkd:.      'xOkkxk:
,clodxkk00x:.    'xOkkkkk;
;llodxkk0000x,   'xOkkkkk;
,cldddddxx0000d' .xOkkxk;
.:codoc:::cox0001;. .xOkxxxx;
.:loolllecc:cd0000x:. .dkxxxxx;
.:odoooll111ccok00000d, 'dkxxxxo.
.:oddoooclo11cok00000kooxxxdxo.
.cdddddxl.,lolloxOk00kkxxdddxo.
.cdxdxx: .:oollooodxxxxdddxo.
.lxxxxd, .:lllllccloooodd'
.oxxxxx, .:clllccclloodd'
.lxxxxx, .:clccclloodl.
:kkkkkx, .:llclloo,
cOkkkk; .:llc:.
:jjjj;.    ...

Autodesk
Netfabb

2018/08/30 20:22:40 Autodesk Netfabb Task Server (v1.0.0)
2018/08/30 20:22:40 Initializing Log Database..
2018/08/30 20:22:40 Logging to ./logs/log_20180830_202240.db..
2018/08/30 20:22:40 Listening on port 8651..
```

## Task Handler

**The task handler part** of the application server acts as a message handler that queues task requests from various clients, queues them in an internal database and dispatches them to running netfabb Ultimate instances that can handle the tasks via LUA script and post the results to the clients.



The current implementation 1.0.0 (Protocol Version 2.0.0) is a minimal example implementation and will get further refinements over several releases. Currently out of scope, but open roadmap items are:











- More authentication options.
- Automatic retry handling of task timeouts, i.e. in case of a disappearing worker a task will remain INPROCESS without being reclaimed by another worker.

- Client authorization (every client can currently submit jobs, once a job is submitted, it is protected by a non-guessable secret).
- The application server only handles task messages and no binary payload data. This will be part of a second storage server version. We currently recommend the use of a network storage (either SMB drive or other REST-based storage services).
- Windows System Service: The current database server is a command line utility which is not embedded in the windows service framework.
- Server configuration and discovery: The application server will be configurable by a proper configuration system.

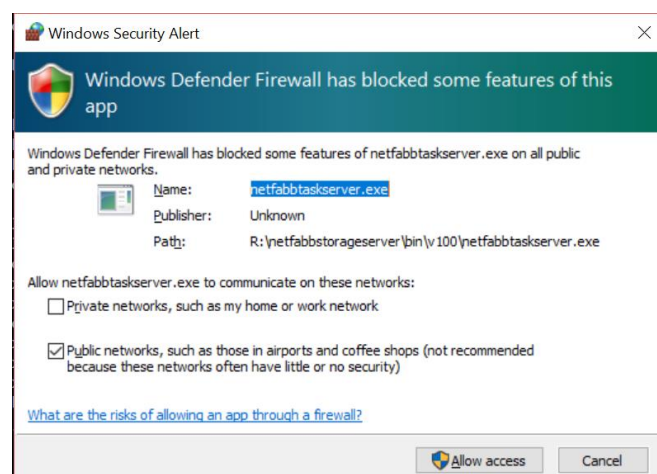
A simple message parameter protocol allows to propagate errors and success messages back to the client.

## Installation

The current installation package is very lightweight and comes with a SQLite database.

	data	23.11.2018 09:40	File folder	
	Documentation	23.11.2018 09:40	File folder	
	Examples	23.11.2018 09:40	File folder	
	logs	23.11.2018 09:40	File folder	
	example.crt	20.11.2018 14:23	Security Certificate	2 KB
	example.key	20.11.2018 14:23	KEY File	2 KB
	netfabbapplicationserver.db	23.11.2018 07:14	Data Base File	178 KB
	NetfabbApplicationServer.exe	23.11.2018 02:11	Application	13.653 KB
	netfabbapplicationserver.xml	22.11.2018 13:10	XML Document	1 KB
	setup_firewall_rules.bat	22.11.2018 13:10	Windows Batch File	2 KB

The application server is a command line daemon that will open a REST endpoint, by default at port 8650.



Logs are written in a SQLite database format into the logs directory. We recommend an application like “DB Browser for SQLite” (<http://sqlitebrowser.org/>) for accessing the log files.

Please note, that the default installation location under “Program Files” is not the best place for this server program, as you might not have the permissions on your system to write data under this directory. Also, you need administrator permissions on your system to start the program.

## Configuration

The application server comes with a minimal XML configuration file:

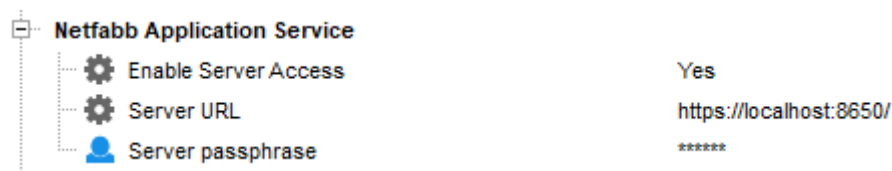
```
<!--
Netfabb Application Server
-->
<config xmlns="http://schemas.autodesk.com/netfabb/applicationserver/2018/11">
  <server host="127.0.0.1" port="8650" />
  <log prefix="./logs/log_" />
  <data directory="./data/" />
  <database type="sqlite" filename="netfabbapplicationserver.db" />
  <https type="tls" certificate="example.crt" privatekey="example.key" />
  <authentication type="passphrase" sessionduration="36000">
    <global passphrase="admin" salt="" />
    <nameduser id="GSM8MPQXTZDA"
passphrase="b727bc64862e65e24b4a86c5bd1826cd738e167d" salt="X1234" />
  </authentication>
</config>
```

The settings here allow to specify:

Setting	Default / Notes
<b>server host</b>	the servers IP address (default: 127.0.0.1)
<b>port</b>	TCP port (default 8650)
<b>log prefix</b>	a prefix for the log files and path
<b>data directory</b>	a directory for all data files
<b>database type</b>	the database type (currently only SQLite)
<b>filename</b>	the DB file
<b>https type</b>	Enabling of SSL: “http type=’tls’” – otherwise leave the field empty
<b>certificate</b>	The path to your certificate
<b>privatekey</b>	The path to your key file
<b>authentication type</b>	
<b>sessionduration</b>	Duration until session timeout

<b>global passphrase</b>	A global shared secret
<b>salt</b>	If the salt is empty, the password is transferred in clear, otherwise a sha1 sum, which is calculated by (sha1 (sha1(salt)+cleartext-password))
<b>nameduser</b>	Named user allow a rudimentary user management to separate them out, not knowing the main passphrase. Generally, there are no different access permissions yet, all is allowed for everyone authenticated

Please note, that when used the salted password must be entered into the config, on server side and into the Netfabb client side in the settings:



To ease this process the installation comes with a small “PasswordSalter.exe” utility program. This program is a easy to use command line program with the following parameters:

Parameter	Explanation
<b>--help</b>	Shows this help
<b>-p password</b>	Password to be salted
<b>-s salt</b>	Input Salt (generated if not provided)

## API Documentation for the Task Handler

### General Request Structure

Every request is a simple REST call to a HTTP Endpoint running on the specific application server IP and port. At this stage all requests are either a GET request or a POST request with a specific JSON body with at least two values:

- **protocol:** Protocol schema string. Identifier for the request protocol type. Currently all protocols are of the form “com.autodesk.netfabbtasks.\*”
- **version:** Protocol version string. This is a protocol version string that is currently unique across all requests and responses that a application server will take. In the future this might be split up to a per request versioning.

The return value is always a JSON object containing the above two string fields protocol and version. In case of a processing error, the protocol will be set to “com.autodesk.error”, and two more string fields are added:

- **errormessage:** A plain text error string describing the issue.
- **loguuid:** A unique identifier that allows to reconstruct the log of the request file in the log database of the application server.

Authentication headers are not supported yet in the protocol.

**Example error response:**

```
{
  "protocol": "com.autodesk.error",
  "version": "2.0.0",
  "errorMessage": "Could not find job: ee66d549-aaac-4200-b5ba-15d7ff0c225c",
  "loguuid": "7386c87b-7eae-48f9-b9c9-c5daeb91d606"
}
```

## New Session Request

For getting a new session a POST request has to be send to the “session/new” end point.

Endpoint	/session/new
Method	POST
Protocol Schema	com.autodesk.netfabbsession.new
Body definition	- <b>userid</b> : userid set in the config of the server
Return values	- <b>sessionuuid</b> : Unique identifier string for identifying the session the worker is working on

**Example request:**

```
{
  "protocol": "com.autodesk.netfabbsession.new",
  "version": "2.0.0",
  "userid": "test"
}
```

**Example response:**

```
{
  "protocol": "com.autodesk.netfabbsession.new",
  "version": "2.0.0",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "authtype": "saltedhash",
  "userid": "test",
  "salt": "563d2c466abdc3dc0328ca26b938ab03691be96"
}
```

## Session authenticate Request

In order to use the new session created it has to been authenticated via the “session/auth” endpoint.

Endpoint	/session/auth
Method	POST

Protocol Schema	com.autodesk.netfabbsession.auth
Body definition	<ul style="list-style-type: none"> <li>- <b>authtype</b>: Has to be “saltedhash”</li> <li>- <b>sessionuuid</b>: The session uuid retrieved with the new session request</li> <li>- <b>authkey</b>: The authkey is calculated as followed: sha1(NETFABB\$sessionuuid\$saltedpassphrase) where the \$sessionuuid is the session UUID retrieved in the new session request and the \$saltedpassphrase is the passphrase of the user saved in the config</li> </ul>
Return values	<ul style="list-style-type: none"> <li>- <b>token</b>: Session token to be used with further requests</li> </ul>

**Example request:**

```
{
  "protocol": "com.autodesk.netfabbsession.auth",
  "version": "2.0.0",
  "authtype": "saltedhash",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "authkey": "18863ac2a57212d24c0e61acd486ebe00b3ba754"
}
```

**Example response:**

```

{
  "protocol": "com.autodesk.netfabbsession.auth",
  "version": "2.0.0",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "token": "eyJzZXNzaW9uIjoizjQwNmJiNjYtYTFiMS00OGY1LTk4ZGQtMDNjZk4ZTQzNTcxIiwidXNlcmIiOiJjoidGVzdCJ9"
}

```



## New Task Request

Creating a task needs a POST request to the corresponding “new task” end point.

Endpoint	/tasks/new
Method	POST
Protocol Schema	com.autodesk.netfabbtasks.new
Authentication	Bearer sessiontoken
Body definition	<ul style="list-style-type: none"> <li>- <b>name:</b> Task identifier string. The Workers will poll for this name to run the correct worker script.</li> <li>- <b>parameters:</b> Key value pairs of arbitrary string parameters that determine the payload of the task description.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>- <b>uuid:</b> Unique identifier string for identifying the task and polling the result information.</li> </ul>

### Example request:

POST ▾	http://localhost:8651/tasks/new	Params
<pre>{   "protocol": "com.autodesk.netfabbtasks.new",   "version": "2.0.0",   "name": "slicestl",   "parameters": {     "inputSTL": "C:/Users/Joy/Downloads/STL Examples/STL Examples/100017U0_PositiveShelled_repaired.stl",     "outputSTL": "C:/Users/Joy/Desktop/slices/task/100017U0_PositiveShelled_repaired.zip"   } }</pre>		

### Example response:

```
{
  "protocol": "com.autodesk.netfabbtasks.new",
  "version": "2.0.0",
  "uuid": "ee66d549-aaac-4200-b5ba-15d75f0c225c"
}
```

## Get Task Status Request

Retrieving the task status needs a GET request to the corresponding “task status” end point.

Endpoint	/tasks/<uuid>
Method	GET
Protocol Schema	com.autodesk.netfabbtasks.status
Authentication	Bearer sessiontoken
Body definition	n/a
Return values	<ul style="list-style-type: none"> <li>- <b>uuid</b>: Unique identifier string for identifying the task and polling the result information.</li> <li>- <b>status</b>: Current task status. Valid values are “NEW”, “INPROCESS”, “SUCCESS”, “ERROR”, “CANCELED”, “RETURNED”</li> <li>- <b>name</b>: Task identifier string. The Workers will poll for this name to run the correct worker script.</li> <li>- <b>parameters</b>: Key/Value pairs of arbitrary string parameters that determine the payload of the task description.</li> <li>- <b>results</b>: Key/Value pairs of arbitrary string results that give output information of the processes.</li> <li>- <b>worker</b>: Name of the worker instance that has processed the task</li> <li>- <b>timestamp</b>: Unix timestamp when the task was created.</li> </ul>

### Example request:

GET ▼	http://localhost:8651/tasks/2cbf08e4-3765-4ca5-8bc4-8e59f563377a	Params
-------	--	--------

### Example response:

```
{
  "protocol": "com.autodesk.netfabbtasks.status",
  "version": "2.0.0",
  "uuid": "ee66d549-aaac-4200-b5ba-15d75f0c225c",
  "status": "NEW",
  "name": "slicestl",
  "parameters": {
    "inputSTL": "C:/Users/Joy/Downloads/STL Examples/STL Examples/100017U0_PositiveShelled_repaired.stl",
    "outputSTL": "C:/Users/Joy/Desktop/slices/task/100017U0_PositiveShelled_repaired.zip"
  },
  "result": null,
  "worker": "",
  "timestamp": "2019-02-08T11:10:43+01:00"
}
```



## Authentication and task creation example using Postman

### 1. Create a session:

#### POST :

<http://localhost:8650/session/new>

#### Request:

```
{
  "protocol": "com.autodesk.netfabbsession.new",
  "version": "2.0.0",
  "userid": "test"
}
```

#### Response:

```
{
  "protocol": "com.autodesk.netfabbsession.new",
  "version": "2.0.0",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "authtype": "saltedhash",
  "userid": "test",
  "salt": "563d2c466abdc3dc0328ca26b938ab03691be96"
}
```

The sessionuuid is to be used in the next request.

### 2. Authenticate the session

#### Example request:

```
{
  "protocol": "com.autodesk.netfabbsession.auth",
  "version": "2.0.0",
  "authtype": "saltedhash",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "authkey": "18863ac2a57212d24c0e61acd486ebe00b3ba754"
}
```

Auth key: SHA1(NETFABB88660803-7283-45f1-aa9c-227be74d6503admin)

i.e. take SHA1 of string NETFABB`sessionuid`admin

#### Example response:

```
{
  "protocol": "com.autodesk.netfabbsession.auth",
  "version": "2.0.0",
  "sessionuuid": "f406bb66-a1b1-48f5-8ddd-03cf98e43571",
  "token": "eyJzZXNzaW9uIjoizjQwNmJiNjYtYTFiMS00OGY1LTkZGQ0tMDNjZjk4ZTQzNTcxIiwidXNlcmIjoiZGVzZCJ9"
}
```

The token is to be used next as Bearer Token



### 3. Create the Task

**POST :**

<http://localhost:8650/tasks/new>

**Request:**

```
{
  "protocol": "com.autodesk.netfabbtasks.new",
  "version": "2.0.0",
  "name": "slicestl",
  "parameters": {
    "inputSTL": "C:/Users/Joy/Downloads/STL Examples/STL Examples/100017U0_PositiveShelled_repaired.stl",
    "outputSTL": "C:/Users/Joy/Desktop/slices/task/100017U0_PositiveShelled_repaired.zip"
  }
}
```

### [More examples](#)

Look into:

- for a full 3-step workflow (create, handle and query task) demonstration see the “TaskHandlerClientDemo.lua” example in the Netfabb Lua script library