Netfabb Application Server

Documentation, July 2019

Contents

[Overview and Architecture 3](#_Toc524063)

[Task Handler 3](#_Toc524064)

[Installation 4](#_Toc524065)

[Configuration 5](#_Toc524066)

[API Documentation for the Task Handler 6](#_Toc524067)

[General Request Structure 6](#_Toc524068)

[New Session Request 7](#_Toc524069)

[Session authenticate Request 8](#_Toc524070)

[New Task Request 9](#_Toc524071)

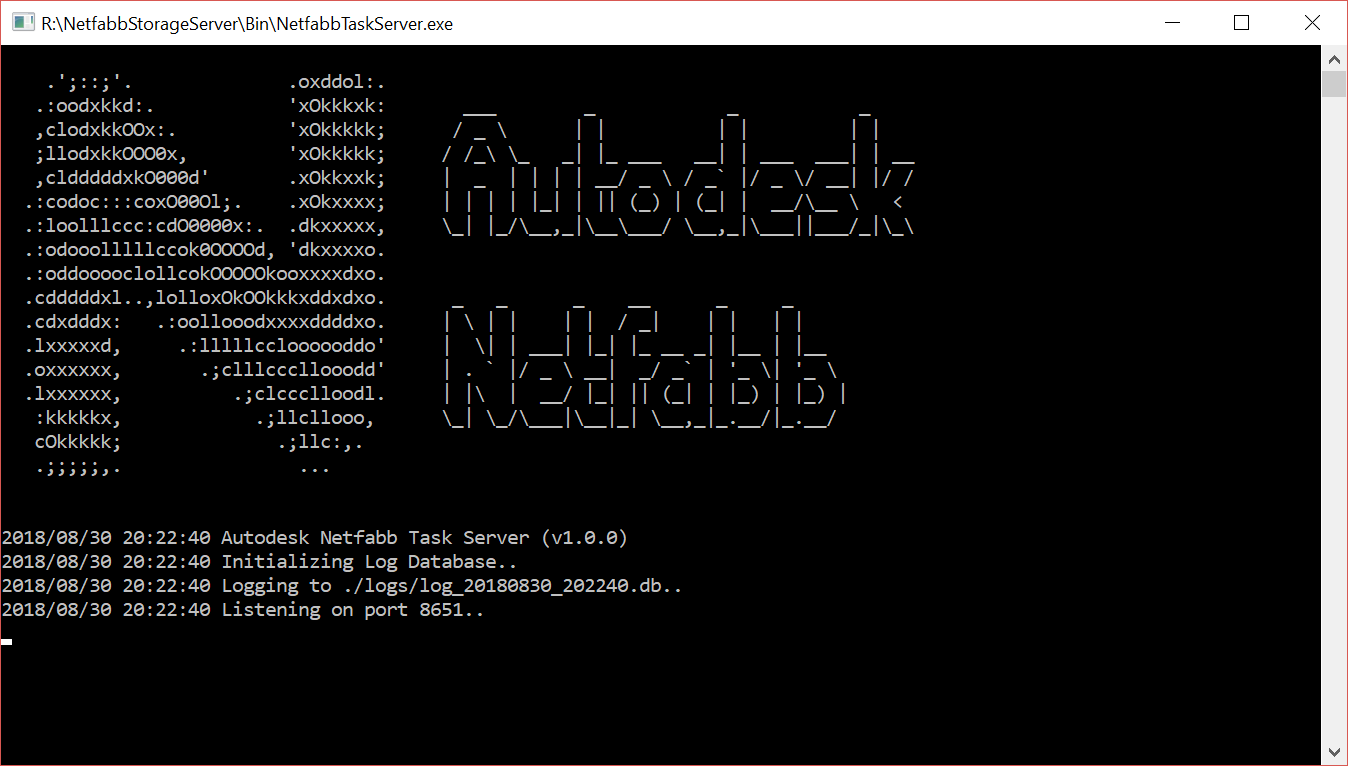
[Get Task Status Request 10](#_Toc524072)

[Authentication and task creation example using Postman 11](#_Toc524073)

[More examples 12](#_Toc524074)

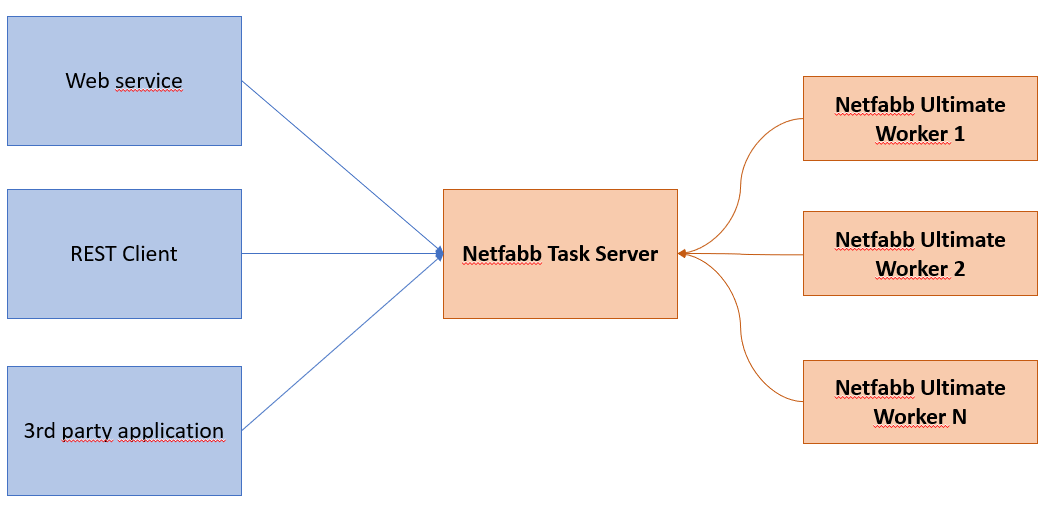
# Overview and Architecture

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.



## 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.



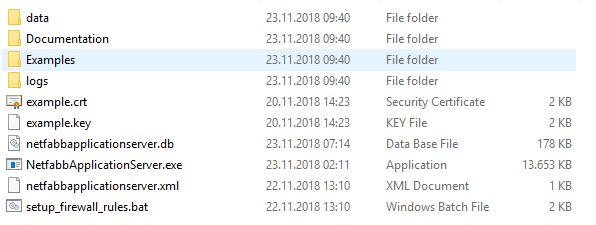
The current implementation 1.0.0 (Protocol Version 2.0.0) is a minimal example implementation and will get further refinements over several releases. Currently out of scope, but open roadmap items are:

* 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.

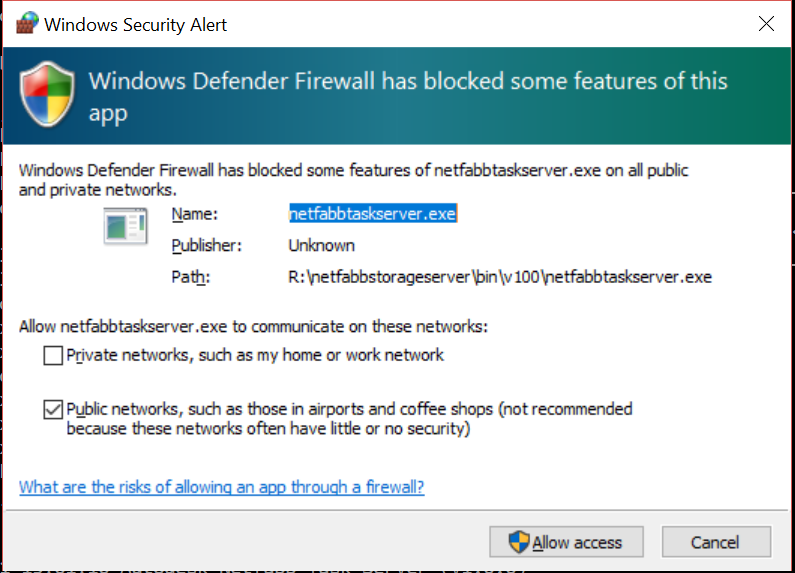
A simple message parameter protocol allows to propagate errors and success messages back to the client.

# Installation

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



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 |
| server host | the servers IP address (default: 127.0.0.1) |
| port | TCP port (default 8650) |
| log prefix | a prefix for the log files and path |
| data directory | a directory for all data files |
| database type | the database type (currently only SQLLite) |
| filename | the DB file |
| https type | Enabling of SSL: “http type=’tls’” – otherwise leave the field empty |
| certificate | The path to your certificate |
| privatekey | The path to your key file |
| authentication type |  |
| sessionduration | Duration until session timeout |
| global passphrase | A global shared secret |
| salt | If the salt is empty, the password in transferred in clear, otherwise a sha1 sum, which is calculated by (sha1 (sha1(salt)+cleartext-password)) |
| nameduser | 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 come with a small “PasswordSalter.exe” utility program. This program is a easy to use command line program with the following parameters:

|  |  |
| --- | --- |
| Parameter | Explanation |
| --help | Shows this help |
| -p password | Password to be salted |
| -s salt | 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:

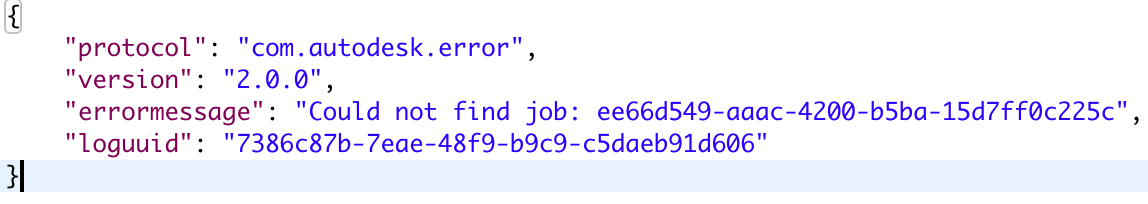
* **protocol**: Protocol schema string. Identifier for the request protocol type. Currently all protocols are of the form “com.autodesk.netfabbtasks.\*”
* **version:** Protocol version string. This is a protocol version string that is currently unique across all requests and responses that a application 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**:

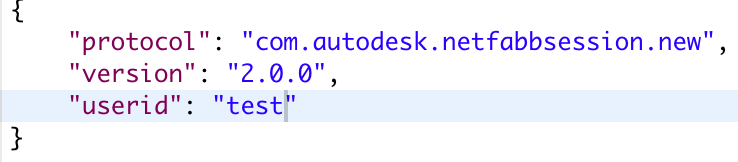


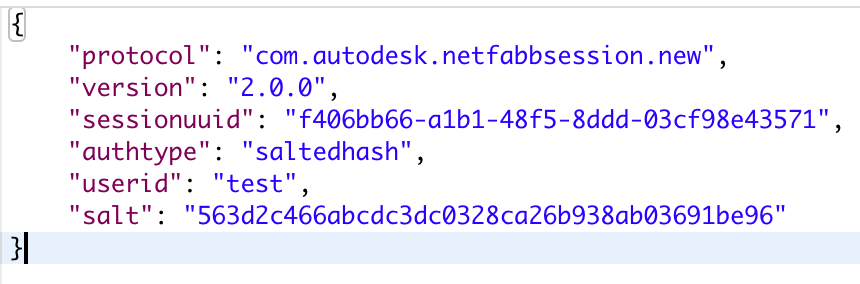
## New Session Request

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 | * **userid**: userid set in the config of the server |
| Return values | * **sessionuuid**: Unique identifier string for identifying the session the worker is working on |

**Example request**:

 **Example response**:

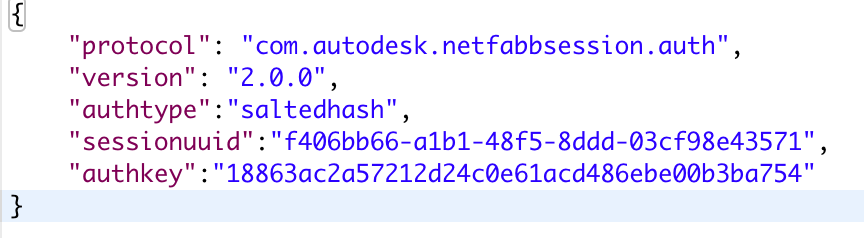


## Session authenticate Request

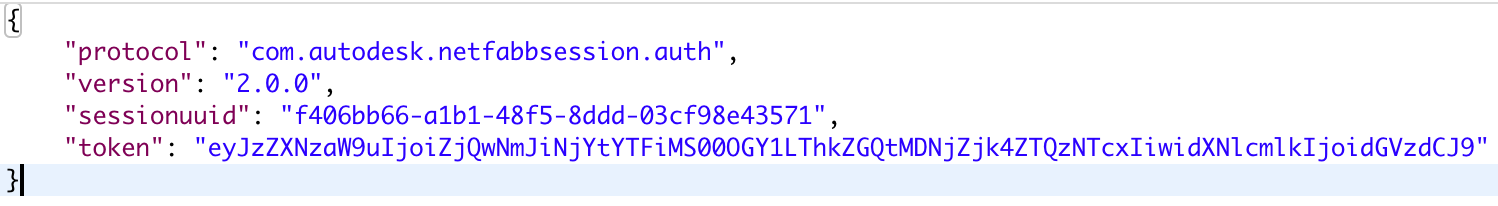
In order to use the new session created it has to been authenticated via the “session/auth” endpoint.

|  |  |
| --- | --- |
| Endpoint | /session/auth |
| Method | POST |
| Protocol Schema | com.autodesk.netfabbsession.auth |
| Body definition | * **authtype**: Has to be “saltedhash” * **sessionuuid**: The session uuid retrieved with the new session request * **authkey**: 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 |
| Return values | * **token**: Session token to be used with further requests |

**Example request**:



**Example response**:

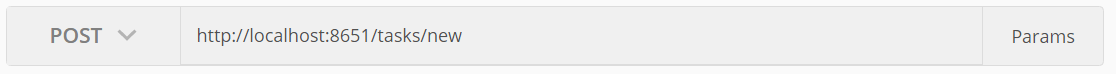


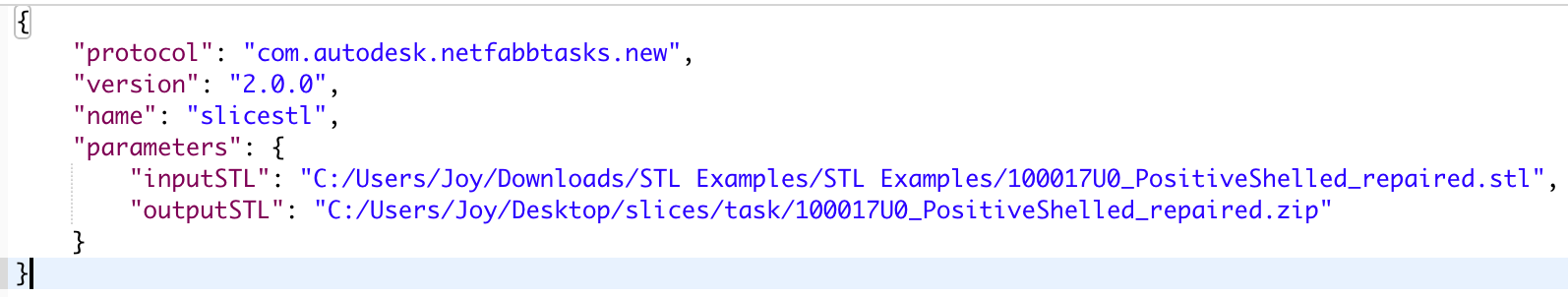
## New Task Request

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

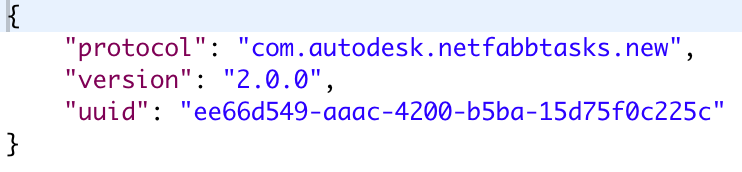
|  |  |
| --- | --- |
| Endpoint | /tasks/new |
| Method | POST |
| Protocol Schema | com.autodesk.netfabbtasks.new |
| Authentication | Bearer sessiontoken |
| Body definition | * **name**: Task identifier string. The Workers will poll for this name to run the correct worker script. * **parameters**: Key value pairs of arbitrary string parameters that determine the payload of the task description. |
| Return values | * **uuid**: Unique identifier string for identifying the task and polling the result information. |

**Example request**:





**Example response**:

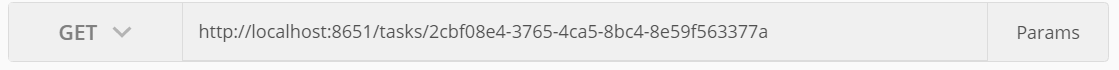


## Get Task Status Request

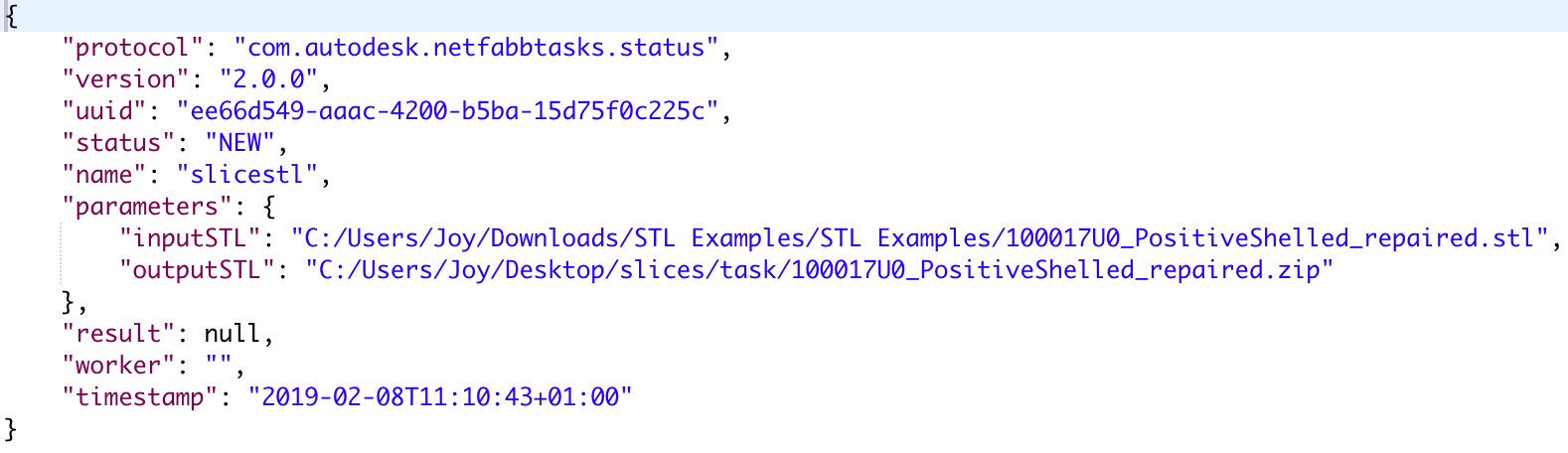
Retrieving the task status needs a GET request to the corresponding “task status” end point.

|  |  |
| --- | --- |
| Endpoint | /tasks/<uuid> |
| Method | GET |
| Protocol Schema | com.autodesk.netfabbtasks.status |
| Authentication | Bearer sessiontoken |
| Body definition | n/a |
| Return values | * **uuid**: Unique identifier string for identifying the task and polling the result information. * **status**: Current task status. Valid values are “NEW”, “INPROCESS”, “SUCCESS”, “ERROR”, “CANCELED”, “RETURNED” * **name**: Task identifier string. The Workers will poll for this name to run the correct worker script. * **parameters**: Key/Value pairs of arbitrary string parameters that determine the payload of the task description. * **results**: Key/Value pairs of arbitrary string results that give output information of the processes. * **worker**: Name of the worker instance that has processed the task * **timestamp:** Unix timestamp when the task was created. |

**Example request**:



**Example response**:



## Authentication and task creation example using Postman

1. Create a session:



The sessionuuid is to be used in the next request.

2. Authenticate the session



3. Create the Task



## 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