Hadouken documentation

Release 5.1.0

Viktor Elofsson and contributors

September 16, 2015

Contents

CHAPTER 1

Overview

Hadouken is a cross-platform headless BitTorrent client. It runs as a Linux daemon/Windows Service and provides a JSONRPC API over HTTP to control it. In addition to running headless, Hadouken also has a powerful extension system that gives developers freedom to extend it in various ways.

Hadouken runs on the following operating systems,

- Windows 7, 8, 8.1 and 10
- Windows Server 2008 R2, 2012, 2012 R2
- Debian 7.8
- Ubuntu 14.04 LTS

Furthermore, it also runs on the following devices,

- Raspberry Pi 2 Model B

Note: In case you find errors in this documentation you can help by sending pull requests!

CHAPTER 2

Features

- A powerful embedded web interface.
- Highly configurable, a single JSON text file to configure all aspects of Hadouken.
- Low memory footprint making it ideal for low-powered devices such as the Raspberry Pi.
- JSONRPC API over HTTP giving third-party developers complete freedom to integrate Hadouken with any kind of system.
- Automatically monitor directories for torrent files and add them based on regular expression matching, giving powerful abilities for sorting and tagging torrents.
- Advanced RSS feed monitoring capabilities making subscribing to various feeds a breeze.
- Move completed torrents matching specific regular expressions or having the correct set of tags.
- Send push notifications to your devices via Pushbullet or Pushover.
- Launch executables on various events, such as when a torrent finishes.
- A powerful JavaScript API making it easy to customize and extend Hadouken with plugins.
- Unattended installations to give domain administrators the ability to set up Hadouken clusters with ease.

CHAPTER 3

Downloads

Hadouken can be downloaded from the release feed. Binaries are provided for Windows. For other platforms you need to build it yourself.

CHAPTER 4

Installation

Installation instructions will vary depending on your platform. See the documentation for your specific platform.

4.1 Installing Hadouken on Windows

4.1.1 Overview

This will guide you through the installation of Hadouken on Windows.

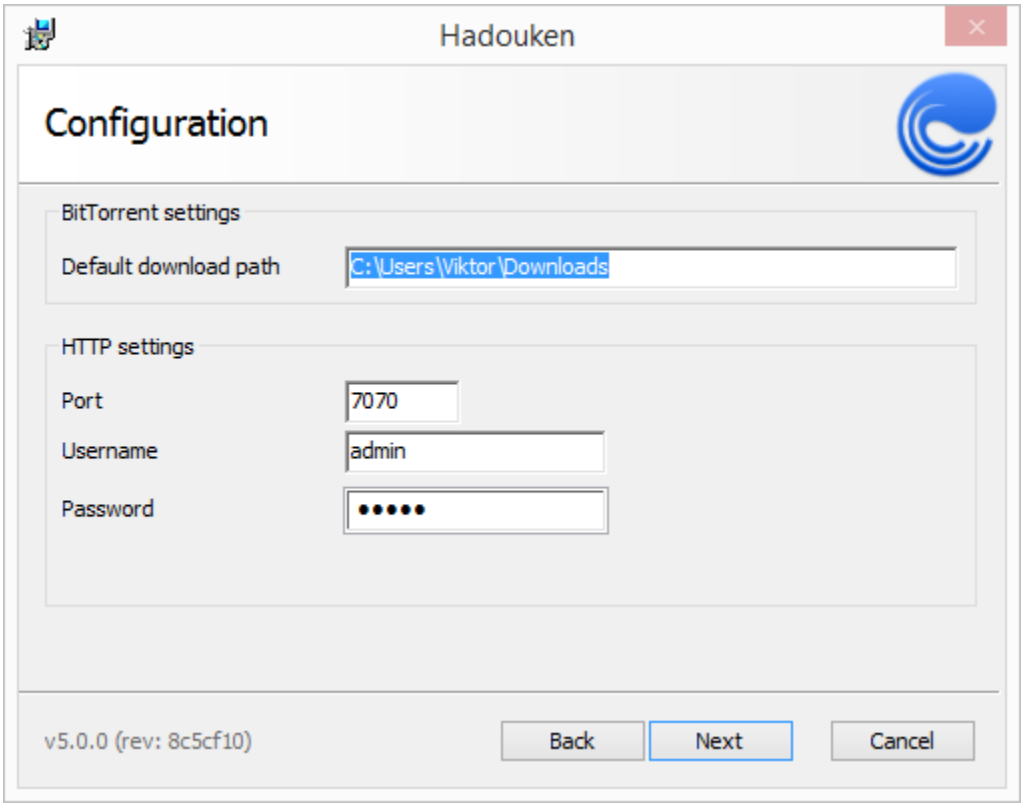
Hadouken runs as a native Windows Service and can be managed through the embedded web interface which is bundled with the installer. The web interface is accessible on *http://localhost:7070/gui/index.html* after installation (adjust the port accordingly).

Getting the installer

We provide MSI (Windows Installer) files for Hadouken which supports both attended and unattended installs. The latest installer can be downloaded from our GitHub release feed.

4.1.2 Installing

Follow the installation procedure and configure the basic settings as you wish. The default username/password is *admin*/*admin* and it is recommended you change this.



4.1.3 Unattended installation

Like most well-written MSI packages, Hadouken supports silent/unattended installation. This is mostly useful in an organization where Hadouken is deployed to clients through GPO.

These are the parameters you can set,

- `INSTALLDIR` - the installation directory. Defaults to `C:/Program Files (x86)/Hadouken`.
- `HDKN_BITTORRENT_SAVE_PATH_REPLACED` - the default save path for torrents. Defaults to `%USERPROFILE%/Downloads`.
- `HDKN_DATA_PATH` - the path where Hadouken stores its state data, i.e torrents and session state. Defaults to `C:/ProgramData/Hadouken`.
- `HDKN_HTTP_AUTH` - sets which type of HTTP authentication Hadouken will use. Valid values are *none*, *basic* and *token*. Defaults to *none*.
- `HDKN_HTTP_PORT` - the HTTP port where the API and event stream will listen. Defaults to *7070*.
- `HDKN_HTTP_TOKEN` - the token to use if *Token* authentication is set. Default value is an empty string.
- `HDKN_HTTP_BASIC_USERNAME` - the username to use when *Basic* authentication is set. Default value is an empty string.
- `HDKN_HTTP_BASIC_PASSWORD` - the password to use when *Basic* authentication is set. Default value is an empty string.

CHAPTER 5

Building Hadouken

5.1 Building Hadouken on Ubuntu

5.1.1 Overview

This will guide you through the process of building Hadouken on Ubuntu. The depenencies *libtorrent* and *cpp-netlib* exists as Git submodules.

5.1.2 What you need

In order to successfully clone and build Hadouken you need the following applications and libraries installed.

- `g++-4.9`
- `cmake`
- `git`
- `libssl-dev`
- `libboost-1.58`

Note: Hadouken will not compile with a *GCC* version lower than 4.9 since that is the version which shipped with C++14 support.

5.1.3 Cloning the repository

Clone the Hadouken GitHub repository.

```
$ git clone https://github.com/hadouken/hadouken
$ cd hadouken
$ git submodule update --init
```

5.1.4 Running the build

By now you should have all you need to build Hadouken.

```
$ ./linux/build.sh
```

5.2 Building Hadouken on Windows

5.2.1 Overview

This will guide you through the process of building Hadouken on Windows using MSVC 12 (Visual Studio 2013). Most of the dependencies are pre-built and will be pulled in during the build process, however *libtorrent* and *cpp-netlib* are placed in-source via Git submodules.

5.2.2 Prerequisites

In order to successfully clone and build Hadouken you need the following applications installed,

- MSVC 12.0 (Visual Studio 2013)
- Git
- CMake (>= 2.8)

5.2.3 Cloning the repository

Clone the Hadouken GitHub repository and initialize the submodules.

```
C:\Code> git clone https://github.com/hadouken/hadouken
C:\Code> cd hadouken
C:\Code\hadouken> git submodule update --init
```

5.2.4 Running the build

By now you should have all you need to build Hadouken. Other dependencies such as OpenSSL and Boost will be automatically downloaded in the build process.

```
C:\Code\hadouken> .\win32\build.ps1
```

CHAPTER 6

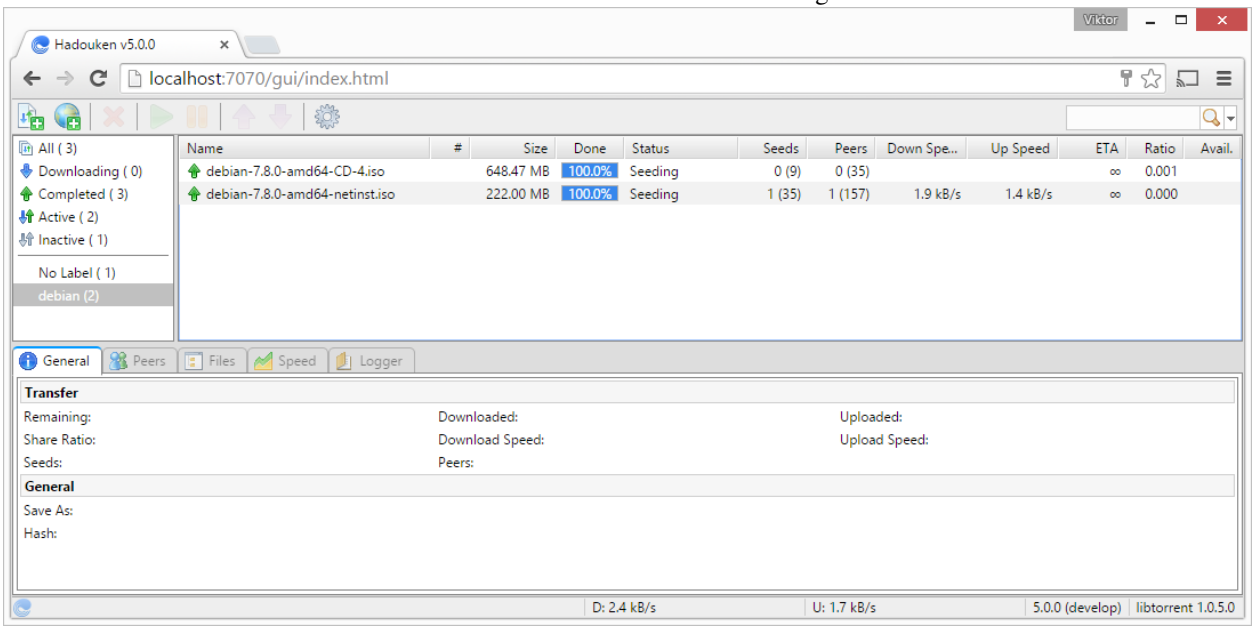
Documentation

6.1 Getting started

6.1.1 Overview

After running the installer, Hadouken can be accessed by pointing your browser to http://localhost:7070/gui/index.html and entering the username/password you specified during installation.

The default credentials are *admin*/*admin* however it is recommended to change this.



6.2 Migrating from v4

6.2.1 Overview

This will guide you through migrating your torrents from a previous version of Hadouken.

Note: Plugin settings cannot be migrated at this point.

6.2.2 Moving torrents

Make sure you have started Hadouken v5 at least once, then copy the files (both *.torrent* and *.resume*) from `C:/ProgramData/Hadouken/Torrents` to `C:/ProgramData/Hadouken/state/torrents`.

Restart the Hadouken service and your torrents should be available.

6.3 Configuring Hadouken

6.3.1 Overview

Hadouken is configured by editing the `hadouken.json` file. JSON is a simple structured format and is easily hand-edited using your favourite text editor.

See the list below for where your configuration file is.

- Windows (installed): `C:/ProgramData/Hadouken/hadouken.json`
- Windows (command line): `%CWD%/hadouken.json`

Note: Before making changes, stop Hadouken. Otherwise, Hadouken will overwrite your changes.

Warning: The configuration examples below only shows the JSON you need to change in order for the setting to have effect. Hadouken *will* fail to start if `hadouken.json` contains invalid JSON.

In depth

Configuring AutoAdd

Overview The AutoAdd extension monitors a list of directories and automatically adds any torrent files it finds. You can provide a regular expression (ECMAScript syntax) as well as specify save path and tags for the torrent it matches.

Enabling AutoAdd

```
{
  "extensions":
  {
    "autoadd":
    {
      "enabled": true
    }
  }
}
```

Monitoring a folder This shows the simplest configuration needed to monitor a folder. The default pattern will add any files with a `.torrent` extension. If you do not specify a save path, the default save path will be used.

```
{
  "extensions":
  {
    "autoadd":
    {
```

```
      "enabled": true,
      "folders":
      [
        { "path": "C:/Torrents" }
      ]
    }
  }
}
```

Matching with patterns One of the more powerful features of AutoAdd is the ability to match torrent files with a regular expression. You can, for example, match any Debian files and save them to their own folder. The pattern below will match all file names starting with *debian-* and ending with a *.torrent* extension, and save them to `C:/Downloads/Debian ISOs`.

```
{
  "extensions":
  {
    "autoadd":
    {
      "enabled": true,
      "folders":
      [
        {
          "path":     "C:/Torrents",
          "pattern":  "^debian-.*\\.torrent$",
          "savePath": "C:/Downloads/Debian ISOs"
        }
      ]
    }
  }
}
```

Adding tags The AutoAdd extension also supports tagging torrents. Tags provide a simple categorization system for torrents, and other extensions can use tags to apply their own set of rules. The configuration below will monitor a folder and add two tags to each torrent it finds.

```
{
  "extensions":
  {
    "autoadd":
    {
      "enabled": true,
      "folders":
      [
        {
          "path": "C:/Torrents",
          "tags": [ "tag1", "tag2" ]
        }
      ]
    }
  }
}
```

Configuring AutoMove

Overview The AutoMove extension can be used to move finished torrents to specific directories depending on basic rules, such as matching the torrent name against a regular expression (ECMAScript syntax) or checking that a torrent has a specific set of tags.

Enabling AutoMove

```
{
  "extensions":
  {
    "automove":
    {
      "enabled": true
    }
  }
}
```

Pattern-based moving The pattern filter matches a torrent name against a regular expression pattern and moves the torrent to the path specified in *path*. We specify to use the filter *pattern* and provide the filter with some data. In this case, the filter expects a pattern and a field.

```
{
  "extensions":
  {
    "automove":
    {
      "enabled": true,
      "rules":
      [
        {
          "path": "C:/Downloads/Debian ISOs",
          "filter": "pattern",
          "data":
          {
            "pattern": "^debian-.*",
            "field": "name"
          }
        }
      ]
    }
  }
}
```

Tag-based moving The tag filter will check a torrent against a list of tags. If the torrent has all the tags, it will move it to the path specified in *path*. In the example below, only torrents which have been tagged with both *debian* and *iso* will be matched and moved.

This can be used together with the AutoAdd extension to set up advanced matching and moving rules.

```
{
  "extensions":
  {
    "automove":
    {
      "enabled": true,
      "rules":
      [
```

```
        {
          "path": "C:/Downloads/Debian ISOs",
          "filter": "tags",
          "data": [ "debian", "iso" ]
        }
      ]
    }
  }
}
```

Configuring Launcher

Overview The Launcher extension provides an easy way to launch executables on various Hadouken events. It can be used to start command-line tools that provide some service, for example to unpack RAR'ed torrents when they finish.

> **Warning:** Executables launched will run with the same privileges as the user which runs Hadouken.

Enabling Launcher

```
{
  "extensions":
  {
    "launcher":
    {
      "enabled": true
    }
  }
}
```

Launching an executable The executable you launch will recieve three arguments passed to its command line. These are, in order,

- The hex-encoded info hash.
- The torrents name.
- The save path.

Each entry in the *apps* array is another array with two fields. The first field is a string with the *event name*, and the second is a string with a path to the application to launch.

The example below will launch `C:/Apps/some-app.bat` every time a torrent is added.

```
{
  "extensions":
  {
    "launcher":
    {
      "enabled": true,
      "apps":
      [
        [ "torrent.added", "C:/Apps/some-app.bat" ]
      ]
    }
```

```
  }
}
```

Available events

- *torrent.added*
- *torrent.finished*

Configuring Pushbullet

Overview The Pushbullet extension integrates with Pushbullet to provide push notifications for various platforms. You can configure which events will send push notifications.

Enabling Pushbullet

```
{
  "extensions":
  {
    "pushbullet":
    {
      "enabled": true
    }
  }
}
```

Configuration To use the extension you have to register at Pushbullet. This will get you a token which you put in the configuration. The configuration below will push notifications when Hadouken loads, any time a torrent is added and any time a torrent finishes downloading.

```
{
  "extensions":
  {
    "pushbullet":
    {
      "enabled": true,
      "token": "YOUR-PUSHBULLET-AUTH-TOKEN",
      "enabledEvents":
      [
        "hadouken.loaded",
        "torrent.added",
        "torrent.finished"
      ]
    }
  }
}
```

Available events

- *hadouken.loaded*
- *torrent.added*
- *torrent.finished*

Configuring Pushover

Overview The Pushover extension integrates with Pushover to provide push notifications for various platforms. You can configure which events will send push notifications.

Enabling Pushover

```
{
  "extensions":
  {
    "pushover":
    {
      "enabled": true
    }
  }
}
```

Configuration To use the extension you have to register at Pushover. This will give you a *user key*. You also need to register an application to get an *API token*.

```
{
  "extensions":
  {
    "pushover":
    {
      "enabled": true,
      "user":  "YOUR-PUSHOVER-USER-KEY"
      "token": "YOUR-PUSHOVER-API-TOKEN",
      "enabledEvents":
      [
        "hadouken.loaded",
        "torrent.added",
        "torrent.finished"
      ]
    }
  }
}
```

Available events

- *hadouken.loaded*
- *torrent.added*
- *torrent.finished*

Configuring RSS

Overview Hadouken can monitor and download torrents from various RSS feeds and also filter any items against regular expressions.

Monitoring a feed The simplest RSS configuration will monitor and download all items from a specified feed.

```
{
  "feeds":
  [
    "url":    "http://some-rss.net/feed",
    "filter": "*"
  ]
}
```

RegExp filters Each feed can be configured with a regular expression (ECMAScript syntax) to include and exclude any items. The following filter will include items with *720p* in the name, and exclude *NUKED* items.

The exclude filter is optional.

```
{
  "feeds":
  [
    "url":    "http://some-rss.net/feed",
    "filter": [ "regex", "720p", "NUKED" ]
  ]
}
```

Feed options There are a few options you can configure for each feed, such as save path and TTL (poll rate).

Save path

```
{
  "feeds":
  [
    "url": "http://some-rss.net/feed",
    "options":
    {
      "savePath": "C:/Downloads/from-some-rss"
    }
  ]
}
```

TTL Use this option with care - it is not nice to hammer feeds just to get an early start. The *ttl* value indicates the poll rate in minutes.

```
{
  "feeds":
  [
    "url": "http://some-rss.net/feed",
    "ttl": 2
  ]
}
```

Dry-run The dry-run option will stop any torrents from getting added to Hadouken and instead output information about the item in the log file. This can be used to debug regular expression filters.

```
{
  "feeds":
  [
    "url": "http://some-rss.net/feed",
```

```
    "options":
    {
      "dryRun": true
    }
  ]
}
```

6.3.2 BitTorrent configuration

Port

By default, Hadouken will use port *6881* for BitTorrent communications.

```
{
  "bittorrent":
  {
    "listenPort": 6881
  }
}
```

Activating GeoIP location

GeoIP is activated by downloading and extracting the MaxMind GeoLite Country database.

```
{
  "bittorrent":
  {
    "geoIpFile": "C:/Data/GeoIP.dat"
  }
}
```

Anonymous mode

Activating anonymous mode will make Hadouken try to hide its identity to a certain degree. The peer ID will no longer include the fingerprint, the user agent when announcing to trackers will be an empty string. It will also try to not leak other identifying information, such as local listen ports, your IP, etc.

Note: Activating anonymous mode may have an impact on your ability to connect to private trackers, which uses the peer ID and user agent to identify white-listed clients.

```
{
  "bittorrent":
  {
    "anonymousMode": true
  }
}
```

Disabling DHT

```
{
  "bittorrent":
  {
    "dht":
    {
      "enabled": false
    }
  }
}
```

Note: The *routers* are ignored if DHT is disabled.

Seed goals

Each torrent in Hadouken can be paused or removed when it reaches the user-specified seed goals. If no default options are specified, the seed goal is set to *2.0* however no action is configured.

To pause torrents when they have been seeded 200% or for 5 hours (18 000 seconds), use the configuration below.

Note: Only torrents added after the configuration change will get the new default options. Each torrent remembers its own options.

```
{
  "bittorrent":
  {
    "defaultOptions":
    {
      "seedRatio": 2.0,
      "seedTime": 18000,
      "seedAction": "pause"
    }
  }
}
```

Available actions are,

- *pause*
- *remove*

Storage allocation

There are two modes in which files can be allocated on disk, *full allocation* or *sparse allocation*. Sparse allocation is the recommended, and default, setting.

- In sparse allocation mode, sparse files are used, and pieces are downloaded directly where they belong.
- In full allocation mode, the entire file is filled with zeros before anything is downloaded. The files are allocated on demand, the first time anything is written to them. This avoids heavily fragmented files.

By setting the *sparse* flag to false, Hadouken will use full allocation. Changing this setting will only affect new torrents.

```
{
  "bittorrent":
  {
```

```
    "storage":
    {
      "sparse": false
    }
  }
}
```

6.3.3 HTTP configuration

Authentication

To configure your username and password, the keys *http.auth.basic.userName* and *http.auth.basic.password* are used.

```
{
  "http":
  {
    "auth":
    {
      "basic":
      {
        "userName": "YOUR-USERNAME",
        "password": "YOUR-PASSWORD"
      }
    }
  }
}
```

Changing port

By default, the HTTP server will listen on port *7070*. This can be changed from the installer or the configuration file. The example below will change the listen port to *8880*.

```
{
  "http":
  {
    "port": 8880
  }
}
```

Enabling HTTPS

By default, HTTPS is disabled. However, enabling it is as easy as generating a private key file and adding the required configuration.

```
{
  "http":
  {
    "ssl":
    {
      "enabled": true,
      "privateKeyFile": "C:/Keys/my-private-key.pem",
      "privateKeyPassword": "my-password"
    }
```

```
    }
}
```

Note: Using a private key which is not trusted by the client computer may generate warnings and errors in the client browser. To avoid problems, add the public key to your client system.

Custom root path

To support advanced proxy scenarios, Hadouken supports customization of the root path for the HTTP server. The default behavior is to serve requests from the root */*.

The example below will change this to let you serve requests from */hadouken*, which means you will reach the API at */hadouken/api* and the GUI at */hadouken/gui*.

```
{
  "http":
  {
    "root": "/hadouken"
  }
}
```

6.4 For developers

Hadouken has both a JSONRPC service and a JavaScript API. You can use the JSONRPC service to create third-party software that integrates with Hadouken in various ways - this is what the official remote client does.

The JavaScript API is used for extending Hadouken directly. You have access to all parts of Hadouken and can tailor it to your needs.

6.4.1 Documentation

The JavaScript API

Hadouken features a JavaScript API which exposes most of the functions you'd need for writing plugins.

Writing plugins

Basic plugin skeleton

Overview The most basic plugin for Hadouken does nothing. It is a simple `.js` file that exports a load function.

The plugin directory Depending on whether Hadouken is installed or running portable, the JavaScript root varies.

- If Hadouken is installed, plugins are (by default) loaded from `%PROGRAMDATA%/Hadouken/js/plugins`.
- If Hadouken is running portable, plugins are (by default) loaded from `js/plugins` which is relative to where the `hadouken.exe` resides.

Basic plugin

```
exports.load = function() {};
```

Reacting to events

Overview This example will show you how to react to the *torrent.finished* event and write the info hash to a file when it finishes.

Example code

```
var fs      = require("fs");
var session = require("bittorrent").session;

// File to write torrent info hashes to.
var file = "C:/Temp/finished-torrents.json";

function torrentFinished(args) {
  var contents = [];

  if(fs.fileExists(file)) {
    contents = JSON.parse(fs.readText(file));
  }

  // Add our info hash to the array.
  contents.push(args.torrent.infoHash);

  // Write the results to the file.
  fs.writeText(file, JSON.stringify(contents));
}

exports.load = function() {
  session.on("torrent.finished" torrentFinished);
};
```

JavaScript reference

Modules

The BitTorrent module

Overview The `bittorrent` module contains all BitTorrent related functions and classes for Hadouken.

```
var bt = require("bittorrent");
```

Properties

`Session` session Gets the BitTorrent session instance. Use the session to add, remove and find torrents.

```
var session = require("bittorrent").session;
```

The JSONRPC API

JSONRPC is a standard defined at jsonrpc.org. It is a simple RPC protocol and most languages has at least a way to encode/decode JSON data and send it over HTTP, which makes it a safe bet for a remote API.

The JSONRPC API can be accessed at `http://localhost:7070/api`. Adjust accordingly.

Note: All examples are shown with only a method name and request/response data - the JSONRPC object container has been left out.

Methods

core.getSystemInfo

Overview Gets an object with information about this Hadouken instance. It contains the Git commitish and branch, libtorrent version and Hadouken version.

Example

```
{
  "method": "core.getSystemInfo",
  "params": []
}
```

Returns,

```
{
  "commitish": "e51736c",
  "branch": "develop",
  "versions":
  {
    "libtorrent": "1.0.5.0",
    "hadouken": "5.0.0"
  }
}
```

webui.addTorrent

Overview Adds a torrent to the session, with the specified save path, label, tags and trackers. This method can add both a file and URL. Files must be base64 encoded.

The method takes three arguments, *type*, *data* and *params*.

- *type* - the source type. Can be *file* or *url*.
- *data* - the base64 encoded torrent file, or a URL to a torrent file.
- *params* - an object with properties describing the torrent to add, eg. save path, tags, label etc.
  - *label*
  - *filePriorities* - if you want to pre-set the priorities for files in this torrent, set this array to their specific priorities.

- *savePath* - a zero-based index specifying which save path to use. Index 0 represents the default save path, and indices higher than one represents entries in the *bittorrent.downloadDirectories* array (savePath=1 is the first entry in that array).
- *subPath* - a sub-directory of the save path.
- *tags* - a string array of tags for this torrent.
- *trackers* - a string array of extra trackers for this torrent.

Example *Add a base64 encoded torrent file*,

```
{
  "method": "webui.addTorrent",
  "params": [
    "file",
    "<base64 encoded data>",
    {
      "label": "software",
      "savePath": 0,
      "subPath": "linux isos",
      "tags": [ "debian", "linux", "oss" ]
    }
  ]
}
```

Returns,

*If you add a URL, the info hash is not known and you will receive *undefined* instead of an info hash.*

```
"<infoHash>"
```

webui.getFiles

Overview Returns the files associated with the torrent identified by the info hash given in the first parameter. The *files* array always has n*2 items where *n* is the info hash and *n+1* is the files array.

Example

```
{
  "method": "webui.getFiles",
  "params": [ "<info hash>" ]
}
```

Returns,

```
{
  "files": [
    "<info hash>",
    [
      "file.iso", // path to file relative to the save path of the torrent
      100, // file size in bytes
      80, // downloaded bytes
      1, // priority
      -1, //first piece
      -1, // num piece
      -1, // streamable
      -1, // encoded rate
```

```
      -1, // duration
      -1, // width
      -1, // height
      -1, // stream eta
      -1  // streamability
    ]
  ]
}
```

webui.getPeers

Overview Returns the peers associated with the torrent identified by the info hash given in the first parameter. The *peers* array always has n*2 items where *n* is the info hash and *n+1* is the peers array.

Example

```
{
  "method": "webui.getPeers",
  "params": [ "<info hash>" ]
}
```

Returns,

```
{
  "peers": [
    "<info hash>",
    [
      "SE", // two-letter ISO3166 country code
      "127.0.0.1", // ip address
      "", // reverse dns entry
      true, // utp
      6881, // port
      "libTorrent", // client name
      "HPL", // peer flags (based on uTorrent)
      870, // progress (divide by 1000 to get percentage)
      10, // download rate
      14, // upload rate
      3, // incoming requests in queue
      7, // outgoing requests in queue
      -1, // waited
      56, // uploaded bytes
      13, // downloaded bytes
      0, // hash fail count
      -1, // peer download
      -1, // max up
      -1, // max down
      -1, // queued
      100, // last active
      -1 // relevance
    ]
  ]
}
```