SERVER

Table of Contents

Overview

Introduction

µTorrent Server is designed for use on computers running Linux and other UNIX-like operating systems. It provides a state-of-the-art implementation of the BitTorrent protocol and a full-featured web-based user interface in a small footprint.

Features

µTorrent Server is a full implementation of the official BitTorrent protocol. Features include:

- Distributed hash table (DHT)
- UPnP port mapping
- NAT-PMP port mapping
- Upload rate limiting
- Download rate limiting
- Queuing
- Configurable limit on number of simultaneously uploading peers
- Incremental file allocation
- Block level piece picking
- Separate threads for file-check and download
- Single thread and single port for multiple torrent downloads
- BitTorrent extension protocol
- Multi-tracker extension support
- Fair trade extension
- Compact tracker extension

- Fast resume
- Queuing of torrent file-check if fast resume not possible
- HTTP seed support
- Resumption of partial downloads from other BitTorrent clients
- File-sizes greater than 2GB
- Selective download of multi-file torrents
- IPv6
- High performance network stack
- uTP - Advanced UDP-based transport with dynamic congestion control

Additionally, µTorrent Server includes a full-featured web-based user interface.

Getting Started

µTorrent Server consist of an executable (`utserver`) that implements BitTorrent services and is controlled through an HTTP-based application programming interface (API).

Command-line Arguments

µTorrent Server supports the following arguments - the keywords are case-insensitive and should be immediately preceded with a dash (`-`):

- `configfile` - path to and name of the configuration file - if this argument is not specified, the `utserver` program looks for a file named `utserver.conf` in the current working directory - see the following section for more details
- `logfile` - path to and name of the log file - specifying `-logfile filename` will direct log output to `filename`, while specifying `-logfile` without a filename will direct log file output to `utserver.log` in the current working directory

- `settingspath` - path to directory to store files of relatively small size (operational settings, torrent resume information, RSS feed information) - if not specified, the `utserver` program will store these files in the current working directory
- `pidfile` - path to and name of the file to create that will contain the process ID of the `utserver` process
- `daemon` - directs the server to run in its own process group
- `usage` - directs the server to generate message describing supported arguments and then exit

Configuration File

At startup the `utserver` process looks for a configuration file which allows the behavior of the product to be customized by changing the values of certain settings. The format of this file is as follows:

- each setting is on a separate line;
- each line consists of colon-separated name of the setting and its value;
- any line whose first non-whitespace character is # is a comment.

For example, a file that sets two values and includes one comment might look like:

```
# This is a comment
dir_temp_files: temp
preferred_interface: eth0
```

For a complete list of application settings, see Application Settings.

Environment

When the `utserver` process creates files intended for end users, such as torrent data files, it uses the value of the file mode creation mask to specify access to these

files. For more information, run `man umask` on a Linux command line. The value of the file mode creation mask is not considered when creating files not intended for use by users, such as settings files.

Application Settings

Settings fall into two categories:

- internal settings, whose values can only be set in the `utserver.conf` file;
- regular settings, whose values can be set in the `utserver.conf` file or the `/api/app-settings-set` RPC API call.

A setting can be of one of three types:

- *string*
- *integer*
- *Boolean* value (`1` for true and `0` for false)

Internal Settings

bind_ip (*string*): IP address to use for socket connections. If not provided, a default IP address will be used. We do not recommend changing this value.

ut_webui_port (*integer*): Default value: 8080. Port number where the `utserver` process accepts HTTP RPC API calls to support the µTorrent-compatible HTTP interface. If the `utserver` process also serves HTML files (see `webui_server_files` setting), also the port of HTTP server.

token_auth_enable (*boolean*) Default value: true. If true, the µTorrent HTTP interface defends against cross-site request forgeries by requiring that a short-lived token be obtained from the µTorrent HTTP interface and included at the beginning of the parameter list of any request made to that interface. If false, the µTorrent HTTP interface will not be protected in this manner.

dir_active (*string*): Default value: "./". Directory in which currently downloaded data is saved. Can be an absolute path or a relative path. If it is a relative path, the value is relative to `dir_root` or the current working directory if `dir_root` is not defined or an empty string. It is recommended that this directory be hidden from users (i.e. not exported through Samba).

dir_completed (*string*): Default value: "". Directory where completed downloads are stored. If the value is an empty string, the value of `dir_active` is used. This value must represent a path that is accessible to users (e.g. exported through Samba). It also has to be on the same volume as `dir_active`.

dir_download (*string*): Default value: "". Optional directory where completed downloads can be stored, instead of in `dir_completed`. If no value is specified for this setting, the value of `dir_completed` is used. The value must represent a path that is accessible to users (e.g. exported through Samba).

This option can be specified multiple times in the file - once for each directory to be designated as such. This option can be used when adding torrents via the µTorrent HTTP interface, not via the SDK interface.

Use the action `list-dirs` to obtain a list of download directories from the µTorrent HTTP interface. Use the option `download_dir` to specify which of these directories to use when adding a torrent by URL or file through the µTorrent HTTP interface; specify the one-based index of the entry of interest. The index of each entry will be in order in which each entry appears in `utserver.conf`, starting with 1 for the first entry, 2 for the second entry, and so on. 0 indicates the default download directory should be used.

dir_torrent_files (*string*): Default value: "". Directory where torrent files are stored. If the empty string, the value of `dir_active` is used. It is recommended that this directory be hidden from users (i.e. not exported through Samba).

dir_temp_files (*string*): Default value: "". Directory where temporary files are stored. If the empty string, the value of `dir_active` is used. It is recommended that this directory be hidden from users (i.e. not exported through Samba). Also, using a separate directory just for temporary files allows for deleting the files in this directory on boot and/or periodically. The `utserver` process creates temporary files with a `.utt` extension - if a value for this setting is specified, the `utserver` process will delete all files with that extension in that directory at process startup. This setting applies only to

POSIX systems. The value should specify a directory, not a symbolic link to a directory.

dir_autoload (*string*): Default value: “”. Directory where torrent files will be recognized and auto-loaded. If the empty string, auto-load is disabled.

dir_autoload_delete (*boolean*): Default value: false. If true, torrent files in the autoload directory will be deleted after being loaded, else they will be renamed with an extension of `.loaded`. The `dir_autoload` setting must be specified for this setting to have an effect.

dir_request (*string*): Default value: “”. Directory where maintenance request files will be recognized, loaded, and deleted. If the empty string, maintenance request handling is disabled. This directory should be hidden from users (i.e., not exported through Samba). Your software running on your device can create the following files in this directory in order to request the following maintenance procedures.

If the file `c.utmr` is created in or moved into this directory, the credentials necessary to access the µTorrent HTTP interface will be reset to username `admin` and a blank password.

If the file `wipl.utmr` is created in or moved into this directory, the IP restriction list that limits the IPs that can use the µTorrent HTTP interface is cleared, so that there will be no restrictions on IP address.

These maintenance operations provide a way to help a user who has either entered new credentials and then forgotten them, or who has entered an IP range in the restricted list and can no longer access the µTorrent HTTP interface as a result.

If the file `rcf.utmr` is created in or moved into this directory, the server will reload the configuration file. If you always use this method to request a configuration file reload, you can safely change the value of this setting while the server is running.

The server will also reload the configuration file if you send a hangup signal to the server; however, a race condition may occur if you send a hangup signal to the server in order to change the value of this setting. *You should either only use the file system interface for requesting configuration file reloads, or you should not change the value of this setting in the configuration file before sending a hangup signal to the server.*

upnp (*boolean*): Default value: true. If true, UPNP functionality for mapping ports is used by `utserver`. We recommend setting this value to true.

natpmp (*boolean*): If true, NAT-PMP functionality for mapping ports is used by `utserver`. Default value: true. We recommend setting this value to true.

lsd (*boolean*): Default value: true. If true, Local Service Discovery is enabled. We recommend setting this value to true.

dht (*boolean*): Default value: true. If true, Distributed Hash Table extension is enabled. We recommend setting this value to true.

pex (*boolean*): Default value: true. If true, Peer Exchange extension is enabled. We recommend setting this value to true.

rate_limit_local_peers (*boolean*): Default value: false. If true, rate limiting also applies to communications with peers in the local subnet. We recommend setting this value to false.

dir_root (*string*): Default value: "". If not empty, **dir_active**, **dir_completed**, and **dir_torrent_files** are relative to this directory.

disk_cache_max_size (*integer*): Default value: 0. Maximum amount of memory used by each of the read, write, and piece caches. Value is in megabytes. If 0, accepts the SDK's default choices on selecting sizes of disk caches. Maximum value is 512.

The value of this setting will be applied every time the `utserver` process starts.

preferred_interface (*string*): Default value: "". If defined, name of network interface to be preferred when attempting to search among network interfaces for an external IP and hardware address. If empty string, preferred interface is ignored.

You need to provide a value for this setting if either 1) the toolchain for your computer does not supply ifaddrs.h, or 2) you want the `utserver` process to choose a different interface than it would choose on its own. You should set a value for this setting if you see an incorrect port mapping on a UPnP router for the subnet to which the device belongs (the IP address of the device will not appear in the port mapping requested by the `utserver` process - instead, the IP address associated with the mapping will be 0.0.0.0 with a device having a toolchain that does not include ifaddrs.h, or some other IP address with a device having a toolchain that includes ipaddrs.h).

The value of this setting will be applied every time the `utserver` process starts.

admin_name (*string*): Default value: “admin”. If defined, name that must be supplied (along with the password) when authenticating to the server via the HTTP interface. This allows the administrator to define an initial non-default value for this name. This value will not be applied from `utserver.conf` if `settings.dat` already exists.

admin_password (*string*): Default value: “”. If defined, password that must be supplied (along with the name) when authenticating to the server via the HTTP interface. This allows the administrator to define an initial non-default value for this password. This value will not be applied from `utserver.conf` if `settings.dat` already exists.

localhost_authentication (*boolean*): Default value: true. If false, HTTP requests originating from the local host will not be required to include administrative user name and password. Disabling authentication of requests originating on the local host may be useful on computers where there are no other users or processes that shouldn’t be prevented from accessing the HTTP interface. *Disabling authentication presents a security risk on those computers that support untrusted users or allow third-party applications to be installed and run outside the control of the administrator.*

logmask (*integer*): Default value: 0. A mask whose bits when set allow certain categories of log messages to be generated. The value of this setting will be applied every time the `utserver` process starts.

The bits (0 - 31) in the value of this setting correspond to a set of internal events and subsystems. The usage of these bits may change without advance notice in a future release.

- 3 - send have
- 6 - hole punch
- 7 - got bad piece request
- 8 - trace
- 9 - piece picker
- 10 - got bad cancel
- 11 - got bad unchoke
- 12 - got bad piece
- 13 - rss
- 14 - rss error

- 15 - got have
- 16 - got bad have
- 17 - error
- 18 - aggregated
- 19 - disconnect
- 20 - out connect
- 21 - in connect
- 22 - UPnP
- 23 - UPnP error
- 24 - NATPMP
- 25 - NATPMP error
- 26 - metadata finish
- 27 - web UI
- 28 - got bad reject
- 29 - pex
- 30 - peer messages
- 31 - blocked connect

ut_webui_dir (*string*): Directory where the web UI file archive `webui.zip` is stored, or which contains a `webui` subdirectory within which the unarchived web UI files are stored. It can be an absolute path or set relative to the current directory. It is recommended that this directory be hidden from users (i.e. not exported through Samba). Default value: "" (to use the same directory as `settings.dat` and other settings files).

finish_cmd (*string*), state_cmd (*string*): If defined, `finish_cmd` is a command that will be executed upon completion of each torrent, and `state_cmd` is a command that will be executed when a torrent changes state. Default value: "" (no command is run for the event(s) associated with that setting).

The command is run asynchronously, so that a lengthy or hung process will not block the server. The server creates a new process group for the command, so that the server does not need to wait for it, and so the kernel process table does not fill up with zombie processes. The command is run as the same user that runs the server process.

The server permits substitutions in the command text as follows:

%F Name of downloaded data file (for single-file torrents)

%D directory where torrent data files are saved

%N torrent title

%S torrent state

%P previous state of torrent

%L label associated with torrent

%T tracker

%M status message

%I hex-encoded info-hash

State (`%S`) and previous state (`%P`) are integers that have the following values:

- 1 (error)
- 2 (checked)
- 3 (paused)
- 4 (super seeding)
- 5 (seeding)
- 6 (downloading)
- 7 (super seeding (forced))
- 8 (seeding (forced))
- 9 (downloading (forced))
- 10 (queued seed)
- 11 (finished)
- 12 (queued)
- 13 (stopped)

uconnect_enable (*boolean*): Default value: false. If true, the values of `uconnect_username` and `uconnect_password` are provided when authenticating to µTorrent Remote. This value is always applied from `btsettings.txt`; it is not saved in `settings.dat`.

uconnect_username (*string*): Default value: “”. If defined, name that must be supplied (along with the password) when authenticating to µTorrent Remote. This value is always applied from `btsettings.txt`; it is not saved in `settings.dat`.

uconnect_password (*string*): Default value: “”. If defined, password that must be supplied (along with the name) when authenticating to µTorrent Remote. This value is always applied from `btsettings.txt`; it is not saved in `settings.dat`.

low_cpu (*boolean*): Default value: false. If true, a short sleep occurs during the process of handling network traffic, so that network traffic handling presents less of a load on the CPU. This value will not be applied from `utserver.conf` if `settings.dat` already exists.

Regular Settings

bind_port (*integer*): Default value: 6881. Port used for BitTorrent protocol. This can be any value in the range 1025-65000.

max_ul_rate (*integer*): Default value: -1. Maximum total upload rate in kilobytes per second. -1 means unlimited. We recommend setting it to -1.

max_ul_rate_seed (*integer*): Default value: -1. Maximum per-torrent upload rate when seeding, in kilobytes per second. -1 means unlimited. We recommend setting it to -1.

conns_per_torrent (*integer*): Default value: 50. Maximum number of connections for a given torrent.

max_total_connections (*integer*): Default value: 200. Maximum number of connection opened at the same time.

auto_bandwidth_management (*boolean*): Default value: true. If true, upload bandwidth is automatically throttled in order to not impact other applications using TCP/IP.

max_dl_rate (*integer*): Default value: -1. Maximum total download rate in kilobytes per second. -1 means unlimited. We recommend setting it to -1.

seed_ratio (*integer*): Default value: 0. Seed ratio in percent (%) (e.g. 100 means 100%). If not 0, seeding will stop after reaching this upload/download ratio.

seed_time (*integer*): Default value: 0. Time after which seeding will stop, in seconds. 0 means seeding won’t stop.

Reloadable Settings

Many of these settings are only read from the configuration file when the µTorrent settings file `settings.dat` does not already exist in the settings directory. Once `settings.dat` exists, the values specified in the configuration file for these settings will be ignored, and the values stored in `settings.dat` will be used. For other settings, the server will load the values from the configuration file every time the program starts or receives a request to reload the configuration file.

The settings for which the values are always applied from the configuration file when the file is read by the server include:

- `dir_request`
- `disk_cache_max_size`
- `finish_cmd`
- `logmask`
- `preferred_interface`
- `state_cmd`
- `uconnect_enable`
- `uconnect_password`
- `uconnect_username`
- `ut_webui_dir`

µTorrent API

What is the µTorrent Web UI?

µTorrent Server is based on the same codebase as the BitTorrent Mainline and µTorrent PC applications, which have a powerful API to control and configure the client from both local and remote applications, as well as display most data

available. Normally, this functionality is used in a web browser with the reference Web UI we provide, but any application can talk to µTorrent directly by using the Web API.

What is the µTorrent Web API?

It is an API to access the functions of the WebUI built into the application. The API is stable and largely complete. Missing functionality will be added over time and compatibility with existing applications will generally be preserved, so little to no work will be needed to keep your web app working with future versions of the applications. Many community projects have extended the functionality of the PC applications using this Web API. These projects form a rich ecosystem and vibrant community. Partners should consider the relative merits of enabling this community to extend their platforms through the Web API. Current projects and discussion can be accessed in the Web API section of the developer forums at forum.utorrent.com.

General notes for API access

The base URI to access the API is:

`http://[IP]:[PORT]/gui/`. The data returned by calls is in the JSON format.

Authentication is done with basic HTTP authentication. The guest account, disabled by default, is limited to a subset of the calls (Action calls that modify torrent state and application and torrent settings are disallowed).

Unless otherwise noted, each request is made using HTTP GET. Parameters are added onto the base URI in the format that is standard for HTTP GET. The first parameter should always be the command (e.g. `?list` or `?action`).

Most action commands require a hash to be passed. This is the infohash of the torrent, obtained from listing all torrents. Each hash is a 40-character ASCII string. Some commands accept multiple infohashes chained together, e.g.“http://[IP]:[PORT]/gui/?action=[ACTION]&

HASH 1]&hash=[TORRENT HASH 2]&...“ to cut down on the number of requests required.

When setting boolean values either by &action=setsetting or &action=setprops, the value parameter should be sent as 0 for “false” or 1 for “true” rather than a string indicating “true” or “false”.

Token Authentication System (important!)

A token authentication system (http://trac.utorrent.com/trac/wiki/TokenSystem) was implemented in µTorrent to prevent cross-site request forgeries (CSRF). All developers creating applications that use the WebUI backend MUST implement support for this system, as the application will otherwise fail if the user has webui.token_auth enabled.

The `token=` parameter must appear on a request’s parameter list before or immediately following the `action` or `list` parameters, or the request will fail.

In 1.8.3 (µTorrent for windows) and earlier, token authentication is disabled by default (though users can enable it manually), but this option WILL be enable by default in future versions, so implementing support now is a requirement for future compatibility.

Modifying Settings

The base parameter for settings is `?action=getsettings`. Using this parameter by itself will give a list of settings in the following format:

```
{
 "build": BUILD NUMBER (integer),
 "settings": [
     [
         OPTION NAME (string),
         TYPE* (integer),
         VALUE (string),
         PARAMETERS {
```

```
                "access": (string)
            }
        ],
            ...
    ]
}
```

OPTION NAME is the name of the setting. They are not listed here, as some of the settings (particularly advanced ones) vary with each version and most are self-explanatory. However, a near-complete list for 1.8.2 is at http://forum.utorrent.com/viewtopic.php?id=55526 for your perusal.

TYPE: The TYPE is an integer value that indicates what type of data is enclosed within the VALUE string. The following is a list of the possible TYPEs and what VALUE type it corresponds to:

- 0 = Integer
- 1 = Boolean
- 2 = String

To change settings, the following URI can be used:

`http://[IP]:[PORT]/gui/?action=setsetting&s=[SETTING]&v=[VALUE]`

Multiple settings can be changed in a single request by chaining together `s=` and `v=` in the URI. `s=` defines the setting and `v=` is the value for the `s=` immediately preceding it.

For example, to set the global upload cap to 10KiB/s and the global download cap to 40KiB/s, you would request this URI:

`http://[IP]:[PORT]/gui/?action=setsetting&s=max_ul_rate&v=10&s=max_dl_rate`

PARAMETERS: The PARAMETERS is a dictionary of additional attributes of the associated setting. The only attribute currently supported is `access`, which specifies if the client can view and/or modify this setting. Here are the possible values:

- Y - read and write
- R - read only
- W - write only

Settings that are neither readable nor writable won't be included in the list of settings generated by the `getsettings` action.

Torrent/labels/RSS/Filters List

To get the list of all torrents and RSS feeds, request `http://[IP]:[PORT]/gui/?list=1` . This will return the torrents in the following fashion:

```
{
        "build": BUILD NUMBER (integer),
        "label": [
                [ LABEL (string),
                TORRENTS IN LABEL (integer) ],
                ...
        ],
        "torrents": [
                [ HASH (string),
                STATUS* (integer),
                NAME (string),
                SIZE (integer in bytes),
                PERCENT PROGRESS (integer in per mils),
                DOWNLOADED (integer in bytes),
                UPLOADED (integer in bytes),
                RATIO (integer in per mils),
                UPLOAD SPEED (integer in bytes per second),
                DOWNLOAD SPEED (integer in bytes per second),
                ETA (integer in seconds),
                LABEL (string),
                PEERS CONNECTED (integer),
                PEERS IN SWARM (integer),
                SEEDS CONNECTED (integer),
                SEEDS IN SWARM (integer),
```

```
                AVAILABILITY (integer in 1/65535ths),
                TORRENT QUEUE ORDER (integer),
                REMAINING (integer in bytes),
                DOWNLOAD URL (string),
                RSS FEED URL (string),
                STATUS MESSAGE (string),
                STREAM ID (string),
                ADDED ON (integer in seconds),
                COMPLETED ON (integer in seconds),
                APP UPDATE URL (string) ],

                ...
        ],
        "rssfeeds": [
                [ IDENT (integer),
                ENABLED (boolean),
                USE FEED TITLE (boolean),
                USER SELECTED (boolean),
                PROGRAMMED (boolean),
                DOWNLOAD STATE (integer),
                URL (string),
                NEXT UPDATE (integer in unix time),
                [
                        [ NAME (string),
                        NAME FULL (string),
                        URL (string),
                        QUALITY (integer),
                        CODEC (integer),
                        TIMESTAMP (integer),
                        SEASON (integer),
                        EPISODE (integer),
                        EPISODE TO (integer),
                        FEED ID (integer),
                        REPACK (boolean),
                        IN HISTORY (boolean) ],

                        ...
                ] ],

                ...
```

```
        ],
        "rssfilters": [
                [ IDENT (integer),
                FLAGS (integer),
                NAME (string),
                FILTER (string as regexp),
                NOT FILTER (string as regexp),
                DIRECTORY (string),
                FEED (integer as feed ID),
                QUALITY (integer in bytes),
                LABEL (string),
                POSTPONE MODE (integer),
                LAST MATCH (integer),
                SMART EP FILTER (integer),
                REPACK EP FILTER (integer),
                EPISODE FILTER STR (string),
                EPISODE FILTER (boolean),
                RESOLVING CANDIDATE (boolean) ],

                ...
        ],
        "torrentc": CACHE ID** (string integer)
}
```

STATUS: The STATUS is a bitfield represented as integers, which is obtained by adding up the different values for corresponding statuses:

- 1 = Started
- 2 = Checking
- 4 = Start after check
- 8 = Checked
- 16 = Error
- 32 = Paused
- 64 = Queued
- 128 = Loaded

For example, if a torrent job has a status of 201 = 128 + 64 + 8 + 1, then it is loaded, queued, checked, and started. A bitwise AND operator should be used to determine whether the given STATUS contains a particular status.

CACHE ID: The CACHE ID is a number randomly generated by µTorrent for the given data. By requesting the torrent list using `http://[IP]:[PORT]/gui/?list=1&cid=[CACHE ID]`, only the items that have changed since the list corresponding to the CACHE ID was sent will be returned. This is used to minimize bandwidth usage and simplify parsing by decreasing the amount of data sent by µTorrent. In this situation, six new dictionary keys replace `"torrents"`, `"rssfeeds"` and `"rssfilters"` and the returned JSON will look as follows:

```
{
        "build": BUILD NUMBER (integer),
        "label": [
                [ LABEL (string),
                TORRENTS IN LABEL (integer) ],

                ...
        ],
        "torrentp": [
                [ HASH (string),
                STATUS (integer),
                NAME (string),
                SIZE (integer in bytes),
                PERCENT PROGRESS (integer in per mils),
                DOWNLOADED (integer in bytes),
                UPLOADED (integer in bytes),
                RATIO (integer in per mils),
                UPLOAD SPEED (integer in bytes per second),
                DOWNLOAD SPEED (integer in bytes per second),
                ETA (integer in seconds),
                LABEL (string),
                PEERS CONNECTED (integer),
                PEERS IN SWARM (integer),
                SEEDS CONNECTED (integer),
                SEEDS IN SWARM (integer),
                AVAILABILITY (integer in 1/65535ths),
                TORRENT QUEUE ORDER (integer),
```

```
        REMAINING (integer in bytes),
        DOWNLOAD URL (string),
        RSS FEED URL (string),

        ...
],
"torrentm": [
        HASH (string),

        ...
],
"rssfeedp": [
        [ IDENT (integer),
        ENABLED (boolean),
        USE FEED TITLE (boolean),
        USER SELECTED (boolean),
        PROGRAMMED (boolean),
        DOWNLOAD STATE (integer),
        URL (string),
        NEXT UPDATE (integer in unix time), [
                [ NAME (string),
                NAME FULL (string),
                URL (string),
                QUALITY (integer),
                CODEC (integer),
                TIMESTAMP (integer),
                SEASON (integer),
                EPISODE (integer),
                EPISODE TO (integer),
                FEED ID (integer),
                REPACK (boolean),
                IN HISTORY (boolean) ],

                ...
                ] ],
        ...
],
"rssfeedm": [
        IDENT (integer),
        ...
```

```
        ],
        "rssfilterp": [
                [ IDENT (integer),
                FLAGS (integer),
                NAME (string),
                FILTER (string as regexp),
                NOT FILTER (string as regexp),
                DIRECTORY (string),
                FEED (integer as feed ID),
                QUALITY (integer in bytes),
                LABEL (string),
                POSTPONE MODE (integer),
                LAST MATCH (integer),
                SMART EP FILTER (integer),
                REPACK EP FILTER (integer),
                EPISODE FILTER STR (string),
                EPISODE FILTER (boolean),
                RESOLVING CANDIDATE (boolean) ],

                ...
        ],
        "rssfilterm": [
                IDENT (integer),
                ...
        ],
        "torrentc": CACHE ID (string integer)
}
```

The `"torrentp"` array contains a list of torrent jobs that have changed since the corresponding CACHE ID and is identical to the “torrents” array in format. Similarly the `"rssfeedp"` and `"rssfilterp"` arrays correspond to the “rss-feeds” and “resfilters” arrays respectively. The `"torrentm"` array contains a list of hashes for torrent jobs that have been removed since the corresponding CACHE ID. Similarly the `"rssfeedm"` and `"rssfilterm"` reflect rss feeds and filters that have been removed since the corresponding CACHE ID. A new CACHE ID will be given in torrentc and this can be used for the next list request.

Files List

To get the list of files in a torrent job, request this URI:

`http://[IP]:[PORT]/gui/?action=getfiles&hash=[TORRENT HASH].`

This will return the following:

```
{
        "build": BUILD NUMBER (integer),
        "files": [
                HASH (string),
                [
                        [ FILE NAME (string),
                        FILE SIZE (integer in bytes),
                        DOWNLOADED (integer in bytes),
                        PRIORITY* (integer) ,
                        FIRST PIECE (integer),
                        NUM PIECES (integer),
                        STREAMABLE (boolean),
                        ENCODED RATE (integer),
                        DURATION (integer),
                        WIDTH (integer),
                        HEIGHT (integer),
                        STREAM ETA (integer),
                        STREAMABILITY (integer) ],

                        ...
                ]
        ]
}
```

PRIORITY: This is an integer value that indicates the file's priority. The following is a list of the possible PRIORITY values and what each corresponds to:

- 0 = Don't Download
- 1 = Low Priority

- 2 = Normal Priority
- 3 = High Priority

This command accepts multiple hashes. It will return multiple "files" key/value pairs.

Torrent Job Properties

To get a list of the various properties for a torrent job, request:

```
http://[IP]:[PORT]/gui/?action=getprops&hash=[TORRENT HASH]
```

This will return the following:

```
{
        "build": BUILD NUMBER (integer),
        "props": [
        {
                "hash": HASH (string),
                "trackers": TRACKERS* (string),
                "ulrate": UPLOAD LIMIT (integer in bytes per second),
                "dlrate": DOWNLOAD LIMIT (integer in bytes per second)
                "superseed": INITIAL SEEDING** (integer),
                "dht": USE DHT** (integer),
                "pex": USE PEX** (integer),
                "seed_override": OVERRIDE QUEUEING** (integer),
                "seed_ratio": SEED RATIO (integer in per mils),
                "seed_time": SEEDING TIME*** (integer in seconds),
                "ulslots": UPLOAD SLOTS (integer)
        } ]
}
```

TRACKERS: This is a list of the trackers used by the torrent job. Each newline is represented by a carriage return followed by a newline ( ).

INITIAL SEEDING/USE DHT/USE PEX/OVERRIDE QUEUEING: These options are all integer values that indicate their respective states. The following is a list of the possible values and what each corresponds to:

- -1 = Not allowed
- 0 = Disabled
- 1 = Enabled

SEEDING TIME: This is an integer representing the minimum amount of time (in seconds) that µTorrent should continue to seed after it has finished downloading the torrent. A value of 0 (zero) means no minimum seeding time.

To change the properties for a torrent, the following URI can be used:

```
http://[IP]:[PORT]/gui/?action=setprops&hash=[TORRENTHASH]
&s=[PROPERTY]&v=[VALUE]
```

Multiple properties can be changed at once by appending more s= and v= pairs. Multiple torrent jobs can be modified in a single request by appending another hash= value along with its s= and v= pairs. Any following s= and v= pairs will modify the properties of the last specified hash.

For example, to set an upload rate limit of 10 KiB/s and a download rate of 20 KiB/s for [TORRENT HASH 1], while simultaneously setting 4 upload slots for [TORRENT HASH 2], request the following URI:

```
http://[IP]:[PORT]/gui/?action=setprops&hash=
[TORRENT HASH 1]&s=ulrate&v=10240&s=dlrate&v=20480&hash=
[TORRENT HASH 2]&s=ulslots&v=4
```

Actions

This section contains a list of all the other possible actions supported by the API. All actions are in the form `http://[IP]:[PORT]/gui/?action=`

?action=start&hash=[TORRENT HASH] This action tells µTorrent to start the specified torrent job(s). Multiple hashes may be specified to act on multiple torrent jobs.

?action=stop&hash=[TORRENT HASH] This action tells µTorrent to stop the specified torrent job(s). Multiple hashes may be specified to act on multiple torrent jobs.

?action=pause&hash=[TORRENT HASH] This action tells µTorrent to pause the specified torrent job(s). Multiple hashes may be specified to act on multiple torrent jobs.

?action=forcestart&hash=[TORRENT HASH] This action tells µTorrent to force the specified torrent job(s) to start. Multiple hashes may be specified to act on multiple torrent jobs.

?action=unpause&hash=[TORRENT HASH] This action tells µTorrent to unpause the specified torrent job(s). Multiple hashes may be specified to act on multiple torrent jobs.

?action=recheck&hash=[TORRENT HASH] This action tells µTorrent to recheck the torrent contents for the specified torrent job(s). Multiple hashes may be specified to act on multiple torrent jobs.

?action=remove&hash=[TORRENT HASH] This action removes the specified torrent job(s) from the torrent jobs list. Multiple hashes may be specified to act on multiple torrent jobs. This action respects the option "Move to trash if possible".

?action=removedata&hash=[TORRENT HASH] This action removes the specified torrent job(s) from the torrent jobs list and removes the corresponding torrent contents (data) from disk. Multiple hashes may be specified to act on multiple torrent jobs. This action respects the option "Move to trash if possible".

?action=removetorrent&hash=[TORRENT HASH] This action removes the specified torrent job(s) from the torrent jobs list and removes the corresponding torrent file(s) from disk. Multiple hashes may be specified to act on multiple torrent jobs. This action respects the option "Move to trash if possible".

?action=removedatatorrent&hash=[TORRENT HASH] This action removes the specified torrent job(s) from the torrent jobs list, removes the corresponding

torrent file(s) from disk, and removes the corresponding torrent contents (data) from disk. Multiple hashes may be specified to act on multiple torrent jobs. This action respects the option "Move to trash if possible".

?action=setprio&hash=[TORRENTHASH]&p=[PRIORITY]&f=[FILE INDEX] This action sets the priority for the specified file(s) in the torrent job. The possible priority levels are the values returned by "getfiles". A file is specified using the zero-based index of the file in the inside the list returned by "getfiles". Only one priority level may be specified on each call to this action, but multiple files may be specified.

?action=getxferhist This action returns the current transfer history.

?action=resetxferhist This action resets the current transfer history.

?action=getversion This action returns information about the server.

```
{
        "version": [
        {
                "product_code": "server",
                "ui_version": (revision number),
                "engine_version": (revision number),
                "major_version": (major version number)
                "minor_version": (minor version number)
                "user_agent": (user-agent)
                "version_date": (date product was built)
                "device_id": (identifies general type of target/OS)
                "peer_id": (identifier sent to peers)
        } ]
}
```

?action=add-url&s=[TORRENT URL] This action adds a torrent job from the given URL. For servers that require cookies, cookies can be sent with the :COOKIE: method (see here). The string must be URL-encoded.

?action=add-file This action is different from the other actions in that it uses HTTP POST instead of HTTP GET to submit data to µTorrent. The HTTP form must use

an enctype of "multipart/form-data" and have an input field of type "file" with name "torrent_file" that stores the local path to the file to upload to µTorrent.

Optional parameters to add-file and add-url actions:

&download_dir=<integer> This action determines which download directory to put the torrent in. The integer refers to the list of download dirs the client has configured (see list-dirs action). 0 always means the default directory. The list of available download directories must be created in the btsettings.txt file. On PC systems the directories can be created by users via the application, but not via the web UI. There is currently no API to create download directories.

&path=<sub_path> This action determines a subdirectory to put the file in, under the already chosen download directory. This path may not contain ".." and must be a relative path.

http://[IP]:[PORT]/gui/?action=list-dirs The return value has a list, download-dirs. Each entry in the list is a dictionary:

```
"download-dirs": [
{ "path": <full path>, "available": <available free disk space in MB>
...
]
```

http://<ip>:<port>/proxy?id=<info-hash>&file=<file index> The `&file=` parameter may be omitted if the torrent is a single file torrent. The call will return the entire file, or a part of it if a range request was made to it.

RSS Feed Actions:

?action=rss-remove&feed-id=[FEED ID] This action removes the corresponding RSS feed from the list of RSS feeds.

?action=rss-update&feed-id=[FEED ID] This action adds or updates an RSS feed.

Optional parameters for rss-update action:

&download_dir=<integer> This action determines which download directory to

put the torrent in. The integer refers to the list of download dirs the client has configured (see list-dirs action). 0 always means the default directory. The list of available download directories must be created in the btsettings.txt file. On PC systems the directories can be created by users via the application, but not via the web UI. There is currently no API to create download directories.

&feed-id=<integer> This parameter identifies the RSS feed receiving the updates. If this parameter is set to -1 or omitted a new RSS feed will be created and the following will be included in the return:

:: "rss_ident" : [RSS IDENT] (integer),

&url=<string> This parameter identifies the URL of the RSS feed.

&alias=<string> This parameter identifies the RSS feed alias.

&subscribe=<boolean> This parameter signifies whether or not the user wishes to subscribe to the RSS feed.

&smart-filter=<boolean> This parameter enables the smart filter functionality on an RSS feed.

&enabled=<boolean> This enables or disables the RSS feed.

&update=1 If this is set the RSS feed is forced to update instead of waiting for the next update interval. The update interval will be set accordingly.

RSS Filter Actions:

?action=filter-remove&filter-id=[FILTER ID] This action removes the corresponding RSS filter from the list of RSS filters.

?action=filter-update&filter-id=[FILTER ID] This action adds or updates an RSS filter.

Optional parameters for filter-update action:

?filter-id=<integer> The ID of the RSS filter to be updated. If this is omitted or set to -1 a new filter will be created and the following will be included in the return

value:

:: “filter_ident”: [FEED ID] (integer),

?name=<string> Name of the RSS Filter.

?save-in=<string> Directory to save the downloaded RSS torrents in.

?episode=<string> Episode expression to download.

?filter=<string> Download filter expression.

?not-filter=<string> Download exception filter expression.

?label=<string> Label to apply to torrents downloaded by this filter rule.

?quality=<integer> Minimum quality to accept when qualifying for download.

?episode-filter=<boolean> Set the enabling of downloading by episode.

?origname=<string> Filter original name.

?prio=<boolean> Prioritize this filter.

?smart-ep-filter=<boolean> Enable/disable smart episode filter.

?add-stopped=<boolean> Enable queuing of torrents selected by this filter as stopped for manual starting rather than adding them to the download queue.

?postpone-mode=<boolean> Enable/disable postpone mode.

?feed-id=<integer> RSS feed to associate this filter with. If set to -1 the feed is associated with all active RSS feeds.

Limitations

It is not possible to rename or relocate individual files in a torrent job.

Legal Notices