BitTorrent Software Development Kit

Version 3.0

Table of Contents

1 Overview

1.1 Introduction

The BitTorrent Software Development Kit (SDK) is designed for use in embedded systems, including network-area storage devices (NASs) and set-top boxes. It provides a state-of-the-art implementation of the BitTorrent protocol and a full-featured web-based user interface in a small footprint. In addition, the BitTorrent SDK aims to be highly-portable to a large number of devices and be easy to use both for integrators and end-users.

1.2 Features

The BitTorrent SDK 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
- BitTorrent DNA support. Managed peer assisted delivery with web seeding.

Additionally, the BitTorrent SDK includes two full-featured web-based user interface examples that are fully customizable and extensible by licensees.

2 Getting Started

The BitTorrent SDK consist of:

- an executable daemon (`bt`) that implements BitTorrent services and is used through an HTTP-based application programming interface (API);
- a `btdog` watchdog process that monitors `bt` process and restarts it in case it crashes. This improves reliability of the system.
- a reference implementation of a web-based user interface (UI) for managing BitTorrent downloads through a web browser.
- a second web-based user interface based on our Windows PC clients' and sharing their legacy web API.

In order to integrate the SDK into your product, you must:

- start the SDK process at device boot time or another appropriate time
- make sure the SDK runs continuously by monitoring the process
- enable control of the SDK through either direct user UI or as a system resource to your applications
- appropriately terminate the SDK process when shutting down the device or at another appropriate time
- periodically and/or at boot time, remove unneeded temporary files created by the SDK process

Reference interfaces are provided, and may be freely customized.

The two web-based UIs provided as a reference implementation of a standard PC-centric BitTorrent experience. You may use it as-is, customize its look-and-feel by modifying CSS styles, or extend it to provide new functionality or tighter integration with your device or other applications.

Please note that the reference applications are intended to demonstrate key features of the SDK and provide a base for licensees to work from. Customization and feature enhancement of the reference designs is the responsibility of the licensee.

Alternatively, Licensees may build a completely new UI using the BitTorrent SDK API. This is often preferable in devices whose primary interface is not a web browser. The BitTorrent SDK does not include a non-web based reference interface at this time.

2.1 SDK Usage

There are 2 ways to deploy SDK on the device:

- run the `bt` process under `btdog` supervision as: `btdog bt` (where `bt` is the name of the process to launch). `btdog` is a watchdog process that will restart the `bt` process in case it exits for any reason.
- run `bt -daemon` which starts the `bt` process in Unix daemon mode.

For the reason of reliability, we strongly recommend using `btdog`.

At startup the `bt` executable looks for `btsettings.txt` which allows the behavior of the SDK 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
bind_port: 9388
max_total_connections: 600
```

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

3 API Documentation

3.1 Overview

The API provides a simple remote procedure call (RPC) interface to the BitTorrent SDK. All API calls are HTTP GET or POST requests. Some API calls return nothing, others return a result. When a result is returned, it may be returned in any one of three formats selectable at call time:

- JSON
- XML
- bencoded

JSON format is recommended for JavaScript applications and is the default return type. XML is a widely-used, standards-based markup language. Bencoded results are small and easy to parse and, therefore, suitable for low-level languages like C, C++, C#, and Java.

JSON is the default return type. Other formats can be requested by appending the `format` argument to the query, with value equal to `xml` (for XML format) or `benc` (for bencoded format).

For example, the query `/api/app-settings-get` returns the current settings in JSON format, while the query `/api/app-settings-get?format=xml` returns them in XML format.

All results are returned as a dictionaries of key-value pairs. Future versions of the BitTorrent SDK may extend API results with new key-value pairs. Consequently, applications built upon the SDK should ignore unknown keys.

Successful API calls return 200 (OK) HTTP status response. Torrent-add with a URL returns the 202 (Accepted) HTTP status response.

3.2 API calls

3.2.1 Getting Application Settings

`/api/app-settings-get[?format=$format]`

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

Sample JSON result:

```
{"settings" :
 {"auto_bandwidth_management" : 1,
  "bind_port" : 6881,
  "conns_per_torrent" : 30,
  "max_dl_rate" : -1,
  "max_total_connections" : 400,
  "max_ul_rate" : -1,
  "max_ul_rate_seed" : -1,
  "seed_ratio" : 0,
  "seed_time" : 0,
  "ul_slots_per_torrent" : 4}}
```

Sample XML result:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<settings>
 <auto_bandwidth_management>1</auto_bandwidth_management>
 <bind_port>6881</bind_port>
 <conns_per_torrent>30</conns_per_torrent>
 <max_dl_rate>-1</max_dl_rate>
 <max_total_connections>400</max_total_connections>
 <max_ul_rate>-1</max_ul_rate>
 <max_ul_rate_seed>-1</max_ul_rate_seed>
 <seed_ratio>0</seed_ratio>
 <seed_time>0</seed_time>
```

```
 <ul_slots_per_torrent>4</ul_slots_per_torrent>
</settings>
</result>
```

3.2.2 Setting Application Settings

`/api/app-settings-set?$name1=$val1[&$name2=$val2...]`

Sets one or more application settings. Silently ignores unknown setting names and invalid setting values.

Returns: nothing.

3.2.3 Getting Information About Managed Torrents

`/api/torrents-get[?format=$format][&hash=$infohash1][&hash=$infohash]`

Returns a list of torrents and information about them. If no `hash` argument is provided, returns information for all torrents. If one or more `hash` arguments are provided, returns information only for torrents with those infohashes. Invalid `hash` arguments are silently ignored.

Returns an empty result if there are no torrents or none of the `hash` arguments is valid.

The information that is returned for each torrent is as follows:

hash (*string*): Infohash of the torrent.

caption (*string*): Name of the torrent.

size (*integer*): Size (in bytes) of the torrent (combined size of all files in the torrent).

done (*integer*): Number of bytes downloaded so far. If `done` is equal to `size`, the torrent has been downloaded completely.

dl_rate (*integer*): Current download rate in bytes per second.

ul_rate (*integer*): Current upload rate in bytes per second.

payload_download (*integer*): Total number of bytes downloaded in a current session (since starting the application).

payload_upload (*integer*): Total number of bytes uploaded in a current session (since starting the application).

peers_total (*integer*): Total number of peers for this torrent.

peers_connected (*integer*): Number of peers to which we are connected.

seeds_connected (*integer*): Total number of seeds (peers that have the complete torrent) for this torrent.

seeds_total (*integer*): Number of seeds to which we are connected.

private (*Boolean*): 1 (true) if this is a private torrent, 0 (false) otherwise.

state (*string*): State can be one of the following: "queued_for_checking", "checking_files", "connecting_to_tracker", "downloading", "finished", "seeding", or "allocating".

stopped (*Boolean*): 1 (true) if the torrent is stopped, 0 (false) otherwise.

distributed_copies (*string*): String representation of a floating point number representing the total number of distributed copies. A value of -1 indicates this torrent is in the "seeding" state. Otherwise, a value less than 1.0 means that a torrent cannot be fully downloaded from the current set of peers.

max_dl_rate (*integer*): Maximum download rate for this torrent in kilobytes per second. -1 means unlimited. Default value: -1.

max_ul_rate (*integer*): Maximum upload rate for this torrent in kilobytes per second. -1 means unlimited. Default value: -1.

An example JSON output:

```
{
  "torrents" : [ {
      "caption" : "Fedora-8-Live-KDE-i686",
      "distributed_copies" : "-1",
      "dl_rate" : 0,
      "done" : 732189042,
      "hash" : "5de112084598ee6b93f3b0602477d7efd1b47632",
      "max_dl_rate" : -1,
      "max_ul_rate" : -1,
      "payload_download" : 0,
      "payload_upload" : 0,
      "peers_connected" : 0,
      "peers_total" : 230,
      "private" : 0,
      "seeds_connected" : 0,
```

```
        "seeds_total" : 198,
        "size" : 732189042,
        "state" : "seeding",
        "stopped" : 1,
        "ul_rate" : 0
    }, {
        "caption" : "mspevack-ohio-linux-fest-2007.ogg",
        "distributed_copies" : "-1",
        "dl_rate" : 0,
        "done" : 351013357,
        "hash" : "d6e183d38da44d03ba56bb3018fb5c75d4de9917",
        "max_dl_rate" : -1,
        "max_ul_rate" : -1,
        "payload_download" : 0,
        "payload_upload" : 0,
        "peers_connected" : 0,
        "peers_total" : 11,
        "private" : 0,
        "seeds_connected" : 0,
        "seeds_total" : 10,
        "size" : 351013357,
        "state" : "seeding",
        "stopped" : 0,
        "ul_rate" : 0
    } ]
}
```

An example XML output:

```
<result>
<torrents>
 <item>
   <caption>Fedora-8-Live-KDE-i686</caption>
   <distributed_copies>-1</distributed_copies>
   <dl_rate>0</dl_rate>
   <done>732189042</done>
   <hash>5de112084598ee6b93f3b0602477d7efd1b47632</hash>
   <max_dl_rate>-1</max_dl_rate>
   <max_ul_rate>-1</max_ul_rate>
   <payload_download>0</payload_download>
   <payload_upload>114688</payload_upload>
   <peers_connected>2</peers_connected>
   <peers_total>228</peers_total>
   <private>0</private>
```

```
    <seeds_connected>0</seeds_connected>
    <seeds_total>195</seeds_total>
    <size>732189042</size>
    <state>seeding</state>
    <stopped>0</stopped>
    <ul_rate>1638</ul_rate></item>
  <item>
    <caption>mspevack-ohio-linux-fest-2007.ogg</caption>
    <distributed_copies>-1</distributed_copies>
    <dl_rate>0</dl_rate>
    <done>351013357</done>
    <hash>d6e183d38da44d03ba56bb3018fb5c75d4de9917</hash>
    <max_dl_rate>-1</max_dl_rate>
    <max_ul_rate>-1</max_ul_rate>
    <payload_download>0</payload_download>
    <payload_upload>0</payload_upload>
    <peers_connected>0</peers_connected>
    <peers_total>9</peers_total>
    <private>0</private>
    <seeds_connected>0</seeds_connected>
    <seeds_total>8</seeds_total>
    <size>351013357</size>
    <state>seeding</state>
    <stopped>0</stopped>
    <ul_rate>0</ul_rate></item>
  </torrents>
</result>
```

3.2.4 Setting the Properties of a Torrent

`/api/torrent-set-props?hash=$infohash&$name1=$val1[&$name2=$val2...]`

Set one or more properties of a given torrent. Invalid properties and invalid values are silently ignored.

Returns: nothing.

The torrent properties that may be set are:

max_dl_rate (*integer*): Maximum download rate for this torrent in kilobytes per second. -1 means unlimited. Default value: -1.

max_ul_rate (*integer*): Maximum upload rate for this torrent in kilobytes per second. -1 means unlimited. Default value: -1.

3.2.5 Starting a Torrent

`/api/torrent-start?hash=$infohash`

Starts the torrent with the given infohash. Infohashes can be obtained with the `/api/torrents-get` call. Does nothing if the torrent is already started.

Returns: nothing.

3.2.6 Stopping a Torrent

`/api/torrent-stop?hash=$infohash`

Stops the torrent with the given infohash. Infohashes can be obtained with the `/api/torrents-get` call. Does nothing if the torrent is already stopped.

Returns: nothing.

3.2.7 Adding a Torrent via URL

`/api/torrent-add?url=$url[&start=yes]`

Adds the torrent with the given URL, where `url` is a link to a torrent file accessible via HTTP. Invalid or inaccessible URLs are silently ignored.

If the optional `start` argument is `yes`, the torrent will start downloading automatically. Otherwise it must started explicitly with the `/api/torrent-start` API call.

Returns: nothing.

This call is asynchronous. It will return immediately without waiting to finish downloading the torrent file. This means that there might be a lag between when the `/api/torrent-add` call finishes and when the torrent appears in the list returned by `api/torrents-get`. This is why it returns a 202 (Accepted) HTTP status code.

Example:

`/api/torrent-add?url=http://torrent.fedoraproject.org/torrents/Fedora-8-Live-i686.torrent`

3.2.8 Adding a Torrent File

`/api/torrent-add[&start=yes]`

Add the torrent file provided by HTTP POST data. The POSTed torrent file data are expected in multipart/form-data encoded format. This call is synchronous.

Returns: nothing

3.2.9 Removing a Torrent

`/api/torrent-remove?hash=$infohash[&delete-torrent=yes][&delete-data=yes]`

Removes the torrent with the given infohash. If the optional `delete-torrent` argument is `yes`, also deletes the torrent file. If the optional `delete-data` argument is `yes`, also deletes the data for this torrent.

Returns: nothing.

It is recommended that `delete-torrent` and `delete-data` always be set to `yes` if the torrent has not yet been fully downloaded.

3.2.10 Getting a List of Files in a Torrent

`/api/torrent-get-files?[?format=$format][&hash=$infohash1][&hash=$infohash]`

Returns a list of files in a torrent and their properties. If no `hash` argument is provided, returns information for all torrents. If one or more `hash` arguments are provided, returns information only for torrents with those infohashes. Invalid `hash` arguments are silently ignored.

Returns an empty result if there are no torrents or none of the `hash` arguments is valid.

The properties returned for each file are:

id (*integer*): A unique id for the file (needed for some other API calls).

name (*string*): Name of the file (might include a directory).

size (*integer*): Size of the file.

done (*integer*): How much of the file has been downloaded.

pri (*integer*): Download priority of the file. Files with higher priority might be downloaded faster than other files.
0 means low, 1 means default (medium), 2 means high. -1 means: do not download at all.

Example JSON response:

```
{
  "torrents" : {
      "1606c977d334534b1bf12149399529a4c89b33d0" : [ {
          "done" : 719859712,
          "id" : 0,
          "name" : "Fedora-7-KDE-Live-i686\\Fedora-7-KDE-Live-
i686.iso",
          "pri" : 1,
          "size" : 719859712
      }, {
          "done" : 370,
          "id" : 1,
          "name" : "Fedora-7-KDE-Live-i686\\SHA1SUM",
          "pri" : 1,
          "size" : 370
      } ],
      "37a721f52791ddd0eac551e380e4716690cee6f9" : [ {
          "done" : 732942336,
          "id" : 0,
          "name" : "Fedora-8-Live-ppc\\Fedora-8-Live-ppc.iso",
          "pri" : 1,
          "size" : 732942336
      }, {
          "done" : 300,
          "id" : 1,
          "name" : "Fedora-8-Live-ppc\\SHA1SUM",
          "pri" : 1,
          "size" : 300
      } ]
  }
}
```

Example XML response:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<torrents>
 <1606c977d334534b1bf12149399529a4c89b33d0>
   <item>
     <done>719859712</done>
     <id>0</id>
     <name>Fedora-7-KDE-Live-i686%5CFedora-7-KDE-Live-i686.iso</name>
     <pri>0</pri>
     <size>719859712</size></item>
   <item>
     <done>370</done>
     <id>1</id>
     <name>Fedora-7-KDE-Live-i686%5CSHA1SUM</name>
     <pri>0</pri>
     <size>370</size></item>
 </1606c977d334534b1bf12149399529a4c89b33d0>
 <37a721f52791ddd0eac551e380e4716690cee6f9>
   <item>
     <done>732942336</done>
     <id>0</id>
     <name>Fedora-8-Live-ppc%5CFedora-8-Live-ppc.iso</name>
     <pri>0</pri>
     <size>732942336</size></item>
   <item>
     <done>300</done>
     <id>1</id>
     <name>Fedora-8-Live-ppc%5CSHA1SUM</name>
     <pri>0</pri>
     <size>300</size></item>
  </37a721f52791ddd0eac551e380e4716690cee6f9>
</torrents>
</result>
```

3.2.11 Setting the Download Priority of a File

`/api/torrent-file-set-priority?hash=$infohash&id=$id&pri=$pri`

Set the download priority of the file specified by the given infohash and file id. The priority argument may be:

-1: do not download the file
0: low priority

1: default (medium) priority
2: high priority

Silently ignores invalid `hash` or `id` or `pri`.

Returns: nothing.

3.2.12 Get File Content

`/api/torrent-file-get?hash=$infohash&id=$id`

Returns the content of the file specified by the given infohash and file `id`. File `id` may be obtained from the `/api/torrent-get-files` call.

If a `hash` or `id` argument is invalid, returns 404 HTTP response.

4 Web UI

The web-based UIs are provided as reference implementations. You may use them as-is, customize their look-and-feel by modifying CSS styles, or extend them to provide new functionality or tighter integration with your device.

Alternatively, you may build a completely new UI on top of the BitTorrent SDK API or μTorrent API.

The BitTorrent SDK UI provides a Downloads view, from which torrent downloads may be managed, and a Settings page, where several settings may be customized.

The BitTorrent SDK actually comes with two alternative Download views:

`fe01.html` a power-user view similar to PC-based clients such as μTorrent.

`fe02.html` a simpler user interface, hiding most detail, appropriate for new or non-technical users

4.1 Installing the Reference UI

The reference UI consists of HTML, CSS, JavaScript and image files. The files can be served by the `bt` process or a standard HTTP server (e.g. Apache).

BitTorrent does not provide end-user customer support for the SDK. The support link that appears on the web UI is initially set to `http://SDK_LICENSEE_SUPPORT_PAGE/` and must be changed by licensees to a customer support site or knowledge-base operated by the licensee.

4.1.1 Serving Files Directly from the `bt` Process

The `bt` process can serve as an embedded HTTP server and serve all the HTML, CSS and JavaScript files. The advantage of this scenario is that no other software is required; you need only configured the `bt` process properly. If you already have an HTTP server running, the `bt` process must use a different HTTP port.

4.1.2 Serving Files from an HTTP Server

If you already have an HTTP server running, the HTTP server can serve all HTML, CSS and JavaScript files, while the `bt` process provides data and functionality via its HTTP API.

One problem with this approach is that the requests to the `bt` process are done via JavaScript `XMLHTTPRequest` calls which are restricted by the same-origin policy of most browsers. That means that an `XMLHTTPRequest` call can only issue requests to exactly the same server from which HTML pages are served. Since the HTTP server (serving HTTP files) runs on a different port than the `bt` process responding to the `XMLHTTPRequest` calls, this violates same-origin policy.

There are several possible workarounds. One workaround, specific to the Apache HTTP server, is to configure the server to proxy `XMLHTTPRequest` calls in a way that is invisible to the HTTP server.

First, you must make sure that Apache is compiled with `mod_rewrite`, `mod_proxy` and `mod_proxy_http`. These modules also must be loaded, which is done by adding the following to `httpd.conf`:

```
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

Assuming that both the HTTP server and the `bt` process are running on localhost (127.0.0.1) and the `bt` process is serving HTTP requests on port 8080, the following lines in `.htaccess` file will correctly configure proxying:

```
RewriteEngine On
RewriteBase /
RewriteRule ^api/(.*)$ http://127.0.0.1:8080/api/$1 [P]
```

4.1.3 Using the First Reference UI (`fe01.html`)

First reference web UI consists of following files:

- fe01.html
- btbase.css
- btlteIE6.css
- btsettings.html
- btfe00.css
- btsdk.js
- btuicommon.js
- img

You may wish to `fe01.html` to `index.html` and change href references in `btsettings.html` from `./fe01.html` to `index.html`

4.1.4 Using the Second Reference UI (`fe02.html`)

Second reference web UI consists of following files:

- fe02.html
- btbase.css
- btlteIE6.css
- btsettings.html
- btfe00.css
- btsdk.js
- btuicommon.js
- img

You may wish to rename `fe02.html` to `index.html` and change href references in `btsettings.html` from `./fe01.html` to `index.html`

4.2 Installing the µTorrent Web UI

The µTorrent Web UI is contained in the archive file `webui.zip`.

4.2.1 Customizing the uTorrent Web UI

Place the `webui.zip` archive file in an empty directory, then run the `unzip` program on the archive. Edit the files as needed to customize them to your needs. Here are some suggestions.

You'll want to customize the banner at the top of the μTorrent web UI to refer to your product. Here's what to do.

If you want to replace the BitTorrent logo with a different logo, replace the file `static/images/gui/sdk_title_sm` with a suitable image of the desired logo.

If you want to replace the text in the banner, edit the file `index.html`, replacing the text below the comment `<!-- /* Bannerbar -->` with the text you would like to present.

You may also want to select a default language for the μTorrent web UI. Currently the default language is set to `en` (English). If you would like to set the default language to something else, edit the file `static/js/gui/webui.js`, replacing the text `"lang": "en",` (part of the definition of `this.config` in the `init` function) with the identifier of your preferred language. See the names of the files in `static/js/gui/lang` to select from the names of the supported languages.

Once you have completed customization, or you would like to view your work in progress, run the `zip` program on the unarchived files to recreate the file `webui.zip`.

4.2.2 Deploying the uTorrent Web UI

You must place `webui.zip` in the directory on your device from which you invoke the `bt` process.

5 Application Settings

Settings fall into two categories:

- internal settings, whose values can only be set in the `btsettings.txt` file;
- regular settings, whose values can be set in the `btsettings.txt` 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)

5.1 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.

webui_enable (*string*) Default value: "all". Determines which HTTP RPC API to enable. "sdk" will enable the SDK 2.0 compatible interface, "ut" will enable only the μTorrent web interface, "all" will enable both web interfaces.

webui_port (*integer*): Default value: 9090. Port number where the `bt` process accepts HTTP RPC API calls to support the SDK 2.0-compatible HTTP interface. If the `bt` process also serves HTML files (see `webui_server_files` setting), also the port of HTTP server.

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

webui_serve_files (*boolean*): Default value: true. If true, the `bt` process will act as an HTTP server and serve HTML, CSS and JavaScript files needed for webui. When set to false, the web UI files will be served by a stand-alone HTTP server (like Apache).

webui_dir_files (*string*): Default value: "webui". Name of the directory with HTML, CSS and JavaScript files that constitute the web UI. It can be an absolute path (recommended) or set relative to the current directory of the `bt` process.

webui_root (*string*): Default value: "/". If the `bt` process also acts as an HTTP server, this value represents the prefix for the web UI. For example, if `webui_root` is "/foo/", a request for "/foo/bar.html" will make the `bt` process return the file `bar.html` from the `webui_dir_files` directory.

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 `btsettings.txt`, 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 `bt` process creates temporary files with a `.utt` extension - if a value for this setting is specified, the `bt` 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.

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

natpmp (*boolean*): If true, NAT-PMP functionality for mapping ports is used by `bt`. 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.

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 device does not supply ifaddrs.h, or 2) you want the `bt` 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 `bt` 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 `bt` process starts.

5.2 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.

ul_slots_per_torrent (*integer*): Default value: 4. Maximum number of peers that can download a given torrent at the same time.

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.

6 μTorrent API

6.1 What is the μTorrent Web UI?

The BitTorrent SDK 3.0 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.

6.2 What is the μTorrent Web API?

It is an API to access the functions of the WebUI built into the application. This is independent of the standard WebUI reference implementation and requires nothing on the application-side besides having the option enabled in the SDK. 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.

6.3 General notes for API access

This alternate API must be enabled in the btsettings.txt through the setting `webui_enable`. The base URI to access it 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=[TORRENT 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”.

6.4 Client Discovery

It is possible to discover a running µTorrent client by scanning a series of ports on the local machine. The ports should be scanned in order, and the first successful hit should terminate the search. The sequence of ports is defined by the following formula:

```
port = 10000 + 5x + 3x^2 + 7x^3
```

Where `x` starts at 0 and is incremented by one each try.

To test a port to see if µTorrent is running, connect to it over the loopback device (`127.0.0.1`) and make an HTTP request for:

```
/gui/pingimg
```

If this returns a 1x1 pixel bmp image, you have found the µTorrent port. This port can be used in the same way that the user configured port to make requests to the web UI. The request for `/gui/pingimg` will only work for connections over the loopback device (localhost).

This python script illustrates how to discover µTorrent:

```
import urllib

# discovers utorrent on the local machine and returns the
# port number it's using, or -1 if not found
def discover_utorrent():
  #  port = 10000 + 5x + 3x^2 + 7x^3
  for i in xrange(0, 50):
    port = 10000 + 5*i + 3*i*i + 7*i*i*i
    try:
      response = urllib.urlopen( \
        'http://127.0.0.1:%d/gui/pingimg' % port)
      return port
    except:
      pass
  return -1

print 'discover utorrent'
port = discover_utorrent()
if port == -1:
  print 'not found'
  sys.exit(1)

print 'found utorrent on port %d' % port
```

Here's an example in Javascript:

```
var _target = '127.0.0.1';
var _port = 10000;

function scan_port(callback, i)
{
```

```
        var port = 10000 + 5*i + 3*i*i + 7*i*i*i;

        var img = new Image();
        img.onerror = function() { if (port > 50000) callback(-
1); else scan_port(i + 1); };
        img.onload = function() { callback(port); }
        img.src = 'http://' + _target + ':' + port + '/gui/pingimg';
}

function on_port(port)
{
        if (port >= 0)
        {
                // uTorrent was found running on port _port
                _port = port;
        }
}

function detect_ut()
{
        scan_port(on_port, 0);
}
```

6.5 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.

6.6 Pairing

As a way to simplify the process of setting up a webUI application for uTorrent, a 3rd party application can request to *pair* with uTorrent. This lets the application gain access to uTorrent's

web UI with less involvement from the user.

Normally when a user sets up a 3rd party application to talk to uTorrent, the user needs to:

1. Enable the web UI
2. Set a username and password
3. Tell the 3rd party application which port uTorrent is running on and which username and password to use to access it.

With the pairing feature, a 3rd party application can, together with the client discovery feature, reduce these steps to a single confirmation from the user.

In order to pair with uTorrent, send the following request:

```
http://127.0.0.1:<port>/gui/pair?name=<name of application>
```

Where *port* is the port uTorrent's web UI is listening on. This call will work whether the web UI is enabled or not. If the user has chosen to disable the pairing feature, this call will always fail.

When uTorrent receives this request, it presents the user with a dialog asking for permission to grant access to this application. If granted, uTorrent will generate a secret key and return it in the body of the HTTP response.

This secret key opens up access to the web UI to the 3rd party application, even when the regular web UI is disabled. The reason why the regular web UI doesn't have to be enabled is so that there's no default username and password that could be used as an exploit to gain access to a user's client.

The 3rd party application authenticates with the web UI with the username "pairing" and the key that was returned by the `/gui/pair` call as the password.

As a security measure, you may only access the web UI through the `pairing` username and the `/gui/pair` from the localhost.

The following python script illustrates how to pair with uTorrent:

```
import urllib
import sys

def pair(port):
```

```
  try:
    return url-
lib.urlopen('http://127.0.0.1:%d/gui/pair?name=test' % port).read(40)
  except:
    return None

port = int(sys.argv[1])
print 'send pairing request'
key = pair(port)

if key == None:
  print 'user rejected'
  sys.exit(1)

print 'you can now log in to the web UI from localhost:'
print 'http://pairing:%s@127.0.0.1:%d/gui' % (key, port)
```

When *token_auth* is enabled in the client (see Token Authentication System (important!)), the client will accept the key itself as a valid token. It is suggested to always pass the key as the `&token=` parameter.

6.7 Using JSONP access

In order to support web sites to access uTorrent through Javascript, any WebUI call accepts a `&callback=` parameter. If this parameter is specified, the resulting response body will be wrapped with a Javascript function call to the specified function.

This lets you make webUI calls as JSONP calls from a web page, to circumvent the same-origin policy. Since specifying username and password in HTTP URLs:

```
http://<username>:<password>@domain.com/path
```

is not standard, and not supported by all browser, there is a secondary way of authenticating as a paired application.

Instead of providing a `Authorization` header in the HTTP request, the pairing key can be passed as a `&pairing=` query string parameter.

Here's an example of how to do this in Javascript:

```
var _target = '127.0.0.1';
```

```
var _port = 10000; // should be auto-detected
var _pass = '';

function call_jsonp(param, callback)
{
        var previous = document.getElementById('getdata');
        var script = document.createElement('script');

        var src = _target + ':' + port + '/gui/' + param;
        if (callback) src += '&callback=' + callback;
        if (_pass) src += '&pairing=' + _pass;
        script.src = src;
        script.type = 'text/javascript';
        script.id = 'getdata';
        var dt = document.getElementById('jsonp');
        dt.replaceChild(script, previous);
}

function on_pass(pass)
{
        // the user accepted the pairing.
        // the pairing key is set in _pass
        _pass = pass;
}

function pair()
{
        call_jsonp('pair?name=test%20script', 'on_pass');
}

function on_list(obj)
{
        // obj is the object containing all the
        // information returned from uTorrent
}

function getlist()
{
        print_message('query torrent list');
        call_jsonp('?list=1&token=' + _pass, 'on_list');
}
```

Which requires the following in the html page as well:

```
<span id="jsonp"><script id="getdata"></script></span>
```

6.8 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)        ],
        ...
 ]
}
```

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&v=40
```

6.9 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),
                STREAMID (string),
                STREAMING PROGRESS (integer) ],

                ...
        ],
        "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.

STREAMID: Generated string used to identify the stream

STREAMING PROGRESS: This is an integer value which represents the percentage of the file which is ready to be streamed. If set to -1, it is streamable, but we are not preparing it for stream. If set to -2, the file is not streamable.

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),
        STREAMID (string),
        STREAMING PROGRESS (integer) ],

        ...
],
"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 "rssfeeds" 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.

6.10 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),
                        STREAMING PROGRESS (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

STREAMING PROGRESS: This is an integer value which represents the percentage of the file which is ready to be streamed. If set to -1, it is streamable, but we are not preparing it for stream. If set to -2, the file is not streamable.

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

6.11 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 sec-
ond),
                "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
```

6.12 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=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.

&sub_path=<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": <avail-
able 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 wether 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.

6.13 Limitations

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

7 Integration Notes

We advise the following best practices:

- Set the dir_temp_files parameter in btsettings.txt to define a temporary directory on the device's hard drive because typically on a device /tmp is a small flash-based partition. Periodic deleting of the files in that directory would be useful; this could be done at boot time or on a cron job. Making this directory only for use by the SDK is useful because that way it is easier to sort through the files (typically named `*.utt`).
- Wherever the SDK binaries are placed (e.g. on a read-only partition), the current directory from which the binaries are run should be on a writable partition because various files will be written there.
- If your device is already using ports 8080 and/or 9090 for another service, use the parameters in btsettings.txt to define the desired port(s) for the HTTP interfaces.
- Choose one or both HTTP API interfaces using the parameter in btsettings.txt.
- If hard disk partitions are marked as noexec, be sure to store the btdog and bt programs on an executable partition.

8 API Changes

8.1 API differences between SDK Versions 2.0 and 3.0

- Removed ability to get/set per-torrent "max_connections" and "max_uploads" settings.
- Added ability to specify the directory in which temporary files are created.
- Added ability to specify the directory in which torrent files are auto-loaded.
- Added ability to specify the port serving the uTorrent-compatible HTTP API.
- Added ability to specify which HTTP user interface to use.
- Added ability to include local peers in rate limiting
- Added ability to specify the slze limit for read/write/piece caching.
- Added ability to specify one or more directories as download directories (dir_download).

9 Legal Notices