BitTorrent App Software Development Kit

Work in progress

Table of Contents

1 Overview

1.1 Introduction

BitTorrent apps make it possible to extend the BitTorrent user interface with a customized interface. This interface can intergrate with the client and display information gathered from the internet as well as the client itself, such as download progress for specific torrents. This document will describe how to build such a btapp and how to integrate with BitTorrent.

Every app is described in a .btapp file. This file may contain icons, graphics, UI layout and program logic. The file format of a .btapp is described later in this document.

1.2 The btapp interface

The interface to a BitTorrent btapp is described by HTML (and Cascading Style Sheets, CSS). The program logic for a btapp is written in JavaScript in the same HTML file. This file is embedded inside the .btapp file. In order to maintain a responsive user interface, HTML is never served off of a server, it is always embedded in the btapp file so that it is rendered immediately, without waiting for a round trip to a server.

Whenever the user navigates away from the interface of a btapp, the whole HTML renderer is taken down, and all execution of Javascript is stopped.

The renderer does not allow the HTML page to ever change, except when it's modified by Javascript. There's no way to reload a page or load a page off of the internet. These restrictions are in place to guarantee a maximum level of responsiveness to the user. The only objects a page can load from the internet are Javascript and images.

Any regular `http` or `https` links will be handed off to the windows shell when clicked on. i.e. they will open in a separate browser window.

Any links starting with `magnet:` will be handled by simply adding the torrent the magnet link refers to. Any link starting with `torrent://` is assumed to be an HTTP URL pointing to a .torrent file. When clicked, this torrent file is downloaded and added.

...

1.3 Integrating with BitTorrent

A btapp can integrate with BitTorrent by talking to its WebUI interface. This interface is described in detail in the next chapter. In order to gain access to this API, a username and password is needed, or pairing key. Every btapp gets its own pairing key when installed. This pairing key is passed into the HTML page as querystring arguments. The listen port used by the client is also required, this is also passed in on the query string.

http:// *implementation-defined* ?pair=<40-byte-key>&port=<listen-port>

In order to make requests to this API, you might be restricted by Javascript's same origin policy. This is why it supports JSONP, which is a workaround for this restriction.

In Javascript, a request to list torrents would like this using JSONP:

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

        var src = 'http://127.0.0.1:' + port + '/gui/' + param;
        if (callback) src += '&callback=' + callback;
        src += '&pairing=' + pair;
        script.src = src;
        script.type = 'text/javascript';
        script.id = 'getdata';
        var dt = document.getElementById('jsonp');
        dt.replaceChild(script, previous);
}

call_jsonp('?list=1', 'on_list');
```

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

A btapp can only modify and query the status of torrents that were downloaded via that btapp. This is a security measure to avoid giving btapps full control over the client.

...

1.4 Example HTML

This is an example HTML file that just lists the torrents in the client in a table with the size of each one to the right. It uses the `pair` and `port` keys passed in as query string arguments to build the jsonp url to make the calls to the client.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script language="JavaScript" type="text/javascript">
/*<![CDATA[*/

function gup(name)
{
        name = name.replace(/[\[]/,"\\\[").replace(/[\]]/,"\\\]");
        var regexS = "[\\?&]"+name+"=([^&#]*)";
        var regex = new RegExp( regexS );
        var results = regex.exec( window.location.href );
        if( results == null )
                return "";
        else
                return results[1];
}

var _url_prefix ='http://127.0.0.1:' + gup('port') + '/gui/' ;
var _pair = gup('pair');

function add_torrent(name, size)
{
        tt = document.getElementById('torrentDiv');
```

```
        tt.innerHTML += name + ' (' + size + ' bytes)' + "<br/>";
}

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

        var src = _url_prefix + param;
        if (_pair) src += '&token=' + _pair + '&pairing=' + _pair;
        if (callback) src += '&callback=' + callback;
        script.type = 'text/javascript';
        script.id = 'getdata';
        script.src = src;
        var dt = document.getElementById('jsonp');
        dt.replaceChild(script, previous);
}

function on_list(obj)
{
        torrentList = obj['torrents'];
        for (i = 0; i < torrentList.length; ++i)
        {
                add_torrent(torrentList[i][2], torrentList[i][3]);
        }
}

function getlist() { call_jsonp('?list=1', 'on_list'); }
function on_addurl(obj) {}
function addtorrent(url) { call_jsonp('?action=add-
url&s=' + escape(url), 'on_addurl'); }

/*]]>*/
</script>
        </head>
        <body class="frontpage">
                <a href="" onclick="javascript:getlist();return false;">Get tor-
rent list</a>
                <br/><br/>
                Torrent URL: <in-
put type="text" size="140" id="torrentURL" value="http://www.mininova.org/get/3190508"/><
                <a href="" onclick="javascript:addtorrent(document.getElementById('torren
rent</a>
                <br/><br/>
                <div id="torrentDiv"></div>
```

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

1.5 Usability and Design Considerations

1.5.1 Icons

A BitTorrent app should include at least 2 icons to represent it in the BitTorrent UI. The pixel dimensions should be: 16x16 and 32x32. Only one is currently required (as --favicon) and it must be 16x16.

1.5.2 Minimum Font Size

To ensure users will be able to read the text contained in the display of btapp content, a font size of 9px should be the absolute minimum. Even this size will not work for all font faces and languages, so a higher font size is encouraged.

1.5.3 Display Dimensions

Btapp content will be displayed in an area of BitTorrent's UI that does not have a fixed height or width. Flexible widths and minimized use of scrollbars is encouraged but not required.

1.5.4 Torrent Status Feedback

The BitTorrent API can be used to provide constant feedback to the user on the progress and status of torrent downloads. If a btapp offers the ability to initiate torrent downloads, use of the API to provide feedback on those torrents' status is encouraged.

1.6 Building a .btapp file

There's a tool called `makebtapp.py` bundled with this package. It is used to transform the files used to describe your app into a .btapp file. It essentially just encodes the files into an xml document.

The simplest use case for this app is to provide one html file and one icon file (16x16 pixels, bmp file). To build the example app, run the following command:

```
makebtapp.py --html btapp-example.html --favicon test.bmp --btapp test.btapp -
-name test --autoupdate-url http://your.domain/path/to/update
```

The .btapp file can then be placed in the Application Data/uTorrent/apps directory for uTorrent to load. Start uTorrent and click the Apps item in the left sidebar.

Your app should now show up under the Apps parent, click on it to render the html in the right pane.

1.7 File format

...

2 μTorrent API

2.1 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.

2.2 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.

2.3 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".

2.4 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);
}
```

2.5 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.

2.6 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.

2.7 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>
```

2.8 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`

2.9 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.

2.10 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.

2.11 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
```

2.12 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.

2.13 Limitations

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

3 Legal Notices