Subtitles Exchange Support: User Manual

Andrea Reale
andrea.reale@mentalsmash.org

28th June 2010

About

This short guide is intended to give insights of how to use the services offered by the Subtitles Exchange subsystem implemented in Triber.

It is directed to developers who want to interact with the Subtitles subsystem to publish, retrieve and search for subtitles: it gives an overview of the API and of the rationale behind it.

For things that you don't find in this manual, you may have more luck taking a look at:
`http://www.tribler.org/wiki/RichMetadata`

Contents

Chapter 1

Quick Start

This chapter is intended to be 10 minutes reference for readers who are only interested in how to use the Subtitles subsystem, and do not need to understand how things work in the details. Only the basic API functionalities will be introduced. For the full API please consult [2].

1.1 Introduction

The Subtitle Exchange Subsystems extends the channel concept in Triber adding the ability to publish, search and retrieve subtitles published within channels.

With it an user can associate one or more subtitles in different languages to an item (i.e. a torrent) in its channel. Other peers will be able to access information about subtitles availability in others' channels and to retrieve them remotely. It is important to remark that every subtitle in a certain language is strictly associated to an item in a channel; since at the time of writing there is a one-to-one correspondence between items and torrents, a subtitle can be uniquely identified by the triple *(channel_id, infohash, language)*, where *channel_id* is the PermID of the channel owner, *infohash* is the 20 bytes infohash of the torrent corresponding to the item, and *language* is the language of the subtitle.

Before explaining how to work with subtitles, it might be a good idea to list the functionalities that are currently implemented and the ones that are missing.

What the Subtitle Exchange Subsystem does:

- Allow a Tribler user to enrich items in its *channel* attaching subtitles to them.
- Disseminate information about subtitles availability throughout the *Overlay Network*.
- Replicate subtitle contents in the network at *subscribers* hosts

Listing 1.1: Retrieve the SubtitlesSupport instance

```
from Tribler.Core.Subtitles.SubtitlesSupport \
    import SubtitlesSupport

...

def method_using_subtitles():
    # retrieve the correct instance
    sub_supp = SubtitlesSupport.getInstance()

    # do things with it
    ...

...
```

- Provide a simple API trough which publish, search, and retrieve subtitles contents both locally or in a remote fashion.

What the Subtitle Exchange Subsystem does not:

- Cope with any kind of GUI integration.
- Integrate and synchronise subtitles with video playback.
- Deal with different metadata other then subtitles (even tough everything is ready for an easy extension).
- Anything else that is not explicitly mentioned in this manual.

1.2 SubtitlesSupport Facade

The basic API is kept as simple as possible. So simple that is possible to perform all the basic interactions just calling methods of a single object acting as *facade* for the entire subsystem. This object is an instance of `Tribler.Core.Subtitles.SubtitlesSupport.SubtitlesSupport` from now on referred with the unqualified name *SubtitlesSupport*.

There is a single instance of this object in a running Tribler Session, and you should always use that instance, since it is initialized at startup with the proper arguments. To retrieve it you can call the static method `SubtitlesSupport.getInstance()` as shown in listing 1.1

Once you have a reference to the singleton instance you can use its methods to publish, search and retrieve subtitle contents. In the next subsections we will show how to perform each of these tasks.

Listing 1.2: Publishing a subtitle

```
...

lang = 'nld' # ISO 639-2 code for dutch language
path = os.path.join('path','to','subtitle.srt')
# the infohash of one of the torrents
# in the local channel
infohash = ...

# sub_supp is the SubtitleSupport single instance
# we have retrieved before
sub_supp.publishSubtitle(infohash, lang, path)

...
```

1.2.1 Publishing a subtitle

Let's assume that an user wants to publish a subtitle for the torrent of a film he previously added to its channel. To do that he needs to have a copy of the file containing subtitles on his local machine. Currently only subtitles in SubRip format (.srt) [1] are supported.

A possible GUI will allow him to select the item in the channel to attach subtitle to, and of course to specify the local path where the .srt file is, and the language of the subtitle he wishes to publish.

After the user has performed the necessary associations the `SubtitlesSupport` instance can be used to do the actual publishing: all it is needed is to call the `publishSubtitle()` instance method.

- `publishSubtitle(infohash, lang, pathToSrt)` :

  infohash is the binary string representing the infohash of the torrent to associate the subtitle to. The method assumes that such a torrent has already been added to the local channel.

  lang is a three character language code specifying the language of the subtitle; for a list of the supported language codes please see the on the wire protocol specification available at [3].

  pathToSrt is a local file system path pointing to the file containing a copy of the subtitle in srt format.

If everything works, after the method returns, the subtitle is copied in a dedicated directory located into the Tribler state dir, and all the necessary information is stored in the MegaCache. Dissemination for the added subtitle automatically starts.
An exemple of the subtitle publishing procedure is shown in listing 1.2.

1.2.2 Searching for available subtitles

There are two different methods in `SubtitlesSupport` that are intended to be used to retrieve information about available subtitles.

- `getSubtitleInfos(channel, infohash)`
- `getSubtitleInfosForInfohash(infohash)`

The first one can be used to search for subtitle for an element in a particular channel; this can be useful for example while browsing a channel. On the other hand the second method, given an infohash, searches for subtitles available in all the known channels. This can be used, for example, while showing the results from a keyword search, in order to show all the subtitles associated to an hit in different channels.

`getSubtitleInfos()` returns a dictionary: each value is an instance of `SubtitleInfo` and its key is a three characters string corresponding to the language code of the subtitle. A returned dictionary could look like:

```
{
    'eng' : subtitleInfo1,
    'ita' : subtitleInfo2,
    'deu' : subtitleInfo3
}
```

In the example there are three subtitles available, respectively in English, Italian and German.

Similarly `getSubtitleInfosForInfohash()` returns a dictionary: this time the keys of the dictionary are channel identifiers (PermIDs), and their associated value is itself a dictionary whose keys are language codes and whose valeus are `SubtitleInfo` instances:

```
{
    channel_id1 : {
                    'nld' : subtitleInfo1,
                    'rus' : subtitleInfo2
                  },

    channel_id2 : { 'kor' : subtitleInfoN }
}
```

In this case subtitles were found in two channels identified by PermIds *channel_id1* and *channel_id2*. In the first subtitles in Dutch and Russian are available; in the second only subtitles in Korean have been published.

The `SubtitleInfo` class is defined in module `Tribler.Core.Subtitles.MetadataDomainObjects.SubtitleInfo`. It contains all the basic information about a subtitle, as shown in listing 1.3, trough several examples.

Listing 1.3: How to use the SubtitleInfo class

```
...

# sub_supp is the SubtitleSupport singleton instance
# retrieved before
# channel and infohash are byte strings identifying
# a channel and the infohash of a torrent in it
results = sub_supp.getSubtitleInfos(channel, infohash)

for lang, subInfo in results.iteritems():
    print "Hit for a subtitle in %s" % lang

    # the lang property of a SubtitleInfo
    # reflects the key in the dictionary
    assert lang == subInfo.lang

    # SHA-1 checksum of the subtitle. It is always
    # set for a valid subtitle
    sha1Checksum = subInfo.checksum
    print "Subtitle's sha1 is %s" % sha1Checksum

    localPath = subInfo.path

    # if localPath is None it means that the subtitle
    # is not locally available
    if localPath is None:

        # the subtitleExists method does the same
        # thing
        assert not subInfo.subtitleExists()
        print "Subtitle not available locally"
    else:

        print "Subtitle is available at %s" % localPath

        # if the subtitle is available the
        # checksum can be verified
        print "Checking checksum..."
        if subInfo.verifyChecksum():
            print "OK"
        else
            print "FAIL"

    ...
```

1.2.3 Retrieving a subtitle

Assume that you've searched for some subtitles for a torrent, and now you have a bunch of `SubtitleInfo` instances, as it has been explained in Section 1.2.2.

For some of them the property `path` will be set to a local path in the file system pointing to the actual .srt file. You can double check that this file actually exists calling the `subtitleExists()` method on the `SubtitleInfo` object. If that is the case there is no need to retrieve remotely the contents and you can use the subtitles simply reading the data at the given path.

On the contrary, for other `SubtitleInfo`s a local path could not be available (again, you can check it by calling `subtitleExists()`). In that case you might want to retrieve the .srt file remotely.

To accomplish this task a method in `SubtitlesSupport` exists.

- `retrieveSubtitleContent(channel, infohash, subtitleInfo, callback)`

Calling this method schedules a remote query directed to peers who are known to have the desired subtitle. The method has no return value.

The `callback` parameter in the method signature is intended to be used as a notification mechanism informing when the subtitle content is in. It must be a function accepting a single parameter. This function will be called by the *OverlayThread* when the requested .srt is locally available and its parameter will be bound to an instance of `SubtitleInfo` whose `path` property reflects the local path of the subtitle file.

It is important to notice that for several reasons it is possible that the subtitle is never received. In such a case the callback method is never called. Listing 1.4 shows how to use this functionality.

1.3 Final remarks

That's it: now you should know how to interoperate with the Subtitles Subsystem for what concerns the basic functionalities. If you're interest in understanding how things work please consult the Full API documentation and the other documents listed in the references.

Listing 1.4: Retrieving subtitle contents

```
...

sub_supp = SubtitleSupport.getInstance()

# we have the subInfo SubtitleInfo instance from previous
# interactions

def call_me_back(retrieved_sub_info):

    # notice that retrieved_sub_info might not be the same
    # instance as subInfo

    print "Subtitle now available at %s" % \
        retrieved_sub_info.path

sub_supp.retrieveSubtitleContent(channel, infohash,
                                 subInfo, call_me_back)

print "Subtitle request scheduled to be sent"

...
```

Bibliography

[1] VideoLAN Wiki: SubRip Subtitle Format Description [Online]. `http://wiki.videolan.org/SubRip`, June 2010.

[2] A. Reale. Tribler Subtitle Exchange Subsystem. API documentation. `http://subtitles.zqx.net`, June 2010.

[3] A. Reale. Tribler wiki: Richmetadata [online]. `http://wiki.tribler.org/trac/wiki/RichMetadata`, June 2010.