SpokenMedia Transcript Editing Communication Protocol

This document describes the communication protocol and data format for the SpokenMedia Transcript Editor client. Both the Client and the Service are expected to conform to the normative descriptions below.

Transcripts can exist in two states, low quality and high quality. The mechanics of editing are specific to each state. A transcript is expected to begin in the low quality state, where many edits to the text itself may be required. Once the transcript has been improved, it can be moved to the high quality state. Individual phrase editing is allowed, but few improvements are expected to be necessary by the time a transcript reaches high quality.

Authentication and Authorization

As a user-based system, verifying users and insuring they behave within their rights to modify transcripts is key to the health and operation of the system. Due to time constraints, full components for authentication and authorization may not be ready; OpenID[1] is recommended as a single sign-on solution, and authorization for all authenticated users to perform these edits is recommended as a solution for the time being.

Transcript Format

All transcript text is expected to be encoded in UTF-8. All transcripts will be in plain text: no markup will be used.

JSON[2] is used for returning the transcript document to the client. The client is responsible for rendering the transcript in a form recognizable to users. The JSON form maps naturally to the latest draft defining the Timed Text Markup Language (TTML)[3]. The JSON form is described below along with datatype definitions. Following the JSON form is an example of the TTML form, along with a description of the mapping between the two.

The `body` property value is comprised of an array of individual timecodes, of which there should be one or more. The array must contain the timecodes in the correct sequence. The timecode format can also be returned as a response when retrieving an individual timecode, without the accuracy measure or the start time, which are only returned as information for the client from the server. Thus, each timecode submitted is comprised of an identifier, and the phrase text itself.

```
{
```

[1] OpenID specification: `http://openid.net/`

[2] JSON specification: `http://json.org/`

[3] TTML Candidate Recommendation 23 February 2010: `http://www.w3.org/TR/2010/CR-ttaf1-dfxp-20100223/`

```
            “id”: <documentID>,
            “lang”: <languageCode>,
            “state”: <state>,
            “body”: [
                  {
                        “id”: <timecodeID>,
                        “accuracy”: <float>,
                        “begin”: <timeExpression>,
                        “p”: <string>
                  }
            ]
   }
```

Definitions of data types used above:

```
   <documentID> := <string> # opaque to application
   <timecodeID> := <string> # opaque to application
   <timeExpression> := “ <offsetTime> “
   <offsetTime> := <timeCount> <fraction>? <metric>
   <timeCount> := <digit>+
   <fraction> := ‘.’ <digit>+
   <metric> := ‘h’ | ‘m’ | ‘s’ | ‘ms’ | ‘f’ | ‘t’[4]
   <state> := “ ‘low’ | ‘high’ “
   <string> := “ <chars> “
   <languageCode> := “ <ISO 639-1 language code>[5] <localeCode>? ”
   <localeCode> := “ - <ISO 3166-1 locale code>[6] “
```

An example TTML representation of the above:

```
   <tt
            xmlns=”http://www.w3.org/ns/ttml”
            xmlns:ttp=”http://www.w3.org/ns/ttml#parameter”
            xmlns:smep=”http://ns.spokenmedia.mit.edu/smep”
            xml:lang=”en-US”
            xml:space=”preserve”
            ttp:timeBase=”media”
            smep:status=”high”
            smep:id=”f23”>
            <head>
                  <ttp:profile>
                        <ttp:features>
```

---

[4] Meanings of metric symbols defined at: `http://www.w3.org/TR/2010/CR-ttaf1-dfxp-20100223/#timing-value-timeExpression`

[5] Official ISO 639-1 list: `http://www.loc.gov/standards/iso639-2/php/code_list.php`

[6] Official ISO 3166-1 list: `http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm`

```
                        <ttp:feature>#content</ttp:feature>
                        <ttp:feature>#core</ttp:feature>
                        <ttp:feature>#profile</ttp:feature>
                        <ttp:feature>#structure</ttp:feature>
                        <ttp:feature>#time-offset</ttp:feature>
                        <ttp:feature>#timing</ttp:feature>
                  </ttp:features>
            </ttp:profile>
      </head>
      <body timeContainer="par">
            <div>
                  <p
                        smep:id="a5"
                        smep:accuracy="0.98"
                        begin="0s"
                        end="2s">
                        sample text
                  </p>
            </div>
      </body>
</tt>
```

In descriptive terms, to transform the JSON form to TTML, each timecode should be assigned an end attribute equal to the beginning of the subsequent timecode, the final timecode being given an end attribute equal to the media duration. The array of timecodes should be enclosed in a `<div>` element. The `<head>` element displayed above should be included as is, and the `<tt>` element should enclose the `<head>` and `<body>`. There are several attributes included above both for the `<body>` and the `<tt>` elements that do not change and should be included as is. The document ID, document status, timecode ID, and timecode accuracy belong in a separate namespace specific to this protocol.

On a performance side note, the transcript should be regenerated after any accepted submissions, allowing any and all of its variant forms to be served as static files, making caching for reads a simple matter for most web servers..

Types

There are a limited number of data types for submitting and retrieving data. Most are simple types; complex types are expected in JSON form as defined below.

Parameters

Transcript is a *plain text* parameter containing the entire transcript.

Phrase is a *plain text* parameter containing an individual phrase from the transcript.

Responses

Transcript is the response when a transcript is requested in JSON form as defined in Transcript Format.

Timecode is the response when a timecode is requested in JSON form as defined in Transcript Format.

Boolean is either the value true or false as defined by Javascript.

Lock is a complex type where duration is in seconds:

```
{
        “available”: boolean,
        “duration”: integer
}
```

Success Status is a complex type:

```
{
        “success”: boolean,
        “message”: text
}
```

Note on Lock Timing

Clocks are not expected to be synchronized between client and server, so lock duration must be independent of clock times. The server should be lenient about the actual duration of a lock, notifying the client of a shorter lock duration than is actually granted to account for network delay. The client must behave in a manner consistent with its granted lock duration and assume without round trip communication that an expired lock is in fact expired. The client may falsely believe its lock has expired when it has not, but this seems to be a better trade off than the server rejecting what the client believes is a validly locked action.

Schema

The overall schema requires interrelated storage for users, roles, videos, transcripts, locks, timecodes, and phrases. Fine details of design for these are left to the implementation. A general description of required fields is provided for each level of quality.

Low Quality

Description

Machine transcription begins the transcript cycle. In some cases, the processing algorithm will generate a very accurate transcript; in others, more user refinement is required. Editing a low quality document will need as much flexibility as possible to make wide ranging changes while retaining a focus on rendering an accurate transcript in which the text matches what was spoken.

Schema

The low quality transcript data schema consists of a document ID, a video ID, document status, and a large text container. A history of revisions is maintained. The document status must be in the lower quality state.

API Methods

`GET /service/[document]/`
Description: Retrieve entire transcript document
Returns: Transcript

`POST /service/[document]/lock`
Description: Reserve document editing for one user, or renew if user already has a lock
Returns: Lock

`POST /service/[document]/release`
Description: Release edit lock before its expiration and without submitting any edit
Returns: Success Status

`POST /service/[document]/edit`
Description: Replace entire transcript with submitted text
Parameters: Transcript
Returns: Success Status

`POST /service/[document]/upgrade`
Description: Move a transcript from low quality to high quality, requiring a document lock and releasing it if successful
Returns: Success Status

Example Session

HTTP Session	Description
`POST /service/f39a/lock` `=> True, 900s (success)`	User notes a transcript in low quality state needs editing and initiates an editing session. This triggers a request for a lock.
`GET /service/f39a/` `=> [JSON document] (success)`	With successful acquisition of a lock, the session initiation concludes with acquiring the plain text of the transcript contained in a JSON response.
`POST /service/f39a/lock` `=> 900s (success)`	The user edits as needed. Notice of impending lock expiration prompts the user to renew the lock for another allotment of time.
`POST /service/f39a/edit` `=> (success)`	The user concludes editing and submits the edited transcript back to the service.
`POST /service/f39a/upgrade` `=> (success)`	Satisfied with the accuracy of the newly edited transcript, the user moves it from low quality to high quality, also releasing the lock.

High Quality

Description

After textual and timing accuracy have been assembled, the transcript is in a high quality state.

Schema

Timecodes have an ID and point to a phrase ID, which points to the most current revision of the phrase. Each phrase revisions points to the phrase ID. The document status must be in the high quality state

Additional API Methods

`POST /service/[document]/downgrade`
Description: Move a transcript from high quality to low quality
Returns: Success Status

`GET /service/[document]/timecode/[timecode]/`
Description: Retrieve a timecode and its associated contents
Returns: Timecode

`POST /service/[document]/timecode/[timecode]/lock`
Description: Reserve timecode editing for one user, or renew if user already has a lock
Returns: Lock Duration

`POST /service/[document]/timecode/[timecode]/release`
Description: Release edit lock before its expiration and without submitting any edit
Returns: Success Status

`POST /service/[document]/timecode/[timecode]/edit`
Description: Modify the contents of a timecode
Parameters: Phrase
Returns: Success Status

Example Session

HTTP Session	Description
`GET /service/f39a/` `=> [JSON document] (success)`	Reviewing a high-quality transcript, a user notes a typo and initiates an editing session. No document-scoped locking is required.
`POST /service/f39a/timecode/22/lock` `=> True, 120s (success)`	The user initiates a timecode editing session, triggering a timecode lock.
`GET /service/f39a/timecode/22/` `=> [JSON document] (success)`	Once locked, the latest revision of the timecode is retrieved.

HTTP Session	Description
`POST /service/f39a/timecode/22/edit` `=> (success)`	The user edits the phrase and submits changes back to the service, also releasing the timecode lock.

API Summary

Elements of the API specified above are reproduced in aggregate here in brief for easier reference.

`GET /service/[document]/`

Returns: Transcript

`POST /service/[document]/lock`

Returns: Lock

`POST /service/[document]/release`

Returns: Success Status

`POST /service/[document]/edit`

Parameters: Transcript

Returns: Success Status

`POST /service/[document]/upgrade`

Returns: Success Status

`POST /service/[document]/downgrade`

Description: Move a transcript from high quality to low quality, requires a lock and also releases it if successful

Returns: Success Status

`GET /service/[document]/timecode/[timecode]/`

Returns: Timecode

`POST /service/[document]/timecode/[timecode]/lock`

Returns: Lock

`POST /service/[document]/timecode/[timecode]/release`

Returns: Success Status

`POST /service/[document]/timecode/[timecode]/edit`

Parameters: Phrase

Returns: Success Status

Changes from Previous Draft (2010-03-13)

- Adaptation and mapping to TTML
- Introduction of accuracy measure

Changes from Earlier Draft (2010-03-08)

- Language modifications for more force and precision
- Removing transitional quality and associated methods
- Schema modifications to address removing transitional quality
- Note on lock timing
- JSON requires double quotes
- Introducing downgrade method
- Locking required for document up- and downgrade