BRIAR CLI Usage Tutorial

This document will give you a rundown of all the commands you need to perform biometric tasks with the BRIAR API.

Checking the Status of a BRIAR service

To see all options for BRIAR CLI commands

```
bash python -m briar -h
```

Checking the Status of a BRIAR service

To check the status of a BRIAR service running on the default BRIAR port, run the following command

```
bash python -m briar status
```

Checking the Status of a BRIAR service on a different port

To check the status of a BRIAR service running on a non-default port or address (e.g. address localhost and port 5000), run the following command

```
bash python -m briar status -p localhost:5000
```

NOTE: All briar CLI commands from here on out can be modified with the -p or --port flag to route the CLI request to that specific address

Listing the biometric databases available on the BRIAR Server

BRIAR Server backends utilize databases that store biometric signatures. These databases can be created and modified using different CLI calls. To list all current databases available on a BRIAR Server:

```
bash python -m briar database list
```

Creating a new database on the BRIAR Server

You can create a new database with named "example_database" with the following call:

```
bash python -m briar database create example_database
```

Enrolling media into a BRIAR Database

Once you have a database, the BRIAR Service will automatically extract all pertinent biometric information from a piece of media when you enroll it into the database.

You can enroll a piece of media called "demo_video.mp4" containing "John Doe" using the following command:

```
bash python -m briar enroll --database example_database --
subject-id "john_doe" --media-id "demo_video" --entry-type
"gallery" --progress
```

The "progress" flag will provide a progress bar denoting the status of the enrollment operation. It is not required.

NOTE 1: The "media-id" flag is used to differentiate between multimple pieces of media that contain the same subject. It does not need to be named the same as the input file, however can be helpful.

NOTE 2: The subject id can be text or numbers

NOTE 3: the "entry-type" flag is set to "Gallery" to denote the enrolled subject will be part of a gallery to search against

NOTE 4: A piece of media used for gallery enrollment is expected to only have a SINGLE subject in it. Multisubject media for gallery enrollment is not currently supported.

Say you want to enroll another video of "John Doe", withe the filename "demo_video2.mp4":

```
bash python -m briar enroll --database example_database --
subject-id "john_doe" --media-id "demo_video2" --entry-type
"gallery"
```

The BRIAR API will automatically collate multiple pieces of media per subject into a single searchable entity. At search time, you will not recieve two results for "John Doe", only the result with the highest match.

Fetching information about a database on the BRIAR Server

You can have BRIAR list database information about a database with named "example_database" with the following call:

```
bash python -m briar database info example_database
```

Searching a database on the BRIAR Server

Say you would like to retrieve the top 20 results for who is in video "search*video*1.mp4" and them to a file named "search_matches.json"

```
bash python -m briar search --database example_database --max-
results 20 --progress "search_video_1.mp4" -o
"search_matches.json"
```

NOTE: The output matches json file takes on the SearchReply message hierarchy (List of SearchMatchInfo jsons): ``` message SearchReply { repeated SearchMatchList similarities = 1; repeated Detection probe_detections = 2; BriarDurations durations = 3;

```
DetectReply detect_reply = 4;
ExtractReply extract_reply = 5;
```

```
BriarErrors errors = 6;
BriarProgress progress = 7;
bool progress_only_reply = 8;
```

} message SearchMatchList { repeated SearchMatchInfo match*list = 1; }; message SearchMatchInfo { float score = 1; // Match Similarity float theoretical*min = 2; // the minimum score that the algorithm can return float theoretical*max = 3; // the maximum score the algorithm can return string subject*id*probe = 4; // DEPRECATED AND OPTIONAL: subject ID of probe image (if known, since usually probes do not have known subjects) string subject*id*gallery = 5; // subject ID of matched gallery subject string entry*id*probe = 8; // the database entry id of the probe string entry*id_gallery = 9; // the database entry id of the gallery

```
string media_id = 6;        // ID of media that probe matched to
string subject_name = 7;    // Name of subject

BriarMedia face = 11;        // Image of face
BriarMedia body = 12;        // Image of Body
BriarMedia gate = 13;        // Image(s) of gait

bool uses_integer_subject_id_gallery = 14;        //DEPRECATED: FOR LEGACY
int32 integer_subject_id_gallery = 15;        //DEPRECATED: FOR LEGACY SYS

};```
```

Verifying two pieces of media using the BRIAR Service

Say you would like to produce a verification score of if the subject in "verify*media1.mp4" and the subject "verify*media2.jpg" are these same entity:

```
bash python -m briar verify "verify_media1.mp4"
"verify_media2.jpg" --progress
```

NOTE: Verification can be performed: image<->image, image<->video video<->video

Creating a database of Probes or Queries

Say you have built a gallery database, and you would like to perform analysis on many probe files at once (files: probe1.mp4 and probe2.mp4). You can build a BRIAR Database of probe files :

```
bash python -m briar enroll --database example_probe_database --
subject-id "unknown_person_1" --entry-type "probe" --progress
"probe1.mp4"

bash python -m briar enroll --database example_probe_database --
subject-id "unknown_person_2" --entry-type "probe" --progress
"probe2.mp4"
```

Performing Batch Search of a database of probes against a database of subjects

Say you have built a gallery database, and you would like to perform analysis on many probe files at once (files: probe1.mp4 and probe2.mp4):

```
bash python -m briar compute-search --search-database
"example_database" --probe-database "example_probe_database" --
output-type pickle -o "batch_search_output.pkl"
```

NOTE: For batch search, currently only Pickle file output is supported

NOTE: This pickle file is a nested list structured as such:

```
list{ { [0]probe_identifier1
[1]list_of_ordered_gallery_identifiers
[2]list_of_ordered_gallery_scores } { [0]probe_identifier2
[1]list_of_ordered_gallery_identifiers
[2]list_of_ordered_gallery_scores } }
```

Performing Batch Verification of a database of probes against a database of subjects

Say you have built a gallery database, and you would like to perform 1:1 verification on many probe files at once.

```
bash python -m briar compute-verify --reference-database
"example_database" --verify-database "example_probe_database" --
output-type pickle -o "batch_verify_output.pkl"
```

NOTE: Output of type "pickle" will result in a singular verification match matrix.