CCU (Computational Cultural Understanding) Reference Documentation

The file contains reference documentation for the CCU system. It should not be used as a tutorial or introduction to CCU. For that kind of information, please read the README file here, and the various README files in the directory for the language you are using (Python or Java) or the docker directory, if you are building a docker image.

Table of Contents:

Environment (Shell/Batch) Variables

To run CCU programs you must have one of these variables set:

- **CCU_SANDBOX**: Directory of sandbox, but only set on host machines.
- **CCU_CONTAINER**: The name of your container, but only set in containers.

To build CCU docker containers and other CCU programs, you must set this variable:

- **CCU_VERSION**: The current version of the CCU library version.

IP Addresses

The most common mistake seen so far in using this library is not using the right IP address. Here are some things to keep in mind:

If you are using containers, you must use the "real" IP address of the host machine, and not the Hololens:

- On Windows,
  - run this command to see the IP address: `python -c "import socket; print(f'IP is {socket.gethostbyname(socket.gethostname())}')"` If that does not work:
  - run ipconfig, and use an IP4 address from there. I used the Ethernet adapter Ethernet address which was the first one listed. there were also IPs for a vEthernets (including one for WSL) and Ethernet 2, but none of those worked for me.
- On a Mac, select the apple icon and then select "Networking" and then use the IP4 address on that dialog.
- On Linux (?and BSD?) run ifconfig and use an IP4 address from there.

If nothing else works, try running this command (even on Mac or Linux machines):

```
python -c "import socket; print(f'IP is {socket.gethostbyname(socket.gethostname())}')"
```

Remember that in modern networking your machine's IP address may change unexpectedly. For example, when rebooting after being shut down for a while if your organization uses DHCP to dynamically assign IP addresses.

If you are running all the programs directly on the host machine, with no containers involved, then you can use the 127.0.0.1 (loopback) address. But this will not work with containers.

Multiple Machine Environments

You can run CCU on multiple computers on the same network.

Pick one machine to be the "host", and run ccuhub on that machine. Also, on all machines set the host_ip value in ccu-config.yaml to be the IP address of the host running ccuhub. And make sure all agents on both machines are pointing to the correct ccu-config.yaml.

Some computers/networks have firewalls that block random ports, even within the same network, so that is a possibility, if everything else is configured correctly.

Messages

CCU messages are documented on the LDC Confluence site for CCU in the "Messages and Queues" page.

Naming

- Container names and file names are in "rotisserie style" or "dash style" like-this.
- Message names and field names are in "snake style" or "underscore style" like_this.

YAML File Reference

There is an example YAML file called example.yaml, and looking at that is the fastest way to understand YAML configuration files. You can copy that file to the name of your container and then edit it to "quickstart" the file creation process. So if your container is named sri-example, then copy example.yaml to sri-example.yaml and start there.

Fields are listed below in alphebetical format (so they are easy to find), but please put name, version, poc, and testingInstructions at the top of your file, as it flows better that way. Example.yaml is organized this way.

behaviors

List of words describing behaviors of the container. If there are none, this section does not need to exist at all.

- Slow-Start means the container takes longer than 10 seconds to start up.
- Not-Realtime means the container takes noticably longer to run than the media it is given. For audio and video media. (No example is provided.)

config

This section contains whatever configuration data the container needs for its own purposes. This section is converted to a Python dictionary or Java map and is available via the CCU.get_config() call. For example, if a container wants to configure its own timeout to 2.3 sections, it might have something like this:

```
config:
    timeout: 2.3
```

dockerCompose

This section contains the docker compose configuration for the container. This text will be used to automatically generate a complete docker compose file for the CCU system as a whole. For a container which uses CPU an example would be:

```
dockerCompose:
    sri-word-alignment:
        environment:
```

```
            - CONFIG_FILE_DIR=/app/mount_dir
            - GPU_ID=-1
        image: cirano-docker.cse.sri.com/sri-word-alignment
        ports:
            - 9097:9001/tcp
        volumes:
            - "${CCU_SANDBOX}:/sandbox"
            - "/home/ccu/ccu/minieval-1/sandbox/sri-data/data/word-alignment:/app/mount_di
```

For a container which uses a GPU an example would be:

```
dockerCompose:
    columbia-communication-change:
        deploy:
            resources:
                reservations:
                    devices:
                        - driver: nvidia
                          count: all
                          capabilities: [gpu]
        image: cirano-docker.cse.sri.com/columbia-communication-change
        network_mode: "host"
        volumes:
          - "${CCU_SANDBOX}:/sandbox"
```

The following docker compose elements are required. If you cannot use them, please contact the integration team to discuss the issues involved: - image: cirano-docker.cse.sri.com/ - network_mode: "host" - volumes: "${CCU_SANDBOX}:/sandbox"

inputs and outputs

List of queue names that the container uses (inputs) or generates (outputs).
Under each queue name is a list of messages that it uses or creates on that queue.
Do not further describe each message type, as this is done on the "Messages and Queue" page on Confluence.

If the container only understands certain languages, then those should be listed under the asr_results message or other applicable messages. The container described below only understands English asr results. Adding a line with "– zh" would mean it understood both English and Mandarin. Languages are described in two letter codes.

Example:

```
inputs:
  - AUDIO_ENV
    - asr_result
      - en
outputs:
  - RESULT
    - age
    - sex
    - face_bbox
    - location-scene
    - people_count
    - gaze
```

name

Example: sri-example.

Because this is an SRI example, the name starts with sri-. If you are at PARC or LCC it would start with parc- or lcc- and so on.

poc

Point of contact. Who should we talk to if something goes wrong or we have questions about this analytic.

Example:

```
poc:
  email: joshua.levy@sri.com
  name: Joshua Levy
```

resources

List of words which describe resources used by the container.

- CPU means only uses CPU (and this is the default if nothing is specified)
- GPU-CPU means it will use a GPU if one is available at runtime, or CPU if not.
- GPU means it needs a GPU at runtime.

- Internet means it accesses the internet at runtime.

Example:

```
resources:
  - GPU
  - Internet
```

testingInstructions: |

A paragraph of text describing how to test this container. This should include (at least) how to run the tests, and the expected outputs. With your container and YAML file you should also include one or more pairs of jsonl files (one for input one for expected output) which test your container. These jsonl files should be referenced here.

Example:

```
testingInstructions: |
  Feeding example-test1.jsonl into the container will cause it to detect faces
  in the image and generate the following outputs: age and sex of each person
  detected; location/scene information of the image; total number of people in
  the scene; gaze information.  The example-out1.jsonl contains the result of
  feeding example-test1.jsonl into it.
```

trainingLanguage

If your container is language specific, then include the language it was trained on, or a list of languages, if it was trained on more than one. If your container is not language specific, your YAML file should not contain this field at all.

Examples (last two are identical in meaning):

```
trainingLanguage: zh
trainingLanguage: [en,zh]
trainingLanguage:
    - en
    - zh
```

latency

Please add latency information to your YAML file. This information can be in one of two forms. If provided as "add N" it means the container will respond about N seconds after receiving the data being analyzed. If provided as "times M", the container will respond about M times after the length of the data being analyzed. Both N and M are numbers, so either integers or floats.

Two examples are provided below. The first describes a container which will respond about 0.1 seconds after receiving the data. The second a container which will respond after twice the length of the media it gets. So 10 seconds, if it gets media data which is 5 seconds long.

```
latency: add 0.1
latency: times 2
```

This latency data represents your best guess as to actual latency. We understand it will not be exact – we are looking for upper bound estimates.

Accuracy data (precision and recall) if available from prior/internal evaluation.

If you have data on the precision and/or recall of your container, please provide that data in your YAML file. We understand it will not be exact:

```
precision: 0.82
recall: 0.9
```

These are numbers, so either integers or floats.

Add calibration data for emotion, valence, arousal, norm_occurrence, and change_point.

Calibration data should be supplied for positive results and negative results. Results between these numbers will be considered neutral. A separate entry will be made for each data type. For example:

```
calibration:
  valence:
    positive: above 750
    negative: below 250
```

For categorical values for emotion, there should be a confidence therhold for llr for each category. If not provided we will use 0. For example:

```
  emotion:
    anger: 0.8
    fear: 0.2
    ...
```

In this example, the container will only report on valence and emotion. For valence, values above 750 will be positive, between 250 and 750 will be neutral, and below 250 will be negative. Valences are reported between 1 and 1000. Emotion is reported as an LLR; for anger numbers above 0.8 are positive, for fear numbers above 0 are positive. If a number is not provided, zero is used by default. These numbers are floats for LLR data and integers for 1-1000 data.

version

Example:

```
version: 0.3
```

The version of your analytic. Can be any format you want, but I recommend either "two dot" or "three dot" formats (ie major-release.minor-release or major-release.minor-release.patch).

The JSONL File Format

The jsonl file format is an informal packaging of json objects into a file. In CCU it is used to record a "script" of messages so that they can be played back for testing purposes.

Each line is either: 1. A comment, which has a # as the first character in the line. 2. A blank line, which consists of nothing except for whitespace. 3. A json object.

If the line consists of a json object, then it has exactly three fields: 1. queue: A String. The CCU queue to put the message. 2. time_seconds: A Number (Float or Integer). When to put the message on the queue. This is a duration from the start of running the jsonl file. 3. message: A Python Dictionary. The message to put on the queue at the time specified.

Examples of this file format can be found in the sample-data directory, and the first one you should look at is sample-data/samples-each.jsonl. This file contains one message of each type described in the Messages.md file. Each message will be put into an approarate queue one second after the last message.

Things to remember when submitting containers.

I've been working on integrating containers, and have these requests for containers:

- Please version your containers! Tag them with both a version number and with "latest".
- Please do not version them in their names. Do not have images named xyz-analytic03, rather have xyz-analytic and then tag it with 03 (or whatever the version is).
- Thanks for including sample output for your test runs, but please remember to send jsonl files, not log files. So make these files with a "python logger.py -j output.jsonl" command, not by redirecting the output of the logger.py program.
- The name and version in your *.yaml file should match the container you send. I know this is hard to remember, especially for the version, but please try.
- Please be consistent about naming containers. If you have a container named frobitz-finder, please do not have files called frobitzer.tar.gz or frobitz_tagger.yaml, or send a directory called frobizTag, or anything else.
  These small variation is naming become a big problem when I'm trying to wrangle 17 different containers. (And that is the actual situation for CCU: 17 containers and growing.)
- Container architecture must be amd64 (which is a version of x86). You will get this automatically if you build on Linux, Windows or older Macs. However, on brand new Mac M1s, you will get the wrong architecture. Therefore you must pass the `--platform linux/amd64` option to your docker build command. If you want it to run on both architecture pass this argument, but it will make your images much larger: `--platform linux/amd64,linux/arm/v8` on some machines you might need a different arm version.

Configuring Containers

There are two ways to configure your containers in the CCU environment.

The first, is to create a subdirectory in the /sandbox directory (akso named $CCU_SANDBOX) with the same name as your container, and then put a configuration file in that directory. This should be used for configuration files which impact just your container. These files can be of any format (although YAML is prefered), and you can put other files in here (like models) if you wish. That creates smaller containers, which are sometimes useful.

The second, is to add fields to the ccu-config.yaml file which is already in the /sandbox directory. This should be used for system wide configuration options. Things that are used by mulitple containers. If you do this, please contact Joshua L. or Victor A. at SRI so that we know what your field(s) do.

Provenance: Setting trigger_id, start_seconds, and end_seconds Fields

This section describes how the CCU system will track the history of how decisions are made and how data flows through the system.

1. All messages will have a container_name field which will identify the container that produced them.
2. If a message is based on previous messages, then the trigger_id field of the newer message will be set to a list of the uuids of all the previous messages that the newer message was based on.

This provides a sort of "machine traceability". Every message has a container_name and a uuid, and all decision messages will have a trigger_id which links back to previous uuids, and so on.

3. If a message is based on a part of the conversation, then the message will have start_seconds and end_seconds fields which are set to the beginning and ending of the conversation which message is based on.

This provides a sort of "human traceability". Every human interaction has a start and end seconds, and all messages based on this interaction will have those times in them.

4. Many messages will have both a trigger_id and start/end_seconds fields.
5. Messages may have other fields that help determine provenance for those messages in more specific ways.

Why both machine traceability and human traceability? Doesn't that require twice as many fields and is twice as complex? There are several reasons: looking at UUIDs is better for computers, but looking at times in the conversation is better for people. Since some parts of provenance needs to be done by computers but other parts by people, it made sense to have both. Also, having two parallel systems added redundancy. If there was a mistake in one, the other could be used (while the first one was being fixed).

The trigger_id field

The trigger_id field is a list of strings which are uuids for the messages that your container used to create the new message. For example, if your container analysed one message with uuid "1234" in order to create a emotion message, then the emotion message should have a line like this to set the trigger_id field to a list containing one string: `emotion_message['trigger_id'] = [old_message['uuid']]`

Since the trigger_id field is a list, if your container uses two or three different messages to create one of its own, then all of them should be put in the trigger_id list.

Since all result messages are based on some kind of other message, all result messages should have a trigger_id field set.

start_seconds and end_seconds fields

Start_seconds and end_seconds fields are floating point number which are the number of seconds since the conversation has started. They are not a timestamp or a timedate, which are absolute times. TA1 containers should not have to create these times. Instead, they should be in the messages your get from the ASR or MT containers.

If you have a translation message, which contains start_seconds and end_seconds fields, and you are creating a norm_occurance message based on that message, then you should just take the start and end fields from the first message and apply them to the second:

```
norm_message['start_seconds'] = translation_message['start_seconds']
norm_message['end_seconds'] = translation_message['end_seconds']
```

These fields should be set any time a result message is based on interactions which have a start time and an end time, but this is not every result message. Also, messages with start and end fields still need trigger_id fields. They serve different purposes.

More Provenance Fields

If there is other data that is important to describing why or how your container generated a message, then please do add more fields to your message to describe how or why your container generated the message. You should document these new fields in the Confluence "Messages and Queues" page, and mark them as optional if they may not be included or if other containers generating the same message may not include them.

container_name fields

All messages will have a container_name field which is automatically set by the CCU library. You don't need to do any work to add it, but you can use it. As an example, the following command will print out a list of all the containers that generated at least one message in the CCU system:

```
python jsonlfilter.py -w container_id  output_log.jsonl | sort | uniq
```