# Snips Commands Reference Guide

**2019-09-02**

## Table of Contents



## 1. Introduction

Snips Commands is a lightweight voice recognition software designed for constrained environments (ARM Cortex-M family).

This software supports wake word detection and voice commands detection. Nominally, the software will listen for a wake word and then listen for commands to fill out an intent, but this is customizable by the developer.

This software is event based using a callback pattern. It can be implemented in a bare metal environment or with an operating system (FreeRTOS for instance).

## 2. Project assets

The Snips Commands library requires three configuration items from the developer: a wake word model, a commands model, and a dialog specification.The wake word model determines which wake word to listen for, and the commands model determines which commands can be recognized. The list of recognizable commands is called the **vocabulary**. The dialog is tied to the commands model as the dialog depends on the vocabulary.

The wake word model, the commands model, and the dialog specification will be auto-generated for the developer. These assets are located in the `snips_assets` directory as follows:

```
snips_assets/:
    commands_model/  dialog/  snips_assets.h  wake_word_model/
```

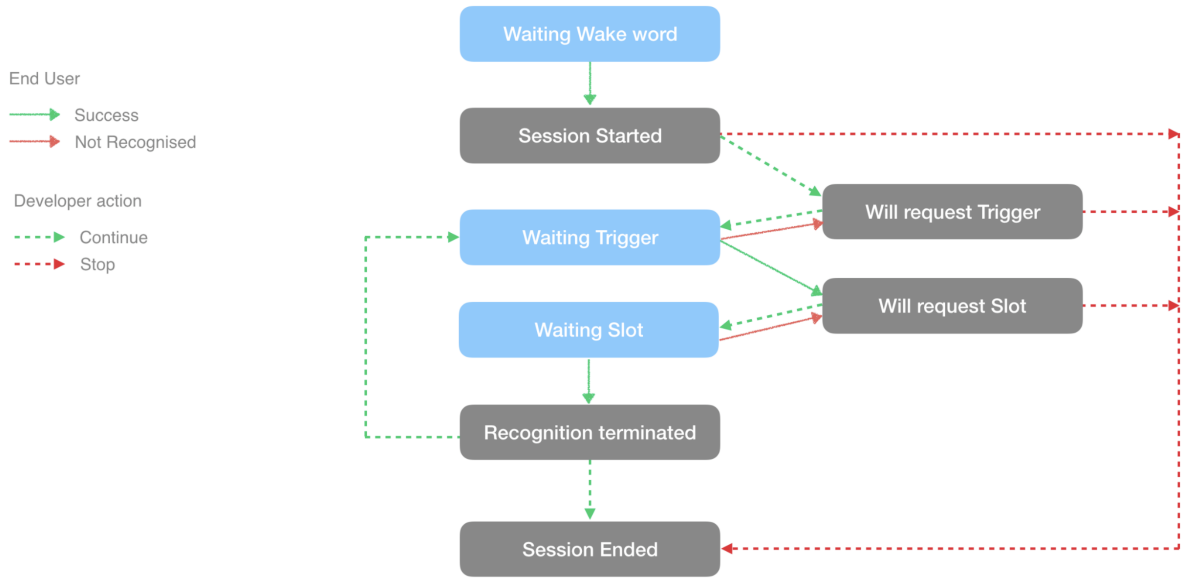
The wake word and commands models are provided as binary blobs both as a file (with the `.snp` extension) and as a C array (the `.c` and `.h` files).

## 3. Session flow

A dialog is made up of three concepts: intents, slots, and triggers. An **intent** is the final output of an interaction, and is composed of a list of slots. A **slot** is like a variable that can take one value out of a list of values from the vocabulary. And a **trigger** is a command from the vocabulary that will start the process of filling in an intent.

For an example we can consider a washing machine. A possible intent could be a description for the kind of wash the user would like. One slot could be the temperature of the wash, and the values could be "hot", "cold", and "warm". Another slot could be the intensity of the wash with values of "heavy" and "light". A trigger could be "warm wash", which would cause the Snips Commands state machine to request only the intensity slot because the temperature slot is implicit. Another trigger for this intent could be "custom wash" where none of the slots are implicitly filled in and must be filled in via interaction with the user.

The state machine of Snips Commands is described below:




A "session" is defined as every state except for when waiting for the wake word. Note that any of the callbacks may return `SnipsSessionStop`, at which point the state would return to the wake word state. The "waiting for slot" state may be skipped if the intent has no slots.

Also note that in either of the `Waiting Trigger` or `Waiting Slot` states, the user may request to cancel the session, in which case the library would jump to the `Recognition Terminated` state.

# 4. C API

## 4.1 Session callbacks

First and foremost, you will need to provide the session callbacks. Most of these will return either `SnipsSessionStop` or `SnipsSessionContinue` which will determine whether the session will continue or will be aborted. This is useful if you want to stop trying to get a slot or trigger after a certain number of tries.

```
typedef struct SnipsSessionCallbacks {
    /**
     * Called when the session is started, either because a wake word is detected or because the
     * function `snips_commands_start_session` is called.
     */
    SnipsSessionControl (*session_started)(void *user_data, const float *detection);
    /**
     * Called when a trigger is expected, either after a wake word is detected or after a trigger is
     * misunderstood.
     */
    SnipsSessionControl (*will_request_trigger)(void *user_data, uint32_t try_count);
    /**
     * Called when a slot value is needed. You will need to somehow ask the user the question
     * corresponding to this slot. A default TTS of the question in your intent should have been
     * provided.
     */
    SnipsSessionControl (*will_request_slot)(void *user_data, const SnipsSlotConfig *slot_config,
                                             const SnipsIntent *partial_intent, uint32_t try_count);
    /**
     * Called when the recognition is done, either successfully or unsuccessfully.
     */
    SnipsSessionControl (*recognition_terminated)(void *user_data, SnipsTerminationKind reason,
                                                  const SnipsIntent *intent);
    /**
     * Called when the session is ended, either because `snips_commands_stop_session` is called
     * or because one of the callbacks returned `SnipsSessionStop`.
     */
    void (*session_ended)(void *user_data, SnipsSessionEndedReason reason);
    void *user_data;
} SnipsSessionCallbacks;
```

## 4.2 Audio callbacks

You may also provide the audio callbacks, which will be used to indicate when a break in audio input is allowed. This is useful for any audio playback you may wish to do.

```
typedef struct SnipsAudioCallbacks {
    void (*should_start_listening)(void *user_data);
    void (*should_stop_listening)(void *user_data);
    void *user_data;
} SnipsAudioCallbacks;
```

### 4.3 Snips Commands

Besides the callbacks, there are only a handful of functions in the Snips Commands API. The most important one is initialization, where you provide the wake word and commands models, the dialog specification, your callbacks, and blocks of memory in which the models will run.

```
/**
 * Initialize the Snips Commands object.
 * The wake word and commands models are provided here, as well as the dialog struct, which is
 * auto-generated. The working memory should be 4-byte aligned, and the size should be the maximum
 * of the `<MODEL>_MEMORY_LEN` macros in the generated models.
 * The slow memory is optional, and can be set to NULL if not provided.
 */
SnipsResult snips_commands_init(SnipsCommands *cmd,
                                SnipsModel wakeword_model,
                                SnipsModel commands_model,
                                const SnipsDialog *dialog,
                                SnipsSessionCallbacks callbacks,
                                uint8_t *working_memory_fast,
                                uint32_t working_memory_fast_size,
                                uint8_t *working_memory_slow,
                                uint32_t working_memory_slow_size);

/**
 * Process a chunk of audio. The audio should be 16kHz 16-bit mono PCM, and the chunk should
 * have a length of 20 ms or equivalently 320 samples.
 */
SnipsResult snips_commands_process_audio(SnipsCommands *cmd,
                                         const int16_t *audio,
                                         uint32_t audio_len);

/**
 * Sets the optional audio callbacks, which are called to indicate when audio input collection
 * should be enabled and disabled.
 */
void snips_commands_set_audio_callbacks(SnipsCommands *cmd, SnipsAudioCallbacks callbacks);

/**
 * Set the timeout (in milliseconds) for triggers and slots.
 */
void snips_commands_set_timeout(SnipsCommands *cmd, uint32_t timeout_ms);

/**
 * Starts the library by calling `should_start_listening` to begin streaming audio.
 */
void snips_commands_start(SnipsCommands *cmd);

/**
 * Manually start a session without waiting for the wake word detection.
 * This will cause the `session_started` callback to be called with `detection` set to NULL.
 */
SnipsResult snips_commands_start_session(SnipsCommands *cmd);

/**
```

```
 * Manually stop the session.
 * This will cause the `session_ended` callback to be called whith `reason` set to
 * `SnipsSessionEndedManually`.
 */
SnipsResult snips_commands_stop_session(SnipsCommands *cmd);
```

# 5. The Dialog Configuration

The structure of the session flow is determined by the `SnipsDialog` structure passed to `snips_commands_init`. This object determines the intents, including which triggers are valid and which slots are needed.

The definition of the `SnipsDialog` structure is found in `snips_dialog.h`. The structure has three main parts; the list of valid commands, the cancel command, and the list of intents. The list of valid commands is pointed to by the `commands` pointer, and is determined by the corresponding commands model. All variables of type `SnipsCommandIdx` are simply indexes into this list. The cancel command such a variable; if it's non-zero, it's an index into the commands list indicating which command can be used by the user to cancel a session.

The list of `SnipsIntentConfig` objects determines the available intents. Each intent has a list of `SnipsSlotConfig` objects and a list of `SnipsTrigger` objects. Each `SnipsSlotConfig` object determines a slot for an intent, and simply lists the commands that are valid values for the slot. The `SnipsTrigger` objects determine which commands can be used to initiate the intent. The trigger indicates a command as well as a list of slots; this way different triggers can have different required slots.

Most of the time, a static instance of the `SnipsDialog` will be automatically generated from a higher-level specification, so there is typically no need to create an instance manually.

# 6. Specifications

## 6.1. System Requirements

The Snips Commands library requires two blocks of memory: one "fast" block (e.g. tightly-coupled memory) and one "slow" block. The exact memory requirements are define in `snips_assets.h`.

**The provided models (the wake word model and the commands model) as well as the `SnipsCommands` instance must be aligned to 8 bytes.**

## 6.2. Static Libraries

The ARM library is built with the following parameters:

- cortex m7
- floating point fpv5-d16 (hard ABI)
- LTO disabled

The library also requires symbols from the CMSIS NN and CMSIS DSP libraries.

## 6.3. Audio Output

The developer is responsible for any audio output including feedback sounds or asking questions related to the current intent. Snips provides some feedback sounds and text-to-speech outputs as part of the generated assets, but it's up to the developer to use these assets. An example of using these assets is provided in the portaudio example. All of the provided audio data are WAV files at 48 kHz stereo with 16-bit depth.

# 7. Examples

## 7.1. Session

Staying with the washing machine demo, an example session may appear like the following:

**User**: "Hey Snips! *Custom wash*."

**Device**: "Would you like cotton, delicate, or synthetic?"

**User**: "*Cotton*."

**Device**: "Sure. Hot, warm, or cold water?"

**User**: "*Warm water*."

**Device**: "Ok. Low, medium, or high spin?"

**User**: "*Low spin*."

**Device**: "Intent detected. Custom wash with cotton, warm water, and low spin."

## 7.2. Linux project

An example project using portaudio is provided. Once portaudio is installed, the project can be compiled like so:

```
cd example/
cmake . -Bbuild
cmake --build build
./build/snips_commands_example
```

The file `CMakeLists.txt` can be modified to use `dialog_none.c`, which implements a trivial dialog with one intent for every possible command in the vocabulary. The example project will log the callbacks to illustrate the session flow.

## 7.3. Multi intent detection

Snips Commands can chain the detections of intent. Actually, each session can contains zero or many intents.

For each **session callback**, the user can return `SnipsSessionStop` or `SnipsSessionContinue`. This will drive the session flow.

if `session_started` returns `SnipsSessionStop`, the related session ends. This will make Snips Commands act as a wake word engine.

if `recognition_terminated` returns `SnipsSessionContinue`, the session continues. Another intent detection will be possible. When the desired number of intent detection is reached, the developer can end the session by returning `SnipsSessionStop`.

## 7.4. Push to talk

The developer can start and stop programmatically a session thanks to:

```
/**
 * Manually start a session without waiting for the wake word detection.
 * This will cause the `session_started` callback to be called with `detection` set to NULL.
 */
```

```
SnipsResult snips_commands_start_session(SnipsCommands *cmd);

/**
 * Manually stop the session.
 * This will cause the `session_ended` callback to be called whith `reason` set to
 * `SnipsSessionEndedManually`.
 */
SnipsResult snips_commands_stop_session(SnipsCommands *cmd);
```

Therefore a session can be either started by a Wake Word or programmatically.